Dark Internet Mail Environment

Architecture and Specifications

December 2014

I would like to dedicate this project to the National Security Agency. For better or worse, good or evil, what follows would not have been created without you. Because sometimes upholding constitutional ideas just isn't enough; sometimes you have to uphold the actual Constitution. May god bless these United States of America. May she once again become the land of the free and home of the brave.

Ladar Levison

CONTENTS

FIGURES

OVERVIEW

This document is divided into nine sections that will introduce the reader to the Dark Internet Mail Environment (DIME) terminology, architecture, security, data formats, and protocol specifications.

PART 1 OVERVIEW: ABSTRACT

The abstract serves as a short introduction to this document.

PART 2 OVERVIEW: TERMINOLOGY

The Terminology section defines all DIME-specific terminology as well as other industry standard terms, acronyms and key words used throughout this document.

PART 3 OVERVIEW: SYSTEM ARCHITECTURE

The System Architecture section establishes the DIME architecture and can be used as guidelines for software developers to create DIME implementations that meet published standards.

PART 4 OVERVIEW: MANAGEMENT RECORD

The Management Record section describes the DNS record used to advertise DIME support, policies and functions as the cryptographic anchor for a DIME enabled domain.

PART 5 OVERVIEW: SIGNET DATA FORMAT

The Signet Data Format section describes the format of user and organizational signets data format.

PART 6 OVERVIEW: MESSAGE DATA FORMAT

The Message Data Format section describes the format of message data.

PART 7 OVERVIEW: DARK MAIL TRANSFER PROTOCOL (DMTP)

The DMTP section details the unauthenticated protocol specification for message transfers and signet lookups. It provides connection standards, command syntax, and certificate requirements.

PART 8 OVERVIEW: DARK MAIL ACCESS PROTOCOL (DMAP)

The DMAP section details the authenticated access protocol specification used within the DIME ecosystem. This protocol specification will not be released as part of the initial publication of this document.

PART 9 OVERVIEW: THREATS AND MITIGATIONS

The Threats and Mitigations section details threats, mitigation strategies, specific vector mitigation, and provides a discussion of security considerations not covered elsewhere.

PART 10 OVERVIEW: KNOWN VULNERABILITIES

The Known Vulnerabilities section will detail any known vulnerabilities as they are discovered.

PART 11 OVERVIEW: CREDITS

The Credits section provides attributions to the people that helped immensely with this document.

PART 12 OVERVIEW: REFERENCES

The References section list the references used in the creation of this document.

PART 1: ABSTRACT

This document provides the reader a detailed overview of the Dark Internet Mail Environment (DIME) and the elements required for successfully implementing DIME including the protocols and message format specification. As revealed in the Overview, this document includes detailed information covering the following artifacts: Terminology, System Architecture, the Management Record, the Signet Data Format, the Message Data Format, Dark Mail Transfer Protocol, Dark Mail Access Protocol, and a discussion of Threats and Mitigations.

This document provides an analysis of security attack vectors and a discussion covering techniques to mitigate those vectors. A secure system is only as strong as its weakest link and the authors do not pretend to have eliminated the human element. The security of DIME is dependent on the strength of the user's password and the strength of an endpoint's defenses. To the degree possible, DIME strives to create a secure system that guarantees the secure delivery of email while minimize leakage of information along the delivery path.

This document should serve as an implementation guide, and its intended audience is the system builder (software developers, integrators). It should be used as a foundation for implementing DIME. This document attempts the presentation of DIME in such a way that a development and implementation team should be able to design a system that confirms to this document's strict user-centric security requirements.

PART 2: TERMINOLOGY

The key words “MUST”, “MUST NOT”, “REQUIRED”, “SHALL”, “SHALL NOT”, “SHOULD”, “SHOULD NOT”, “RECOMMENDED”, “MAY”, AND “OPTIONAL” as used in the context of this document are to be interpreted as described below. [KEYWORD]

Key Words	Definition
MUST	This word, or the terms "REQUIRED” or "SHALL", asserts that the definition is an absolute requirement of the specification.
MUST NOT	This phrase, or the phrase "SHALL NOT", asserts that the definition is an absolute prohibition of the specification.
SHOULD	This word, or the term "RECOMMENDED", asserts that there may exist valid reasons in particular circumstances to ignore a particular item, but the full implications must be understood and carefully weighed before choosing a different course.
SHOULD NOT	This phrase, or the phrase "NOT RECOMMENDED", asserts that there may exist valid reasons in particular circumstances when the particular behavior is acceptable or even useful, but the full implications should be understood and the case carefully weighed before implementing any behavior described with this label.
MAY	This word, or the term "OPTIONAL", asserts that an item is truly optional. One vendor may choose to include the item because a particular marketplace requires it or because the vendor feels that it enhances the product while another vendor may omit the same item. An implementation which does not include a particular option MUST be prepared to interoperate with another implementation which does include the option, though perhaps with reduced functionality. In the same vein, an implementation which does include a particular option MUST be prepared to interoperate with another implementation which does not include the option (except, of course, for the feature the option provides).
EXPERIMENTAL	This word describes functionality that is in the process of development; functionality marked as experimental in this document and may broken by future specifications

The terminologies used throughout this document are defined in the following section for the reader’s benefit.

Terminology	Definition
Account Modes	
Trustful	In this mode, the server handles all privacy issues on behalf of the user requiring a user to trust the server
Cautious	In this mode, the server is used for synchronizing encrypted copies of keys and messages; this mode is designed to provide a user experience similar to that of email today while minimizing the amount of trust placed in the server

Paranoid	This mode requires the minimum amount of trust in a server specifically, the server will never have access to a user's private keys (encrypted or unencrypted)
Actors	Author, Origin Server, Destination Server or Recipient
AN	Alternate Name or Alt Name
APT	An Advanced Persistent Threat is a continuous threat with unlimited resources capable of carrying out extremely sophisticated attacks
ASCII	American Standard Code for Information Interchange
Attacker	Any unauthorized party attempting to gain access to message data (content, metadata, etc.)
Author	The cryptographic identity associated with the creator of a message
CA	Certificate Authority
CN	Common Name
DER	Distinguished Encoding Rules
Destination	Represents the recipient's service provider; the host associated with a DIME-enabled recipient domain
DIME	Dark Internet Mail Environment
DMAP	Dark Mail Access Protocol
D/MIME	Dark/Multipurpose Internet Mail Extension
DMTP	Dark Mail Transfer Protocol
DNS	Domain Name System
End-to-End Encryption	Represents two end-points that could either be a server or a specific user device
Fingerprint	
Core Fingerprint	The cryptographic portion of a user signet; SHA-512 hash
Ephemeral Fingerprint	The current signet (a combination of the author and recipient user's Full Fingerprint); SHA-512 hash
Full Fingerprint	The cryptographic and attribute portion of an organization or user signet; SHA-512 hash
Root Fingerprint	The core fingerprint for the first user signet in a chain of custody; SHA-512 hash
Host	DNS name and host IP address which could resolve to one or more servers routable over TCP/IP
IP	Internet Protocol
Key Ring	The private keys associate with a user's current and former signets
KS	Key Store; the authoritative DMTP server providing user and organizational signets
Management Records	DNS record advertising DIME support for the domain, provides policies and serves as the cryptographic anchor for the domain

MDA	Mail Delivery Agent (sometimes referenced as Message Delivery Agent)
MitM	Man in the Middle
MS	Message Store
MSA	Message Submission Agent
MTA	Mail Transfer Agent (sometimes referenced as Message Transfer Agent)
MUA	Mail User Agent (a fancy way to describe an email client)
OCSP	Online Certificate Status Protocol; used for obtaining the revocation status of X.509 certificates
OPA	Organization Privacy Agent
Organization	The service provider or corporation associated with a domain name
Organizational Domain	A domain name which excludes any subdomain; a domain plus a TLD extension
Origin	Represents the user's service provider; the host associated with a DIME-enabled user domain
PA	Privacy Agent (see OPA and UPA)
PFS	Perfect Forward Secrecy ensures that any one session key cannot be compromised if any long-term key is compromised in the future
POK	Primary Organization Key: private key capable of signing organizational signets, user signets, and outbound messages
Recipient	The cryptographic identity associated with the recipient of a message
RR	Resource Records
SR	Signet Resolver; resolves domain names to a signet
Signature	Signatures are created using Ed25519 which is an implementation of EdDSA using the Twisted Edwards curve: $x^2 + y^2 = 1 (121665/121666)x^2y^2$ This curve is birationally equivalent to Curve25519.
Signets	
Core Signet	The Core Signet represents the cryptographic information allowing the user to perform encryption and decryption; it is signed by an organizational signet
Full Signet	The Full Signet represents the Core Signet and also includes biographical and geographical user information; it is signed by a second organizational signet
Organizational signet	This signet is used by the organization to sign user signets, decrypt inbound message, decrypt 'recipient' chunk on received messages, and decrypt 'author' chunk for outbound messages before signing or 'author' chunk for bounced messages

User Signet	This signet is used to decrypt components of incoming messages, sign-outbound and decrypt inbound messages, and sign new public keys before transmitting to organization server for publication
Signet Resolver	Translates a domain name or email address into a organization or user signet similar in the way DNS translates hostnames into an IP addresses
Signet Ring	The collection of user and organizational signets retrieved and authenticated by a Signet Resolver and then stored locally
SMTP	Simple Mail Transfer Protocol
SNI	Service Name Identifier
SOK	Secondary Organization Key: private key capable of signing user signets and outbound messages
TCP	Transmission Control Protocol
TLS	Transport Layer Security
TTL	Time to Live
UPA	User Privacy Agent
User	A person or collection of people represented by a single email address; and is used throughout this document in a way that is distinct from how it commonly used in reference to access control systems

PART 3: SYSTEM ARCHITECTURE

Internet electronic mail (email) often transits through a series of independent services. Email privacy is made challenging by the need to disclose handling information to stations along this path. In addition to the usual protection of content, a design goal for secure email must be to limit what meta-information is disclosed so that a handling agent only has access to the information it needs to see. The Dark Internet Mail Environment (DIME)[1] achieves this with a core model having multiple layers of key management and multiple layers of message encryption. The system architecture modularizes functionality and that modularity permits a variety of implementation and deployment strategies, and should even permit transit over alternative infrastructure message transfer services.

The essential challenge in email privacy is protection against compromised handling agents. Simple wiretapping of transit channels is reasonably well protected against by Transport Layer Security (TLS) [TLS]. However, TLS operates over only one Transmission Control Protocol (TCP) hop and email often travels through a significant number of these hops. Every transfer agent, including the immediate submission and delivery agents associated with the author and recipient(s), may become compromised. When a handling agent is compromised, the attacker could use the breach to gain access to keys, metadata, message content or all three. Hence, mechanisms to protect each are needed. DIME builds upon email's classic distributed architecture to address these concerns: [IMA]

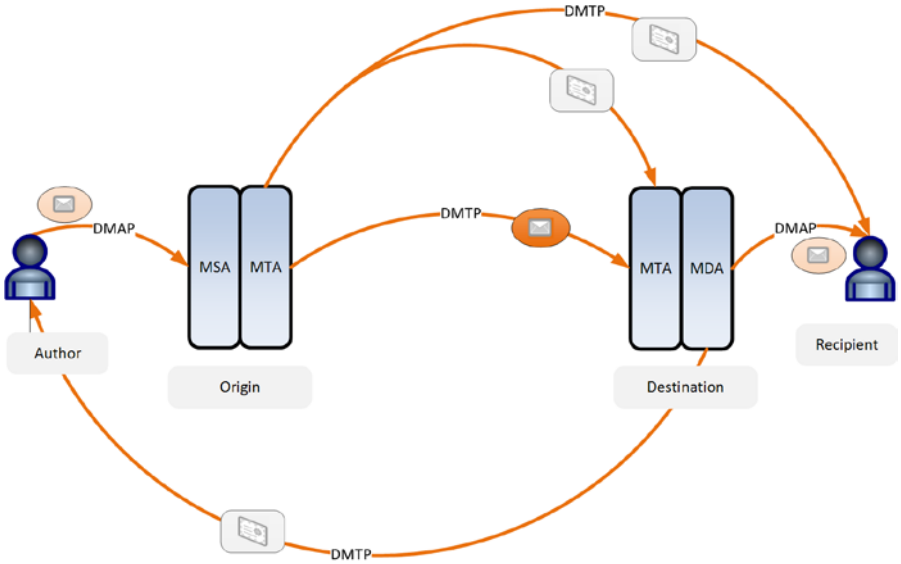

Figure 1 - Email Basic Handling Architecture

CORE OPERATIONAL DIRECTIVES OF DIME

The core operational directives of DIME are meant to simplify the adoption of a secure email system, minimize exposed data to only the data required for the system to function and only to the actors that have absolute need, and mitigate the breach possibility by unauthorized users. The directives are:

1. Automate key management, which includes: creation, rotation, discovery and validation
2. Transparently encrypt and sign messages to ensure content confidentiality
3. Ensure the system is resistant to manipulation by Advanced Persistent Threats (APTs)
4. Link security to the complexity of a user's password, and the strength of an endpoint's defenses
5. Minimize the exposure of metadata
6. Give control back to the user

At a high level, these core operational directives are achieved through the following elements:

- A handling agent only sees information about its immediate neighbors – the agent from which the message came and the agent to which it next goes. This specifically means that while the message transits the open Internet, the only visible organizational information is about the source (origin, MSA) host and destination (receiver, MDA) host. Author and recipient mailbox addresses are encrypted and then embedded within the message object. The origin host only sees the author mailbox address and the destination host and the destination host only sees the origin host and the recipient mailbox address. The origin host does not see the recipient mailbox address and the destination host does not see the author mailbox address.

- Only the author and recipient can decrypt an entire message. The origin host and destination host only have access to their portion of the encrypted envelope and to the overall message structure.

- Messages are tree structured and content encryption is per leaf with independent keys for each leaf, permitting access to individual parts of the message without having to process other parts. This is especially helpful for clients with limited resources and/or bandwidth when accessing messages held in a message store. It also permits other handling actions, such as validation of message signatures, without having to download the entire message.

- Validation of keys is accomplished without the use of a formal CA construct, and no single source of keying information is automatically trusted. The basic validation model is to obtain the key from a credible primary source and then confirm it with another pre-authenticated source. Two pre-authenticated sources are a management record signed using DNSSEC or a TLS certificate signed by a recognized Certificate Authority (CA) because each can be cryptographically traced by a Signet Resolver (SR) back to a trusted key that shipped with the SR.

- Public conveyance can be over a variety of transport services. This greatly lowers the barriers to DIME adoption.

This document provides a description of DIME's abstract network service architecture. An abstract network service architecture is distinct from any particular software design that might implement it, or specific scenarios that might

derive from it. In particular, implemented software modules might combine or separate abstract network modules. For example, the user agent and the storage service might be implemented together. Alternatively, the user agent might be split between a simple user interaction module and a remote user 'semantics' module. (This is, in fact, the usual method of providing webmail user services; the variant of webmail that has the server download code to the user's browser dynamically is actually a small operational distinction that does not affect the model.)

DIME FUNCTIONAL COMPONENTS

DIME's addition to a classic email architecture entail a few, security-related modules, replicated at the author's and recipient's sites. The basic modules are:

The basic architecture has four categories of components:

- Classic email agents
- Privacy processing agents
- Key stores and signet resolvers
- Encrypted message object

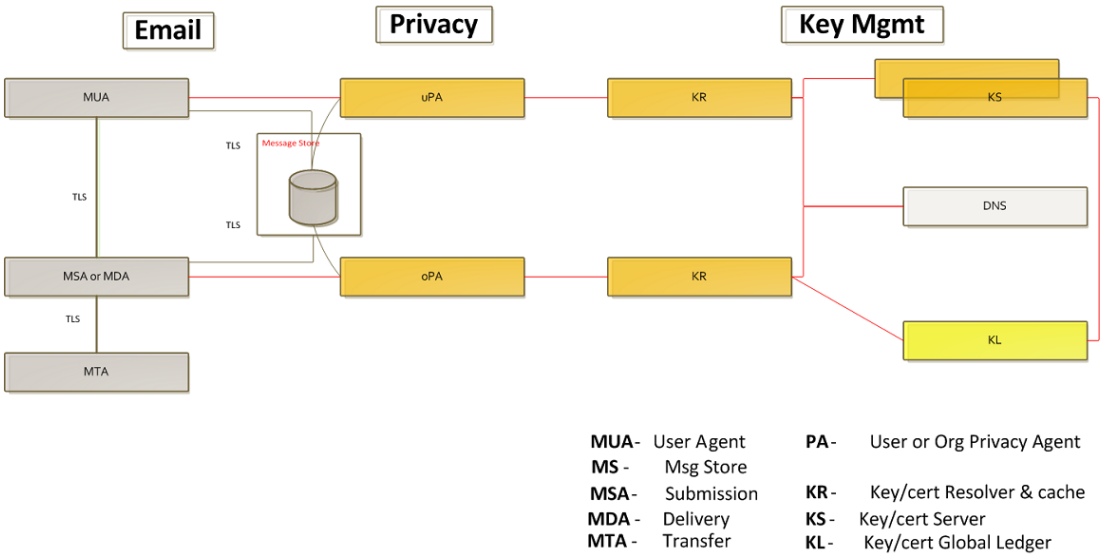

Figure 2 - DIME Functional Component

TRANSPORT

DIME can be adapted to a variety of message transport or transfer services, with the choice of channel creating trade-offs between wiretapping and traffic analysis protection, versus scaling and interoperability with existing services. Using

Simple Mail Transfer Protocol (SMTP) ensures maximum reach but also has maximum exposure. Dark Mail Transfer Protocol (DMTP) is designed to provide similar reach (if adopted) with minimum exposure.

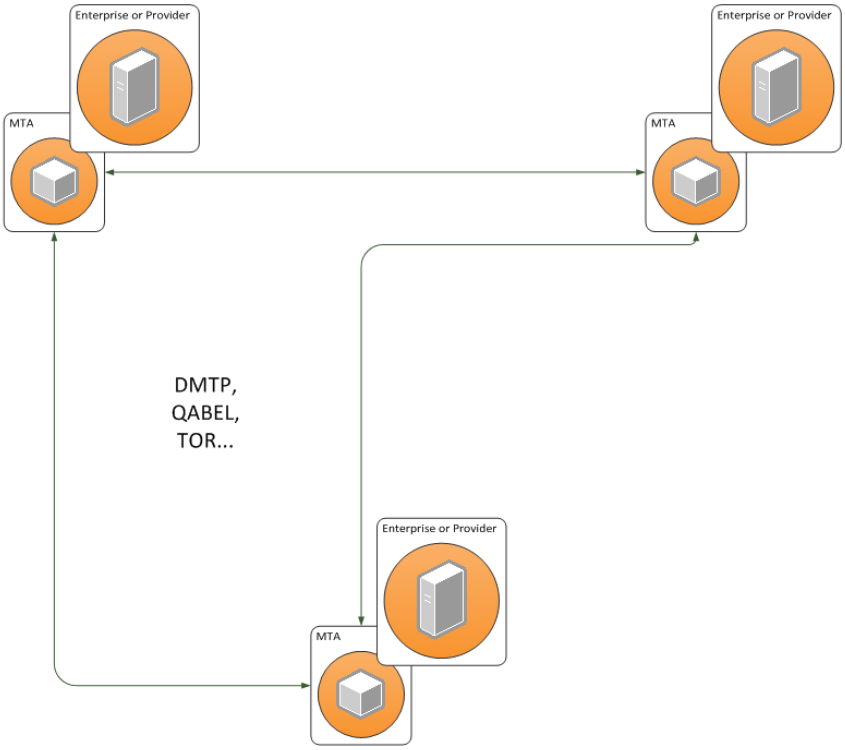

Figure 3 - DIME Transport

MESSAGE OBJECT

In terms of handling and protection, each copy of a message is between the author and one recipient.

The basic message protection model encrypts the entire message, as well as each component, called a chunk, to a distinct, ephemeral symmetric key; this includes encrypting each part of the message content (and attachments), with different keys, to permit separable handling and protection. Access to keys is limited to essential actors: author, origin (submission server), destination (delivery server), and recipient. For example, the origin needs to see information about the destination, but not about the recipient. Messages are decrypted only when the information is needed. A chunk has one or more encrypted key slots. For each actor permitted to decrypt a chunk, there is a separate slot, with its own copy of the symmetric key; the key is encrypted to the actor's signet. A chunk that can be processed by three actors will have three copies of the symmetric key associated with that chunk.

The representation of a message is as a tree-structured object:

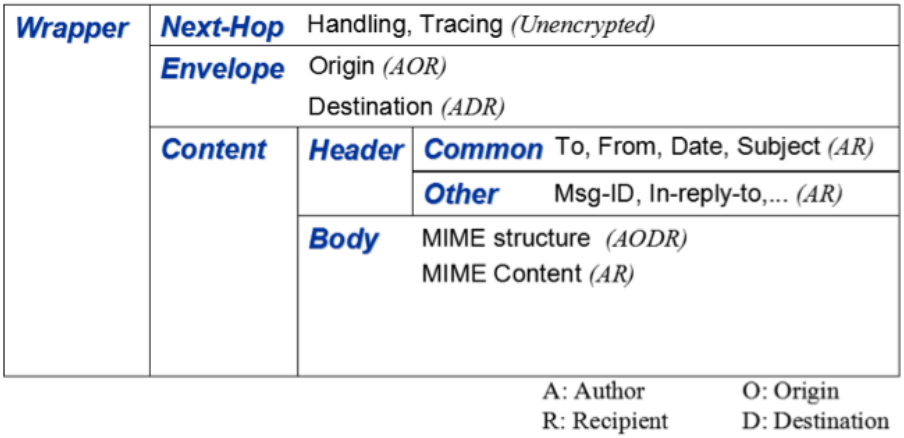

Figure 4 - DIME Message Object

The basic structure is:

- *Wrapper* surrounding the entire message
- *Next-Hop* transit handling information, in cleartext, for the currently-active transport
- *Envelope,* with Origin and Destination information, separately encrypted
- Classic message content, including a header and body (possibly with attachments) [IMF]

The envelope has further sub-structure, with each portion having independent encryption, in order to permit selectively hiding information. It contains metadata such as the envelope information traditionally used by SMTP, [SMTP] as well as privacy-related references.

CLASSIC EMAIL AGENTS

The traditional email user and handling agent functional components are present in DIME. These are:

- MUA - Mail User Agent
- MSA - Message Submission Agent

- MDA - Mail Delivery Agent
- MTA - Mail Transfer Agent
- MS - Message Store

Messages on the Message Store (MS) have the content leaf nodes encrypted, with the structure in the clear. This permits selectively accessing leaves, such as by resource-limited devices, over limited channels.

PRIVACY PROCESSING AGENTS

DIME message processing semantics are embodied in two additional network modules: Organization Privacy Agent and User Privacy Agent.

ORGANIZATION PRIVACY AGENT (OPA)

The Organization Privacy Agent interfaces between a user's email agent and the rest of the Internet. It facilitates user key management and it creates a domain-name based package around the personal addressing and content of messages. That is, it creates transit package that hides all information about the message, except what is needed for immediate handling. This is accomplished through two functions:

Certification: The authenticity of a user's signet is asserted by a cryptographic signature generated by the Organization.

Encryption: The Organization wraps and unwraps the full user email address, so that only the associated *domain* name is visible in the unencrypted envelope while in transit. The message is transmitted over a channel encrypted by TLS so that none of the message is visible during transit. The message object encryption is distinct from the channel encryption used for transmitting messages. TLS is responsible for providing perfect forward secrecy against eavesdroppers recording network communications.

USER PRIVACY AGENT (UPA)

The User Privacy Agent is the user's email agent. It facilitates traditional drafting of email, alerts user's to potential signet issues, and facilitates the automatic encryption process. This is accomplished by:

Encryption: The Recipient address is encoded for transit, to be unwrapped by the Destination host. The address is not visible to the author's origin host. Similarly the author's address is encoded so that it is visible to the recipient, but not to the destination. The terminal leaf nodes of message content are encrypted, so that only the author and recipient can decode it.

SIGNET LOOKUP SERVICES

Signet lookup services are provided by a purpose-built Key Service (KS), modeled after the Domain Name System (DNS) architecture [DNS]. A Privacy Agent (PA) makes the request to a local Signet Resolver (SR) that in turn queries the

appropriate remote authoritative KS[2]. Lookups also use tailored Resource Records (RR) in DNS to locate KS and validate the retrieved signet.

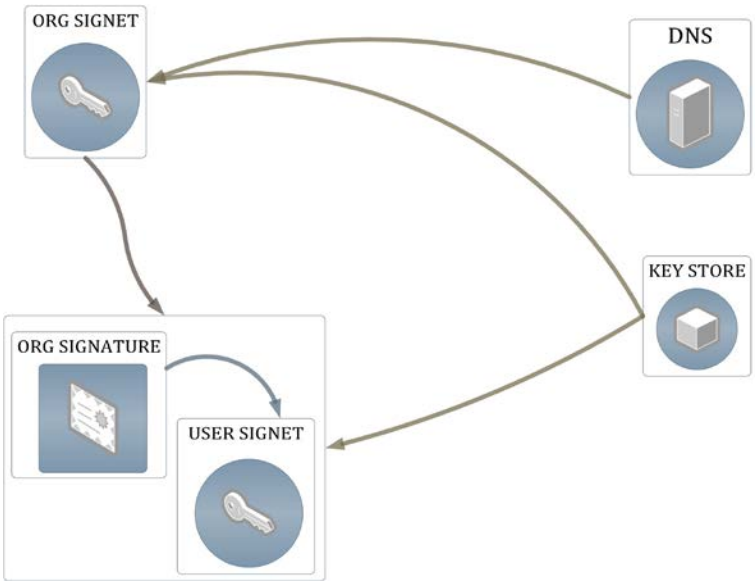

Figure 5 - Signet Lookup Services

DIME avoids using a classic Certificate Authority (CA) mechanism for validating the signet's association with a name or address. It does this with a simpler, two-level mechanism:

- An organization with a domain name certifies individual users. The organization's signet is available through (at least) two mechanisms (the management record in DNS and an authenticated KS). The combination serves as relatively independent confirmation.
- A user signet supplied by an organization pairs key information with a user address. It includes a variety of other user attributes. Unlike a classic CA-based certificate, a DIME signet is not automatically trusted. Rather the evaluator of it treats it as input. For example, the evaluator seeks at least one confirmation of the address/signet association from another source.

[1] Perhaps sending a message through this service could be called "dropping a dime"?

[2] The KS is used both for local and remote key access, using different servers.

PART 4: MANAGEMENT RECORD

Version 1

A Dark Internet Mail Environment (DIME) management record is published in the Domain Name System (DNS) system and serves as the cornerstone for a DIME enabled organizational domain. The management record advertises policies and hostname information and provides the cryptographic trust anchor for all DIME related functionality. The existence of a management record determines whether messages addressed to a particular organizational domain be sent using the DIME protocols, or as "naked" messages using the Simple Mail Transfer Protocol (SMTP). Organizational domains lacking a valid management record must be considered "legacy" and Mail Transfer Agents (MTAs) should apply any applicable policies regarding the delivery of naked messages.

The only required field, and the primary purpose for a management record, is distributing the Primary Organizational Key (POK). The POK is a public key used for organizational signing, and whose corresponding private key must be used to sign the organizational signet and, if applicable, may be used to generate the Transport Layer Security (TLS) certificate signatures used to validate DIME protocol connections. The POK may also be used, in addition to Secondary Organizational Keys (SOK), for signing user signets and outbound messages.

LOCATION

Signet resolvers with support for the optional "DIME" q-type, should search for the management record using the DIME q-type first. If the q-type is unsupported, or the request fails, a resolver must also query the target domain using the "TXT" q-type using the prefix "_dime" along with the target domain. A fully qualified domain name for management records when sending a query for the TXT resource record would be _dime.example.tld if the target mailbox was using the organizational domain example.tld.

Signet resolvers attempting to locate the applicable management record for mailbox addresses with a subdomain, such as user@sub.domain.example.tld, should use an increasingly specific search pattern. Starting with the base organizational domain, resolvers should continue by working towards the specific subdomain supplied until it encounters a management record with the "subdomain" policy field equal to the value "strict," or the resolver reaches the specific fully qualified subdomain under consideration. For the domain sub.domain.example.tld, a resolver should begin by requesting the DIME resource record for example.tld or the TXT resource record for _dime.example.tld. Unless a subdomain policy of strict is encountered, a resolver must continue by searching for the DIME record using domain.example.tld or a TXT record using _dime.domain.example.tld. As the final step in our example, a resolver should search for a DIME resource record using the fully qualified domain or a TXT resource record using _dime.sub.domain.example.tld.

If the final search does not return a valid management record, and the nearest ancestor domain returned a management record with a subdomain policy of "explicit," then a signet resolver must fail, and the target domain is considered legacy. Otherwise, if the nearest ancestor management record supplied a subdomain policy of "loose" then the values it provides should be applied to the target domain. The increasingly specific query process ensures management records associated with ancestor domains may assert control over subdomains.

TEXT RECORDS

Operational considerations must be made if the management record is published using the TXT q-type. The DNS rules regarding TXT records stipulate that individual strings have a maximum length of 255 characters. As such, management records that exceed 255 must be split across strings. For optimal compatibility, management records must not split individual fields across strings.

While most DNS resolvers allow TXT responses up to 4096 octets over UDP, a handful of non-conformant, but widely deployed DNS implementations truncate UDP responses to 512 octets (primarily Cisco PIX/IOS implementations). Organizations seeking to interoperate with the widest variety of resolvers should ensure their management records fit within 512 octets.

The authoritative DNS servers for management records should support DNS queries using TCP so that resolvers may choose to perform management records queries over TCP.[3]

SECURITY

The DIME security model depends upon the reliability and security of the global DNS system. For this reason we strongly recommended organizations use DNSSEC to prevent the manipulation of DNS responses for their domain. For management records secured using DNSSEC, resolvers must validate the DNSSEC signatures.

Verifying the signature for an organizational signet using the management record POK value is the preferred method for validating the relationship between a signet and an organizational domain. If a management record is protected using DNSSEC, no other validation paths are required. A management record protected by DNSSEC is considered a pre-authenticated verification source. Consult the signet specification for additional information regarding validation.

DNSSEC support ensures an attacker in a privileged network position will be unable to execute downgrade attacks. If the response for a management record is replaced by a XXXXXXNOTFOUNDXXXX message, a victim might send a naked message over SMTP that otherwise could have been sent surely using DIME. An attacker could also manipulate the POK and DX field values in the response to carry out a Man in the Middle (MitM) attack on the target domain.

REFRESH

If a resolver currently has a cached management record, it must never reduce the amount of time until a record is purged from cache because the "expiry" value retrieved during a refresh has been replaced by a smaller value. This rule should only be applied to the expiry value; all other fields may be overwritten by an updated management record even if resolver ignores the expiry value. This ensures an attacker with control over the DNS servers for a domain is unable to reduce the amount they must wait for cached management records to be expunged and peers to downgrade into legacy mode.

FIELDS

Management records provide information using fields, with each field being comprised of a name/value pair. The table below is provided to indicate the properties associated with each field. The table outlines which fields are required, recommended, or optional, what the type of value each field provides, and whether a default value is applied when the field is absent.

Management records should use the short version of a field name when specifying values, but may use long version. Resolvers must be capable of parsing and recognizing records with both long and short field names. Management records must use lowercase letters for field names and enumerated values, although parsers should match against these strings case insensitively. If a management record does contain uppercase characters in the field names or enumerated values, parsers should accept the input and may issue an optional warning to the user about the incorrect syntax.

A field is properly defined as a name followed immediately by an equal sign (ASCII value ox3d) and the desired value. A field ends when the first space (ASCII value ox20) or semicolon character (ASCII value ox3b) is reached. A field also ends if the end of the management record is reached; the final field is the only one which does not require a terminating character. Tab characters (ASCII ox??) must be treated as spaces, and extraneous whitespace, if discovered, must be ignored.

A single field definition must not span multiple TXT record strings. This means every string must end with either a space or semicolon, with the exception of the last one. This requirement ensures resolvers that concatenate TXT strings together are the same that insert whitespace between strings.

Fields may defined in any order. Fields which allow multiple values must specify every value as a fully formed field, using the complete name/value sequence defined above. If an additional value is encountered for a field which does not support multiple values, a resolver must use the first valid field value encountered. Resolvers that encounter additional instances of unique fields may optionally warn users, or silently ignore it.

FIELD DEFINITIONS

Key	Short	Disposition	Multiple	Type	Default (Where Applicable)
primary	pok	Required	Yes	String	
tls	tls	Recommended	Yes	String	
version	ver	Optional	No	Numeric	1
expiry	exp	Optional	No	Numeric	30
syndicates	syn	Optional	Yes	String	
deliver	dx	Optional	Yes	String	
policy	pol	Optional	No	Enumerated	mixed
subdomain	sub	Optional	No	Enumerated	mixed

PRIMARY

Provides the POK, or public key used to validate signatures supplied by the organizational signet. Signet resolvers must ensure the organizational signet it retrieves provides a POK that matches one of the POK definitions in the management record. A resolver must also ensure the two signatures supplied with an organizational signet are validated using the matching POK value. If an organization rotates their organizational signet, they should include the POK for both signets in the management record until the original signet has expired. A POK may also be used to sign user signets and outgoing messages. When attempting to validate organizational signatures created with a valid POK, consumers may choose to rely on the management record for validation instead of the full organizational signet.

The POK value must be exactly 43 characters in length, and must be a valid Base64 string without padding characters, otherwise a management record must be rejected and the user must be notified of the fatal error. The POK value once decoded must be 32 bytes and contain a valid Ed25519 public key. Resolvers must reject management records where the value is incorrectly encoded or does not correspond to a valid Ed25519 public key. Note that the Base64 specification traditionally calls for a single padding "=" character when 32 octets have been encoded, however management records must omit this padding character when publishing a management record.

The Ed25519 reference implementation represents public keys using compressed points in little endian form. This deviates from other standards which require elliptical curve public keys be stored uncompressed, using big endian multiprecision integers (MPI). See RFC 4880 for details regarding the MPI format, and RFC 6637 for details regarding point encoding formats.

TLS

The certificates for DMTP hosts may be validated using the signatures provided in a TLS field. The value for a TLS field represents a 64 byte Ed25519 signature generated from a certificate in Distinguished Encoding Rules (DER), or native binary, format. The 64-byte signature must be encoded using Base64 without padding characters. Resolvers must reject management records with TLS values that are not 86 bytes in length, or if the value does not contain a valid Base64 string without the two trailing pad characters removed.

If one, or more, values are provided in the DIME management record, then the TLS certificate received while connecting to a DMTP host must match one of the provided values. This requires organizations using multiple certificates to secure DMTP connections provide a signature for all, or none, of the certificates being used. If a resolver finds a TLS field in the management and encounters a DMTP host using a certificate that does not match any of the provided signatures, it must cleanly shutdown the TLS connection and disconnect, treating the connection attempt as a failure. If the threshold for connection failures has not been reached, and additional hostnames are available the consumer should continue onto the next host until it discovers one with a certificate matching one of the TLS field signatures.

For management records protected by DNSSEC a matching TLS field signature must be available, otherwise consumers must reject for self-signed certificates, or certificates signed by unrecognized Certificate Authorities (CA). If the

management record is not protected using DNSSEC, but still provides a TLS field signature, a consumer should ensure the certificate matches one of the available signatures first, followed by the validation rules required by TLS v1.2 [RFC 5246].[4]

SYNDICATES

The syndicate field provides alternate domains where signet information may be retrieved for validation purposes. The value must be a valid CNAME and not an IP address. If an IP address is provided for a value, a consumer must reject the management record and notify the user. If the value is a valid domain name, but does not resolve, a consumer should ignore the field and proceed. Implementations may optionally warn users when deliver field values do not resolve properly.

DELIVER

Provides the fully qualified domain name for authoritative signet lookups, and for delivering D/MIME messages. The value must be a valid CNAME and not an IP address. If an IP address is provided, a consumer must reject the entire management record and notify the user. If the value is a valid domain name but does not resolve, a consumer should ignore the field and proceed. Implementations may optionally warn users when deliver field values do not resolve properly.

If multiple values are provided and one of values does not properly resolve to an IP address, or the consumer is unable to connect over port 26 it must continue to the next deliver field value. If three of the deliver field values fail, or if a management record does not specify the deliver field, consumers must rely upon the MX record for the target domain. See the DMTP specification for connection details.

VERSION

The DIME management record syntax version, which controls how a record should be parsed and validated. As of this writing, only a single value for this field, "1," is valid and is the implied version number if the field isn't specified. Any value other than "1" must result in a resolver error, unless the version provided has been implemented by the resolver.

EXPIRY

The number of days a management record must remain cached, and a domain should be considered DIME enabled after removing the management record. If a resolver is unable to refresh a management record after the number of days provided by this field value, then the cached record should be expunged and the domain should revert to legacy mode. The TTL value provided with the DNS query should determine how often a resolver attempts to refresh a cached management record.

Provides the policy applied when transferring messages between origin and destination domains. Message acceptance and delivery must conform to the advertised policy when one of the organizations involved is DIME-enabled. D/MIME messages must be rejected when the delivery does not conform to the policy, or if the organization does not have a valid management record. In the absence of an explicitly defined policy field, resolvers must apply a default policy of mixed. The table below illustrates the appropriate outcome for a message between two domains with each of the possible policy dispositions.

Sending Domain \ Receiving Domain	Legacy	Experimental	Mixed	Strict
Legacy	Naked	Naked	Naked	X
Experimental	Naked	Dark/Naked	Dark/Naked	Dark
Mixed	Naked	Dark/Naked	Dark	Dark
Strict	X	Dark	Dark	Dark

Figure 6 - Policy Dispositions

Experimental. This organizational domain will be sending both D/MIME and naked messages. Destinations with policies of experimental or mixed should accept both, while those with a policy of strict must reject naked messages. If a domain does not have a management record available then this organization supports the delivery of naked messages. Organization with a policy of experimental should publish valid signets for all DIME enabled addresses, or the appropriate error code for valid addresses which are not yet DIME enabled. Senders with a policy of mixed or experimental may choose to deliver naked messages if they encounter an experimental policy for the destination and the recipient addresses does not have a valid signet available.

Mixed. This domain will only send D/MIME messages to domains with a policy of mixed or strict. For domains without a management record, this domain will deliver a naked message. For experimental domains, the message will be sent using D/MIME if a valid signet is available, and as a naked message for addresses which result in the appropriate error code to indicate the recipient is not DIME enabled. Only D/MIME messages must be accepted from other domains with a policy of strict or mixed, while domains with an experimental policy should be allowed to deliver D/MIME and naked messages. Naked messages will be accepted from and delivered to domains without a valid DIME management record.

Strict. This domain must only accept D/MIME messages and must send D/MIME messages. If a strict domain encounters a recipient domain without a management record or if signet resolution fails, the send attempt will fail.

SUBDOMAIN

Determines whether a resolver should apply the management record to subdomain addresses. In the absence of an explicitly defined policy field, resolvers must apply a default policy of mixed.

Explicit. Subdomains must provide a management record and organizational signet. The absence of management record results in the subdomain being classified as legacy.

Mixed. Subdomains may supply a management record and organizational signet, which are used instead of the parent domain. If the management record is missing, the values and organizational signet of the parent should be applied to the subdomain address.

Strict. Subdomains must always use the management record and organizational signet of the parent domain which supplies this value.

EXAMPLES

At a minimum, all legal DIME management records must provide a POK. All other values are optional, with default values used in their absence. A simple DIME management record would look like:

```
sig=NTE1NTU5Nzg1MjZBNDEzMDRENkI1QTQ0NEQ2QTUxMzA
```

DIME management records should specify values for the recommend field TLS, so that resolvers may validate DMTP connections using the TLS provided upon connection. A simple DIME management record which provides a signed TLS value might look like:

```
tls=YjRjYWZlMTBlMjE2NzZjNDY0MjM1NGNiYTEwMjI0M2YwMTM4Yjc5OGFmNjg5ZWE1MTU3NmE3N2I
5MDEzNzkwYQ sig=bWwyYzIzNTBnbTFibHVrOHVrazFkb2F2MzRtbDJkY2E
```

Finally, a DIME management record with all of the fields specified might look like:

```
ver=1 pok=MmprdmRjaWtjOTYyZDllMGhrOGNhMTRsZmoyamt2ZGM pol=mixed
syn=mirror.example.tld
tls=QUYxRjA0MkZDMjQ0OUEzOUJENEE5QkU2MTdENDM3OUVEQTI1QjQ1REYwODEwODE2ODlGMUE2Q0U
1MjQ3M0Y2Mw dx=dmtp.domain.tld ttl=1776 exp=30 sub=strict
```

[3] Should we add a field like "tcp=true" which can indicate to resolvers that they should perform the query using TCP?

[4] DNS-Based Authentication of Named Entities (DANE) [RFC 6698] provides similar functionality, but lacks widespread deployment. The primary functional difference is DANE records only support the publication of complete certificates, a public key or a hash value. DIME uses the POK and cryptographic signatures to validate certificates.

DANE also requires (by specification and not function) the deployment of DNSSEC, while DIME decided to classify DNSSEC support as a strong recommendation.

PART 5: SIGNET DATA FORMAT

This specification details the format and semantics for the signet data format. The Dark Internet Mail Environment (DIME) uses the signet data format to transfer cryptographic information for use in encryption and signing operations. A signet carries with it signatures which must be evaluated by the consumer when determining whether to accept the validity of a signet for an organization or user identity. In addition to the required cryptographic information, a signet may be used to advertise information about the signet owner, or information used to facilitate other non-cryptographic functions commonly supported by DIME implementations.

The signet data format requires a small number of fields containing public keys and signature data. This specification defines additional fields used to optional DIME functionality. The signet data format also allows for an unlimited number of undefined fields. Undefined fields provide an arbitrary name and data value that may be recognized by DIME protocol extensions, or simply to carry arbitrary data for experimentation or use by non-DIME functionality.

ENDIANNESS

The signet data format is a binary schema, which relies on numeric values to convey information and facilitate parsing. The binary values defined by this specification will always use network byte order, which is defined as a big endian representation, requiring the most significant byte to be stored in the smallest address, and the least significant byte be stored in the largest address. Implementations running on little endian systems will need to convert the values to ensure proper processing. [v]

HEADER LAYOUT

All signets must start with a 5-octet header. The first two octets are a magic number that indicates the signet data format. Signets conforming to this specification must set the magic number appropriately from the following table:

Magic Number	Label
1215	User Signet Signing Request
1776	Organizational Signet
1789	User Signet
1952	Organizational Private Keys
2013	User Private Keys

The remaining 3 octets are used to provide the length of the signet data in binary form, without including the 5 octets used by the header. Since the length parameter is 3 octets, signets have a maximum size 16,777,220 octets or 16,777,215 for the fields plus 5 octets for the header. Signet resolver and parser implementations conforming to this specification must be capable of handling signets up to their maximum possible size.

Signet resolver and parser implementations conforming to this specification must accept signets with magic numbers as detailed above and must the parser must generate an error if it encounters any unrecognized magic numbers. The only time a magic number will change is if an update is large enough to warrant.

While parsing signets conforming to signet data formats, a parser must ignore any fields with unrecognized type codes. The unrecognized field types should have a 2 octet value length parameter immediately after their single octet type parameter. This scheme will allow parsers to skip the unrecognized fields and continue processing the signet. Backwards compatibility is guaranteed to use this scheme for all magic numbers defined within this specification. If the magic number is not equal to those defined in this document, parsers must reject the signet unless they implement a newer signet format specification version. The addition of the new magic number will not guarantee backwards compatibility.

FIELD LAYOUT

This specification provides details for a number of defined fields, whose name is provided by this specification and omitted from the signet layout. The signet data format allows for an unlimited number of undefined fields that include an arbitrary name and value.

DEFINED FIELD LAYOUT

A single octet is used to provide the field type. For fields defined by this specification, a variable layout is used which is specific to the defined field type. Following the type code is a single octet used field specific flags, and is only present for the fields that require it. Variable length value fields must always provide a length parameter that will be 1, 2 or 3 octets. The size of the length parameter is determined by the field type. For defined fields with fixed length values the length parameter is omitted, as the value must always be the length required by the field definition. For defined fields holding a variable length value, the minimum value size is 0, while the maximum value size is determined by the number of octets used by the length parameter. Implementations must be able to handle a length parameter value of 0, which is functionally equivalent to the field being omitted. Fixed length fields must always provide values matching the length associated with the field type.

```
           [ Min ] [ Max ] [ Optional ]
[ Type   ] [  1  ] [  1  ] [          ]
[ Length ] [  1  ] [  3  ] [ XXXXXXXX ]
[ Value  ] [  0  ] [  ~  ] [          ]
```

UNDEFINED FIELD LAYOUT

The undefined field layout has been designed for flexibility, allowing implementations to create fields with variable length names and values. Undefined fields are indicated by the single octet type parameter, which will indicate undefined fields using the value 32. The type parameter is followed by the length of the name encoded in as a single octet parameter. The name value follows the length parameter and must be comprised of valid UTF-8 characters. Names must always be at least 1 character in length, and should always begin with a capital letter. Name values must be constructed without the use of whitespace characters, and may use up to 255 octets. The name parameter is followed by the value length parameter, which is provided using 2 octets. Implementations must accept undefined fields with a

value length of 0. The maximum length of an undefined field value is 65535 octets. Values may include binary data, with octets of any possible value, and signet parsing implementations must be capable of handling binary data in undefined field values without issue.

Implementations should be capable of handling invalid undefined fields where the length of the name is 0, or where the name value includes invalid UTF-8 sequences or whitespace characters. Implementations responsible for signet creation must remove these invalid undefined, and consumer implementations must never use the value of an undefined field with an invalid name for any purpose. Implementations may choose whether to provide users with a warning when invalid name values are encountered.

```
            [ Min ] [   Max   ] [ Optional ]
[ Type   ] [  1   ] [    1    ] [           ]
[ Length ] [  1   ] [    1    ] [           ]
[ Name   ] [  0   ] [  255    ] [           ]
[ Length ] [  2   ] [    2    ] [           ]
[ Value  ] [  0   ] [ 65536 ] [           ]
```

FIELD ORDERING

All fields within a signet must be sorted according to their single octet type parameter, and appear in ascending order. If a consumer encounters a signet that does not conform to this field order, or if a signet includes a unique field multiple times, it must be considered malformed and rejected.

Undefined fields may appear in any order, however the recommended ordering for undefined fields is to sort them alphabetically based on the name parameter. Because the alphabetical ordering is optional, a consumer must assume that undefined fields will be unsorted and act accordingly when searching for an undefined field with a desired name value. Implementations that encounter an undefined field with an identical name value multiple times should use first occurrence of the field by default. Any remaining fields with the desired name value should be ignored unless an extension is multiple values.

ENCODING

As of this publication, the default encoding scheme for user and organizational signets is Radix-64 also known as ASCII armor. [PGP]

SIGNET FIELD INTRODUCTION

The first column provides the number value used to identify fields of a particular type, followed by the English language name for the field. The status column is used to indicate which fields must be provided by a valid user and organizational signet, along with which fields should be defined, and those that are optional. The flags column indicates whether a single octet for providing flags follows the type octet. For fields listed as a fixed type, the length column provides the length for values. The fields listed as using variable type, the length column provides the length for values. Undefined fields are listed, but do not utilize the same layout as defined fields.

Field ranges for signets are defined in the following table:

Signet Field Range	Description
1 – 15	Encryption Fields (1-3 Organization, 4 – 15 Reserved | 1-6 User, 7 – 15 Reserved)
16 – 92	Common Short Fields
93 – 159	User or Organization Specific Short Fields
160 – 199	Common Long Fields
200 – 250	Organization and User Specific Long Fields
251	Undefined Fields
252	Image Field
253 - 255	Signet Termination Fields

Implementations encountering signets with a magic number matching this specification must be capable of separating semantics from syntax. To ensure this, the layout for an unrecognized field is determined by the numeric type code, with the layout governed by the ranges below. Consumers must be capable of parsing the unrecognized fields syntactically, by advancing over, and semantically ignoring any fields it does not recognize or support.

The maximum size for a field value is determined by the number of octets allocated to hold the value length. The ranges below will dictate the number of octets used by a field to store the length of its value. The maximum size for a defined field is determined by which range it is associated with. Fields outside of the predefined ranges either have a fixed length, are cryptographic and specified elsewhere in this specification, or reserved. If a reserved field is encountered with a value, the signet must be reject. The predefined ranges associated with 1 and 2 octet defined fields capable of variable length values:

Signet Field Range	Length Parameter	Min	Max
16 – 159	1 octet	0	255
160 – 250	2 octets	0	65,535
251	1 for name / 2 for value	0	255 / 65,535
252	3 octets	0	*variable**

* The maximum size of an image field is determined the cumulative size of a signet's other fields. The image field must never overflow the 3 octet signet length. Consumers must ensure the cumulative length of the fields does not overflow the overall length and reject any signet which does. Users must be notified of the error, and informed the overflowing signet could be malicious.

RESERVED FIELD TYPES

The signet field types for all organizational signets conforming to this specification, will be a big endian numeric value in the range 0x01 through 0x03 or 0x0F through 0xFF. The valid signet field types for user signets will be a big endian numeric value in the range 0x01 through 0x06 and 0x0F through 0xFF. Parsers encountering a signet with a value for any type code that falls outside of these ranges must reject the signet as invalid. Parsers adhering to this specification

must also be able to identify and process all of the fields described in this document. An implementation may ignore the value of any informational field, and must ignore the values for any unrecognized field types.

COMMON FIELDS

The two signet types (user and organizational signets) include fields that are common between them. The following tables list the defined signet fields that are common between the user and organizational signets.

Common fields already identified in this specification are detailed in the below table followed by a textual description after the table. These field descriptions will not be repeated in the signet-specific sections.

Any implementation must not vary the use of these common fields between the user and organizational signet formats.

	Label	Status	Multiples	Type	Length
16	Name	Optional	No	Variable	1
17	Address	Optional	No	Variable	1
18	Province	Optional	No	Variable	1
19	Country	Optional	No	Variable	1
20	Postal-Code	Optional	No	Variable	1
21	Phone	Optional	Yes	Variable	1
22	Language	Optional	No	Variable	1
23	Currency	Optional	No	Variable	1
24	Cryptocurrency	Optional	No	Variable	1
25	Motto	Optional	No	Variable	1
26	Extensions	Optional	No	Variable	1
27	Message-Size-Limit	Optional	No	Variable	1
28 – 92	Unused 1-byte Fields	--	--	--	1
160	Website	Optional	No	Variable	2
161 – 250	Unused 2-byte Fields	--	--	--	2
251	Undefined-Fields	Optional	Yes	Variable	1 + 2
252	Image	Recommended	No	Variable	3

NAME FIELD

Should provide UTF-8 string of characters containing an organization or user's preferred name. When displaying the value of this field, the label "Name" should be used.

ADDRESS FIELD

Should provide UTF-8 string of characters corresponding to the organization, or user's physical address. When displaying the value of this field, the label "Name" should be used.

PROVINCE FIELD

Should provide UTF-8 string of letters corresponding to an organization or user's province, or the principal administrative division of the signet owner's country. This is more commonly called the state, region, territory, district, or canton depending on the locale. The contents of this field should not be abbreviated. When displaying the value of this field, the label "Province" should be used, unless the client is sophisticated enough to consider the associated country and supply the appropriate colloquial term.

COUNTRY FIELD

Should provide UTF-8 string of letters corresponding to an organization or user's country. The contents of this field should not be abbreviated. When displaying the value of this field, the label "Country" should be used.

POSTAL CODE FIELD

Should provide UTF-8 string of numbers or letters corresponding to an organization or user's postal code. The contents of this field should not be abbreviated. When displaying the value of this field, the label "Postal-Code" should be used.

PHONE FIELD

Should provide an organization or user's phone number. The value must begin with a string of numbers. A semicolon terminates the numeric portion of the field. If a non-numeric value is encountered before the semicolon then a parser must ignore the field entirely. Otherwise if any valid UTF-8 characters appear after the semicolon, they should be considered the label associated with the number. The label is optional, but if it is supplied, a label must not exceed 16 UTF-8 characters and must not include any whitespace characters. If the label exceeds 16 UTF-8 characters, contains a whitespace character, or supplies an invalid UTF-8 octet sequence (aka invalid Unicode codepoint), then the label must be discarded. A consumer may use the numeric portion of a phone number field even if the label is invalid. For fields with missing or invalid labels, the default string "Phone" should be used. A sample phone number field value is:

```
4108546334;OFFICE
```

LANGUAGE FIELD

Should provide a string of characters containing an organization or user's preferred language identifier. A semicolon terminates the string, and provides an optional separator. The string which follows the semicolon should be considered a secondary language identifier and used if the preceding value is unsupported. The sequence may repeat until either the signet owner's list of preferred languages is exhausted or the length limit for the field value is reached. The final language may terminate with a semicolon, but its inclusion is optional. When displaying the value of this field, the label "Language" should be used.

The value of this field is a language tag plus an optional subtag. The subtag is typically used to indicate a country or region. The language and subtag values will be separated by a dash. [LANGUAGE] The following value would indicate the language is English and the country is the United States of America:

```
en-US
```

The value may be used to select a signet owner's preferred language. The value also provides guidance when formatting dates, times, numbers and currency amounts. When a consumer encounters multiple language identifiers it should select the first fully supported value it encounters. If none of the identifiers are fully supported, a consumer should examine the list a second time, and discard the subtag when making comparisons, considering only the language identifier. The first supported language it encounters should be selected. If this field is missing, and a signet has supplied a value for the field "Country" then its value may be considered as an alternative.

For user signets where the Language and Country fields are missing, invalid or their values unsupported, a consumer may fall back to considering the associated organizational signet using the same logic described above. If all of the described logic fails, a consumer may consider whether the organizational domain includes a country specific Top-Level-Domain (TLD), and select based on its value. If all else fails, the default value "en-US" should be used.

Consumers should use the list of recognized languages and subtags maintained by the Internet Assigned Numbers Authority (IANA) when evaluating language identifiers. [IANA-LANG]

CURRENCY FIELD

A UTF-8 string of letters which should correspond to the foreign exchange symbol associated with an organization or user's preferred form of national currency. A semicolon terminates the 'currency' string, and provides an optional separator. If a UTF-8 string follows the semicolon it should be another foreign exchange symbol. The sequence may repeat until either the signet owner's list of preferred currencies is exhausted or the length limit for the field value is reached. The final currency symbol may terminate with a semicolon, but its inclusion is optional. When displaying the value of this field, the label "Currency" should be used.

CRYPTOCURRENCY FIELD

A UTF-8 string which should correspond to an organization or user's or preferred cryptographic currency. The 3 character cryptocurrency type is separated from the address information by a colon. A semicolon terminates the cryptocurrency string, and provides an optional separator. If a UTF-8 string follows the semicolon it should be another cryptocurrency type separated from the address by a colon. The sequence may repeat until either the signet owner's list of preferred cryptocurrencies is exhausted or the length limit for the field value is reached. The final cryptocurrency symbol may terminate with a semicolon, but its inclusion is optional. When displaying the value of this field, the label "Cryptocurrency" should be used.

If an implementation supports the use of this field value then the following cryptocurrency symbols must be recognized:

Symbol	Name	Website
BLK	Blackcoin	https://www.blackcoin.co/
BTC	Bitcoin	https://bitcoin.org/
DRK	Darkcoin	https://www.darkcoin.io/
LTC	Litecoin	https://litecoin.org/
PPC	Peercoin	http://www.peercoin.net/
STR	Stellar	https://www.stellar.org/
XRP	Ripple	https://ripple.com/currency/

In the event this field is empty, a consumer should assume the preferred cryptocurrency is Bitcoin. The value of this field should match the following syntax:

```
BTC: 19gy9ifMJuHoRbVpXBgtf6NTAT6PiDb8SQ
```

MOTTO FIELD

A UTF-8 string of numbers corresponding to a user or organization motto or vision statement. When displaying the value of this field, the label "Message-Size-Limit" should be used.

MESSAGE SIZE LIMIT FIELD

A string of numbers corresponding to the system wide size limit for incoming messages, when provided by an organizational signet, and the user specific size limit when provided with a user signet. The minimum legal value is 1 megabyte. When both signets provide legal values for this field, then the smaller of the two values takes precedence. If the value falls below this limit it must be ignored. When displaying the value of this field, the label "Message-Size-Limit" should be used.

WEBSITE FIELD

A UTF-8 string of letters or numbers corresponding to a signet owner's website. The value for this field must be a valid Hypertext Transfer Protocol (HTTP) Universal Resource Locator (URL) or HTTP Secure (HTTPS) URL. If the field does not contain a valid HTTP or HTTPS value it must be ignored. The URL should use HTTPS, although this requirement remains optional. When displaying the value of this field, the label "Website" should be used.

UNDEFINED FIELDS

TBD

IMAGE FIELD

A binary string corresponding to a user's or organization's image. This could be used to store a photograph of a user or a logo for an organization. If a user signet lacks an image, the MUA should display the image provided by the organizational signet. If both the user and organizational signets lack a valid image, then an MUA should use a default image depicting a silhouette.

Valid images must be in the Portable Network Graphics (PNG) format [PNG], invalid PNG images should be ignored, along with any image that uses a different format. Image should have a matching width and height, giving them an aspect ratio of 1. The recommended dimensions for images are: 512x512, 1024x1024 and 2048x2048. Implementations should restrict images to 1 megabyte. Consumers must be capable of handling signets with images up to 16 megabytes, but may ignore the image if it exceeds 1 megabyte. Consumers should dynamically resize and if necessary crop images to dimensions matching the area available for displaying it.

ORGANIZATIONAL SIGNET FIELDS

The following table lists the defined fields which apply only to organizational signets.

Type	Label	Status	Multiples	Type	Length
1	Primary-Organizational-Key	Required	No	Fixed	32
2	Secondary-Organizational-Key	Optional	Yes	Fixed	32
3	Encryption-Key	Required	No	Fixed	32
93 - 159	Unused 1-byte fields	--	--	--	1
200	Contact-Abuse	Recommended	No	Variable	2
201	Contact-Admin	Recommended	No	Variable	2
202	Contact-Support	Recommended	No	Variable	2
203	Web-Access-Host	Recommended	No	Variable	2
204	Web-Access-Location	Recommended	No	Variable	2
205	Web-Access-Certificate	Optional	No	Variable	2
206	Mail-Access-Host	Recommended	No	Variable	2
207	Mail-Access-Certificate	Optional	No	Variable	2
208	Onion-Access-Host	Optional	No	Variable	2
209	Onion-Access-Certificate	Optional	No	Variable	2
210	Onion-Delivery-Host	Optional	No	Variable	2
211	Onion-Delivery-Certificate	Optional	No	Variable	2
212 - 250	Unused 2-byte fields	--	--	--	2

PRIMARY ORGANIZATIONAL KEY FIELD

Must provide a valid 32 octet compressed Ed25519[vi] public key. The corresponding private key must be used to self-sign the organizational signet. The private key associated with the Primary Organizational Key (POK) is authorized for all organizational signing operations. Data signed using the POK may be validated using a DNS query, without retrieving a full organizational signet. When retrieving an organizational signet, a consumer must ensure the supplied POK matches at least 1 of the POK field values in the management record. When displaying the value of this field, the label "Primary-Organizational-Key" should be used and the key encoded using base 64.

ENCRYPTION KEY FIELD

Must provide a valid 32 octet compressed secp256k1 public key. The corresponding private key will then be needed to access the Origin and Destination chunks of a message, as described in the next chapter. When displaying the value of this field, the label "Encryption-Key" should be used and the key converted into a base 64 string.

SECONDARY ORGANIZATIONAL KEY FIELD

A binary series of octets containing with an organization's Secondary Organizational Key (SOK) and a set of flags for the authorized signing operations. The value contains a permissions octet followed by a 32 octet Ed25519 public key suitable for authenticating signatures. When displaying the value of this field, the label "Secondary-Organizational-Key" should be used, the flags displayed separated, and the key converted into a base 64 string.

The first octet for this field provides the permissions octet. This octet contains a collection of bit positions, which if enabled indicate the appropriate operation is authorized. At least one of the first 3 bit positions must be enabled. If any of the reserved flags have been enabled, the field value must be ignored and any associated signature verification operations must fail. The bits in the permissions octet authorize the secondary key to sign listed data type:

```
[   1 ] [ User Signets                    ]
[   2 ] [ Outbound Messages               ]
[   4 ] [ TLS Certificate                 ]
[   8 ] [ Software                        ]
[  16 ] [ Reserved                        ]
[  32 ] [ Reserved                        ]
[  64 ] [ Reserved                        ]
[ 128 ] [ Reserved                        ]
```

CONTACT ABUSE FIELD

A UTF-8 string of letters corresponding to the email address for the organization's abuse contact. If this field is omitted the mailbox name "abuse" is combined with the organizational domain name for the signet to derive an abuse contact. This field must provide a value, or an organization must capable of receiving complaints using the default address. When displaying the value of this field, the label "Contact-Abuse" should be used.

CONTACT ADMIN FIELD

A UTF-8 string of letters corresponding to the email address for the organization's administrative contact. When displaying the value of this field, the label "Contact-Admin" should be used.

CONTACT SUPPORT FIELD

A UTF-8 string of letters corresponding to the email address for the organization's support contact. When displaying the value of this field, the label "Contact-Support" should be used.

WEB ACCESS HOST FIELD

Should consist of a UTF-8 string of letters or numbers corresponding to the DNS name (not IP) of the web access hostname which offers Hyper Text Transfer Protocol Securely (HTTPS) and provides web based access to user email accounts. A semicolon terminates the hostname string, and provides an optional separator. The UTF-8 string which follows the semicolon should be considered a second web access hostname. The sequence may repeat until either the list of web access hostnames is exhausted or the length limit for the field value is reached. The final web access hostname may terminate with a semicolon, but its inclusion is optional. When displaying the value of this field, the label "Web-Access-Host" should be used.

WEB ACCESS LOCATION FIELD

A UTF-8 string of letters or numbers corresponding to a HTTPS resource location for the organizational webmail system. When displaying the value of this field, the label "Web-Access-Location" should be used.

WEB ACCESS CERTIFICATE FIELD

Should consist of a base 64 string which provides the encoded Ed25519 signature for the TLS certificate supplied by the web access host over HTTPS. A semicolon terminates the TLS certificate signature string, and provides an optional separator. The base 64 string which follows the semicolon should be considered a second base 64 TLS certificate signature. The sequence may repeat until either the list of valid TLS certificate signatures is exhausted or the length limit for the field value is reached. The final base 64 TLS certificate signature may terminate with a semicolon, but its inclusion is optional. When displaying the value of this field, the label "Web-Access-Certificate" should be used.

MAIL ACCESS HOST FIELD

Should consist of a UTF-8 string of letters or numbers corresponding to the DNS name (not IP) of the mail access hostname which offers connectivity using the Dark Mail Access Protocol (DMAP). A semicolon terminates the hostname string, and provides an optional separator. The UTF-8 string which follows the semicolon should be considered a second mail access hostname. The sequence may repeat until either the list of mail access hostnames is exhausted or the length

limit for the field value is reached. The final mail access hostname may terminate with a semicolon, but its inclusion is optional. When displaying the value of this field, the label "Mail-Access-Host" should be used.

MAIL ACCESS CERTIFICATE FIELD

Should consist of a base 64 string which provides the encoded Ed25519 signature for the TLS certificate supplied by the mail access host for DMAP connections. A semicolon terminates the TLS certificate signature string, and provides an optional separator. The base 64 string which follows the semicolon should be considered a second base 64 TLS certificate signature. The sequence may repeat until either the list of valid TLS certificate signatures is exhausted or the length limit for the field value is reached. The final base 64 TLS certificate signature may terminate with a semicolon, but its inclusion is optional. When displaying the value of this field, the label "Mail-Access-Certificate" should be used.

ONION ACCESS HOST FIELD

Should consist of a UTF-8 string of letters or numbers corresponding to the onion hostname for mail access. A semicolon terminates the hostname string, and provides an optional separator. The UTF-8 string which follows the semicolon should be considered a second onion access hostname. The sequence may repeat until either the list of onion access hostnames is exhausted or the length limit for the field value is reached. The final onion access hostname may terminate with a semicolon, but its inclusion is optional. When displaying the value of this field, the label "Onion-Access-Host" should be used.

ONION ACCESS CERTIFICATE FIELD

Should consist of a base 64 string which provides the encoded Ed25519 signature for the TLS certificate supplied by the onion access host. A semicolon terminates the TLS certificate signature string, and provides an optional separator. The base 64 string which follows the semicolon should be considered a second base 64 TLS certificate signature. The sequence may repeat until either the list of valid TLS certificate signatures is exhausted or the length limit for the field value is reached. The final base 64 TLS certificate signature may terminate with a semicolon, but its inclusion is optional. When displaying the value of this field, the label "Onion-Access-Certificate" should be used.

ONION DELIVERY HOST FIELD

Should consist of a UTF-8 string of letters or numbers corresponding to the onion hostname for mail delivery and signet lookups using the Dark Mail Transfer Protocol (DMTP). A semicolon terminates the hostname string, and provides an optional separator. The UTF-8 string which follows the semicolon should be considered a second onion delivery hostname. The sequence may repeat until either the list of onion access hostnames is exhausted or the length limit for the field value is reached. The final onion delivery hostname may terminate with a semicolon, but its inclusion is optional. When displaying the value of this field, the label "Onion-Delivery-Host" should be used.

ONION DELIVERY CERTIFICATE FIELD

Should consist of a base 64 string which provides the encoded Ed25519 signature for the TLS certificate supplied by the onion delivery host. A semicolon terminates the TLS certificate signature string, and provides an optional separator. The base 64 string which follows the semicolon should be considered a second base 64 TLS certificate signature. The sequence may repeat until either the list of valid TLS certificate signatures is exhausted or the length limit for the field value is reached. The final base 64 TLS certificate signature may terminate with a semicolon, but its inclusion is optional. When displaying the value of this field, the label "Onion-Delivery-Certificate" should be used.

USER SIGNET FIELDS

The following table lists the defined user signet fields.

	Label	Status	Multiples	Type	Length
1	Signing-Key	Required	No	Fixed	32
2	Encryption-Key	Required	No	Fixed	32
3	Alternate-Encryption-Key	Optional	No	Variable	1
4	Custody-Signature	Required	No	Fixed	64
5	User-Signature	Required	No	Fixed	64
6	Organizational-Signature	Required	No	Fixed	64
93	Supported-Codecs	Optional	No	Variable	1
94	Title	Optional	No	Variable	1
95	Employer	Optional	No	Variable	1
96	Gender	Optional	No	Variable	1
97	Alma-Mater	Optional	No	Variable	1
98	Supervisor	EXPERIMENTAL	No	Variable	1
99	Political-Party	EXPERIMENTAL	No	Variable	1
100	Extensions	EXPERIMENTAL	No	Variable	1
101 - 159	Unused 1-byte fields	--	--	--	1
200	Alternate-Address	Optional	No	Variable	2
201	Resume	EXPERIMENTAL	No	Variable	2
202	Endorsements	EXPERIMENTAL	Yes	Variable	2
203 - 250	Unused 2-byte fields	--	--	--	2

SIGNING KEY FIELD

Must provide a valid 32 octet compressed Ed25519 public key. The corresponding private key must be used to sign the signet signing request, generate the chain of custody signature when the signet is rotated, and sign all outbound messages. When displaying the value of this field, the label "Signing-Key" should be used and the key information encoded using base 64.

ENCRYPTION KEY FIELD

Must provide a valid 32 octet compressed secp256k1 public key. The corresponding private key will be needed to access messages encrypted to this signet, as described in the next chapter. When displaying the value of this field, the label "Encryption-Key" should be used and the key converted into a base 64 string.

ALTERNATE ENCRYPTION KEY FIELD

The first octet indicates the security level claimed by the owner. Security levels are advisory, and must not be trusted for accuracy. The second octet indicates the alternate encryption scheme, curve, and/or algorithm. Currently the only standard alternate scheme is the use of the E-521 curve in addition to the secp256k1 curve. The remaining octets are used to store actual key material. Specifically an E-521 public key in compressed, binary form, using a big endian byte ordering. [E521] When displaying the value of this field, the label "Alternate-Encryption-Key" should be used.

SECURITY LEVELS

When a user signet claims a security level for an alternate encryption key, the information must be treated with skepticism, used carefully, and considered only as an advisory. The exception is when the author and recipient belong to the same organization. Under this scenario a user should know if the organizational servers should be trusted to ensure authentic security level claims. For consumers outside an organization, there is no guarantee a user's signet is reporting the correct level, or whether an organizational domain is validating those claims. Servers should check and confirm security level claims and reject signing requests attempting to advertise an inaccurate security level.

Security levels, even in an advisory role, may provide guidance to authors, and allow them to make informed decisions about the sensitivity of any materials sent. A security level octet uses bit positions to dictate the value. Only the most significant security level should be considered relevant. The currently defined security levels are:

0 – Unprotected

Server side encryption, trustful account mode, requires absolute trust in the service provider.

1 – Sensitive

Client side encryption, cautious account mode, but thin and thick clients are supported, allowing for web access.

2 – Secret

Client side encryption, cautious account mode, thin client support is disabled for content protected by an alternate encryption key, mandating that a thick client be used to access the attachment and display sections of a message.

3 – Top Secret

Client side encryption, cautious account mode, thin client support is disabled, mandating that a thick client is always used, multiple devices are allowed.

4 – Top Secret // Special Access

Client side private key storage, paranoid account mode, mandates that a single thick client is always used.

5 – Top Secret // Special Access // Extremely Compartmented Information

Hardware security module must be used for key storage and encryption, paranoid account mode, mandates the use of a singular purpose built access device.

The remaining bits operate independently of the security level and advertise special access programs. The recommended policy is to limit the use of special access program claims to alternative encryption keys claiming a top secret clearance level. A further recommendation is to only consider multiple program clearness for the top secret special access and top secret special access extremely compartmented information clearance levels. Special access programs only carry meaning inside a confined social network involving correspondents trusted to advertise their clearance truthfully, or who rely on a common server to enforce accurate claims. The program labels are:

6 – Yankee White

7 – Shadow Hunter

8 – Underclass Applebaum

The octet values (and bit positions) associated with security levels and special access programs are:

```
[   1 ] [ Security Level // S                    ]
[   2 ] [ Security Level // SEC                  ]
[   4 ] [ Security Level // TS                   ]
[   8 ] [ Security Level // TS // SA             ]
[  16 ] [ Security Level // TS // SA // ECI      ]
[  32 ] [ Special Access // Yankee White         ]
[  64 ] [ Special Access // Shadow Hunter        ]
[ 128 ] [ Special Access // Underclass Applebaum ]
```

CUSTODY FIELD

When rotating a user signet, this field must contain a 64 octet Ed25519 signature for the first 3 user signet fields, and created with a user's previous signing key. If this is the first user signet ever created, or if the private signing key for the previous signet is unavailable, this field must be omitted. Consumers must reject a signet if the value supplied by this field is invalid. Signet resolvers must also issue a security error whenever a previous user signet is stored in the resolver's signet ring, and the chain of core signets linking the local signet with the freshly retrieved signet contains an invalid or missing custody signature. When displaying the value of this field, the label "Custody" should be used and the signature information encoded using base 64.

USER SIGNATURE FIELD

Must provide a 64 octet Ed25519 signature for the binary data stream compromising the first 4 fields in the user signet, which validates using the public Ed25519 signing key stored in field 1 (Signing-Key). Consumers must reject a signet if the value supplied by this field is invalid. When displaying the value of this field, the label "User-Signature" should be used and the signature information encoded using base 64.

ORGANIZATIONAL SIGNATURE FIELD

Must provide a 64 octet Ed25519 signature for the binary data stream compromising the first 5 fields in the user signet, which validates against the Ed25519 POK, or an authorized SOK found in the associated organizational signet. Consumers must reject a signet if the value supplied by this field is invalid. When displaying the value of this field, the label "Organizational-Signature" should be used and the signature information encoded using base 64.

SUPPORTED CODECS FIELD

Should provide a semicolon delimited list of optional media codecs supported by a user's client. The final media codec identifier may terminate with a semicolon, but its inclusion is optional. The list of media codec identifiers should not contain any repeat values, and the values supplied should use uppercase characters. Consumers must evaluate the list of codecs case insensitively. When displaying the value of this field, the label "Supported-Codecs" should be used.

ALTERNATE ADDRESS FIELD

A UTF-8 string of letters, numbers, and '@' corresponding to a user's alternate email address. A semicolon terminates an individual alternate email value, and serves as an optional separator. An additional alternate email address may be supplied following the semicolon, and the pattern may repeat until all of a user's alternate email addresses have been listed or the length limit for the field value is reached. When displaying the value of this field, the label "Alternate-Address" should be used.

TITLE FIELD

A UTF-8 string of letters and numbers corresponding to a user's job title. A semicolon terminates the 'title' string, and provides an optional separator. The UTF-8 string which follows the semicolon should be considered the 'title' label. Signet creators may omit the 'title' label. When displaying 'title', the label must be displayed, if present, otherwise an implementation may choose to omit the label, or display the default label value. For 'title' fields without a label the string "Title" should be used as the implied default value.

EMPLOYER FIELDS

A UTF-8 string of letters and numbers corresponding to a user's employer name. A semicolon terminates the 'employer' string, and provides an optional separator. The UTF-8 string which follows the semicolon should be considered the 'employer' label. Signet creators may omit the 'employer' label. When displaying 'employer', the label must be displayed, if present, otherwise an implementation may choose to omit the label, or display the default label value. For 'employer' fields without a label the string “Employer” should be used as the implied default value.

POLITICAL PARTY FIELD

A UTF-8 string of letters corresponding to a user's political party affiliation. This field is EXPERIMENTAL and may not be included in the final specification based on feedback. A semicolon terminates the 'political party' string, and provides an optional separator. The UTF-8 string which follows the semicolon should be considered the 'political party' label. Signet creators may omit the 'political party' label. When displaying 'political party', the label must be displayed, if present, otherwise an implementation may choose to omit the label, or display the default label value. For 'political party' fields without a label the string “Political Party” should be used as the implied default value.

GENDER FIELD

A UTF-8 string of letters corresponding to a user's gender. A semicolon terminates the 'gender' string, and provides an optional separator. The UTF-8 string which follows the semicolon should be considered the 'gender' label. Signet creators may omit the 'gender' label. When displaying 'gender', the label must be displayed, if present, otherwise an implementation may choose to omit the label, or display the default label value. For 'gender' fields without a label the string “Gender” should be used as the implied default value.

ALMA MATER FIELD

A UTF-8 string of letters corresponding to a user's alma mater. A semicolon terminates the 'alma mater' string, and provides an optional separator. The UTF-8 string which follows the semicolon should be considered the 'alma mater' label. Signet creators may omit the 'alma mater' label. When displaying 'alma mater', the label must be displayed, if present, otherwise an implementation may choose to omit the label, or display the default label value. For 'alma mater' fields without a label the string “Alma Mater” should be used as the implied default value.

EXTENSIONS FIELD

TBD

SUPERVISOR FIELD

A UTF-8 string of letters corresponding to a user's supervisor name. It can be used as a contact when a user is out of the office. This field is EXPERIMENTAL and may not be included in the final specification based on feedback. A semicolon terminates the 'supervisor' string, and provides an optional separator. The UTF-8 string which follows the semicolon should be considered the 'supervisor' label. Signet creators may omit the 'supervisor' label. When displaying 'supervisor', the label must be displayed, if present, otherwise an implementation may choose to omit the label, or display the default label value. For 'supervisor' fields without a label the string “Supervisor” should be used as the implied default value.

RESUME FIELD

A UTF-8 string of letters corresponding to a user's resume. This field is EXPERIMENTAL and may not be included in the final specification based on feedback. A semicolon terminates the 'resume' string, and provides an optional separator. The UTF-8 string which follows the semicolon should be considered the 'resume' label. Signet creators may omit the 'resume' label. When displaying 'resume', the label must be displayed, if present, otherwise an implementation may choose to omit the label, or display the default label value. For 'resume' fields without a label the string “Resume” should be used as the implied default value.

ENDORSEMENTS FIELD

A binary string corresponding to any endorsements a user may have. Endorsements may be used to build a level of trust or confidence that a user is of good character. The value must provide a 64 octet Ed25519 signature for the core signet at the root of a user’s chain of custody. The octets following the signature provide the email address of the signer. The address is terminated by a semicolon, and is followed by the core fingerprint derived from the core signet containing the signing key used to generate the endorsement. Any endorsement associated with a signet that is no longer part of the signer’s chain of custody must be ignored. Endorsements provided generated by users on different service providers may provide a measure of confidence that a signet is valid when it is retrieved for the first time. When displaying the value of this field, the label “Endorsements” should be used.

This field is EXPERIMENTAL and may be altered dramatically, or removed entirely based on community feedback.

SIGNET TERMINATION FIELDS

The following table lists the cryptographic fields appended to the end of a signet. These termination fields are common to both organizational and user signets.

Type	Label	Status	Multiples	Type	Length
253	Full Signet Signature	Required	No	Fixed	64
254	Signet Identifier	Required	No	Variable	2

255	Fully Identified Signet Signature	Required	No	Fixed	64

ORGANIZATIONAL SIGNATURE FIELD

Organizational and user signets must contain an Ed25519 signature in this field taken over the fields 1 through 252 in binary form. Organizational signets must provide a signature which authenticates against the POK provided in field 1 (Primary-Organizational-Key). User signets must provide an Ed25519 signature over fields 1 through 252 in binary form and which authenticates against a POK or authorized SOK supplied by the organizational signet of the associated domain. When displaying the value of this field, the label "Organizational-Signature" should be used with the signature value encoded using base 64.

SIGNET IDENTIFIER FIELD

Provides the domain name for organizational signets and the email address for user signets. The value must be supplied in lower case form.

ORGANIZATIONAL SIGNATURE FIELD

Organizational and user signets must contain an Ed25519 signature in this field taken over the fields 1 through 254 in binary form. Organizational signets must provide a signature which authenticates against the POK provided in field 1 (Primary-Organizational-Key). User signets must provide an Ed25519 signature which authenticates against a POK or authorized SOK in the organizational domain's signet. When displaying the value of this field, the label "Organizational-Signature" should be used with the signature value encoded using base 64.

SPLITTING

Signets may be split anywhere there is an organizational signature. When the signet identifier is removed (fields 254 and 255), what remains is considered a full signet. A user signet may also be split following the first organizational signature, leaving only fields 1 – 6. What remains is considered a core signet.

FINGERPRINTS

Organizational fingerprints are a SHA-512 hash of the signet fields 1 through 253. The signet data must be supplied to the hash function in binary form. The 5-octet signet header, along with fields 254 and 255 must be omitted when generating an organizational fingerprint.[vii]

User signets are used to generate two different types of fingerprints. A user's core fingerprint is a SHA-512 hash of the signet fields 1 through 6. The 5-octet signet header, along with fields 7 through 255 must be omitted when generating a user's core fingerprint.[viii] A user's full signet fingerprint is the SHA-512 hash of the signet fields 1 through 253. The 5-

octet signet header, along with fields 254 and 255 must be omitted when generating a user's full signet fingerprint. Both fingerprint types must be generated by supplying the binary signet data to the hash function.

VALIDATION

A signet is only considered valid if there is a primary lookup source, plus a secondary pre-authenticated source of confirmation.

The default method for achieving this with an organizational signet is a DMTP retrieval of the full signet, whose signature is cryptographically verified using the POK found in a management record signed using DNSSEC. Without DNSSEC, a tertiary source of confirmation is required. Currently that means the TLS certificate supplied by the DMTP server must be signed by a recognized Certificate Authority.

The default method for achieving this with a user signet, the first it is requested, is a DMTP retrieval of the full user signet, and then cryptographically verifying the signatures against the organizational signing keys. Subsequent retrievals must also provide a valid custody signature, which links the freshly retrieved signet to the previously retrieved signet.

Future plans call of the creation of a global ledger which will act as a non-reputable reflective record for user signets.

---

[v] The original plan was to use a little endian representation. It seemed natural that the binary format would match the platform being used to develop the reference implementation. However a member of the DIME development team objected passionately to this decision. After consulting others, it became clear that big endian was the "safe" choice, as binary formats used on the Internet have historically used big endian. However most of those standards were also developed before the rise of the personal computer. As such, this is a decision that would benefit from community input? Should we target the platform of yesterday (mainframes, VAX, big endian), the platform of today (desktops, X86, little endian), or the platform of tomorrow (mobile, ARM, bi-endian)?

[vi] As discussed in the management record chapter, the Ed25519 algorithm uses compressed little endian public keys. How these are encoded and whether they should be converted to big endian is still an open question.

[vii] Should consumers be able to split the core of an organizational signets off, like they can with a user signet? It would mean field number 4 would become another self-signature.

[viii] The distinction between a full and core fingerprint is both important and confusing. Perhaps we should always use the core fingerprint when referencing a signet? The downside is that if the informational fields of a signet are updated, the fingerprint wouldn't change, and consumers wouldn't receive the updated signet. Of course signets should never be modified. They should always be rotated. Thus any information field update would be associated with a key rotation, and a core fingerprint would be changed. Unfortunately this is only a best practice. No technical barriers prevent an organization from altering the informational fields and resigning them. The issue is further complicated by the fact that if users actually do a fingerprint comparison, they won't be confirming the full signet, just the core signet; an important, and yet non-obvious distinction.

PART 6: MESSAGE DATA FORMAT

This chapter describes the Dark Multipurpose Internet Mail Extensions (D/MIME) data format. The D/MIME format is an encryption scheme intended to protect Multimedia Internet Mail Extensions (MIME) [MIME] formatted messages. Like similar formats, D/MIME relies on cryptographic algorithms to ensure message confidentiality, author authenticity and non-repudiation. Unlike similar formats, D/MIME also encrypts message headers and envelope information, which makes it a fully encrypted message format. D/MIME messages are designed to minimize the leakage of metadata while being handled by transferred and ultimately delivered within a Dark Internet Mail Environment (DIME).

INTRODUCTION

The D/MIME format was created with the goal of protecting routing and delivery information, along with the historical objective of protecting message content and file attachments. With the D/MIME format, the sending and receiving service providers only have access to the minimum amount of information they need to fulfill their designated roles. The sending (origin) host will only know the domain of the recipient while a receiving (destination) host will only know the domain portion of return-path (origin), not the sender (author).

To facilitate the efficient access of D/MIME messages, the format has been structured into distinct sections, which are further subdivided into chunks. Each chunk is protected by its own unique ephemeral symmetric key. This will allow devices with resource constraints (like bandwidth, processing power, or storage space) to decrypt and validate portions of a message independently without compromising security. This allows a client to avoid downloading, decrypting, and validating an entire message before accessing its contents. The chunks have been optimized for the most commonly observed access and usage patterns. One of the primary goals for DIME was to ensure users could continue using Internet electronic mail (email) in a manner that was similar to how they have traditionally behaved. This meant being able to access encrypted messages efficiently using a variety of different platforms and devices.

Specific cryptographic primitives have been chosen based on security, context, and reputation. The D/MIME algorithms are believed to be secure for the usages described in this chapter. To ensure a common baseline, and to facilitate interoperability between DIME implementations, only one algorithm in each category is mandatory. Extensions are available which allow the use of alternative algorithms and strategies to be layered on top of the encryption schemes described below. The primitives selected are Elliptical Curve Encryption (ECC) [ECDH] for asymmetric operations (using curve secp256k1) [SEC], the Advanced Encryption Standard (AES) [AES] for symmetric encryption, the Secure Hash Algorithm (SHA) for hash operations, and the Edwards-curve Digital Signature Algorithm (EdDSA) for cryptographic signing operations (using curve Ed25519). Users may also advertise alternative public encryption keys using the curve known as E-521 [E521].

HISTORICAL CONTEXT

The D/MIME message format draws its inspiration from the OpenPGP [PGP] and Secure Multipurpose Internet Mail Extensions (S/MIME) [SMIME] formats. Without the research and development efforts invested in the development of

those standards, DIME would not be possible. The changes described in this document draw upon the experiences and the lessons learned by community while implementing, deploying, and communicating with messages protected by OpenPGP and S/MIME. Readers already familiar with those standards will find this specification easier and more accessible.

The primary difference between D/MIME and OpenPGP or S/MIME is that it is a fully encrypted message format. D/MIME protects the envelope and headers of a message, in addition to its contents. Historically, the return path and recipient address associated with a message have been called the envelope. In the past, the message envelope was transferred at the protocol level, exposing it to collection by compromised handling agents. D/MIME encrypts envelope information within the message object, and relies on DIME capable systems to extract and process it. Encryption is used to ensure an agent only has the information necessary to relay a message to its next hop. This minimizes the amount of information exposed to the minimum amount necessary for a mail system to function.

D/MIME employs a simple tree like binary structure, with each leaf encrypted separately. This allows a system to access portions of a message without compromising the remainder. It also allows resource constrained clients to validate cryptographic signatures, and access pieces without having to download a message in its entirety. Also noteworthy is that the most commonly needed message headers have been separately encrypted, allowing them to be downloaded separately and displayed during list operations more efficiently.

Domain Keys Identified Mail (DKIM) [DKIM] is technological parent of another aspect of DIME. To improve security, and restrict its abuse, DIME systems require that D/MIME messages be signed by the author and then signed again by the organizational domain. Authors are required to generate a tree signature in addition to a full signature. Cleartext MIME content is also signed by the author. The organizational domain must also sign the full contents of a message, and may generate a bounce signature which allows it to verify the origin of a partial bounce.

The final aspect of D/MIME messages which is distinct from OpenPGP and S/MIME is that each message must be encrypted separately for each recipient. This ensures handling agents can't determine how many recipients a message is being sent to, and if the cleartext contents are encrypted using distinct symmetric keys, it will ensure each copy of a message is uniquely distinct.[9]

LEAKAGE

Does not mask metadata for two people on the same mail system

The structure of a message is still accessible, and must remain so for efficient access by resource constrained MUAs, which would allow attackers to fingerprint and then track messages if they could compromise the handling agents, or compromise the TLS connections used during transfer operations

TRANSFER ENCODING

D/MIME is a binary format, and any alteration of the encrypted data would cause the signature validation algorithm to fail. To ensure messages are handled properly, and without any alteration, messages are encoded using the Privacy

Enhanced Mail (PEM) [PEM] mail format. This allows D/MIME messages to be processed, handled and viewed by humans, and processed by the customary mail tools and techniques without corruption. Because the PEM format increases the size of messages, a system specifically designed to handle D/MIME messages may process and store messages in their binary form. If an implementation does process and store binary D/MIME messages, it must ensure any system, or component it hands a message to is similarly capable of handling the binary format without corrupting the data. Unless it obtains such assurances it must first encode a message into the PEM format before transferring it.

The PEM format relies on encapsulation boundaries to delimit individual messages and communicate the type of data being carried. D/MIME messages must use the "ENCRYPTED MESSAGE" boundary, with the binary D/MIME data armored using base 64 and stored within the boundaries. In contrast to convention, D/MIME messages should not include the traditional base 64 "=" padding characters. Instead the padding octets should be calculated using the formula:

length modulo 4 = *pad*

The result will determine the number of padding octets required. A D/MIME message armored using the PEM format would use the syntax:

```
-----BEGIN ENCRYPTED MESSAGE-----
message
-----END ENCRYPTED MESSAGE-----
```

ENDIANNESS

The D/MIME format is a binary schema, which relies on numeric values to convey information and facilitate parsing. The binary values defined by this specification will always use network byte order, which is defined as a big endian representation, requiring the most significant byte to be stored in the smallest address, and the least significant byte be stored in the largest address. Implementations running on little endian systems will need to convert the values to ensure proper processing.

DATA STRUCTURES

D/MIME messages are comprised of a message header, and an arbitrary number of individual chunks. Chunks are comprised of a chunk header, a payload and, for encrypted chunks, the appropriate number of keyslots. Every encrypted payload is protected using a distinct, randomly generated key. The randomly generated keys are stored inside the keyslots. Keyslots are protected using a distinct shared secret which is unique for each message, and distinct for each actor authorized to access a message. The number of keyslots is determined by which actors must have access to the preceding payload.

TRACING

TBD

ALGORITHMS

The D/MIME message format relies on 3 cryptographic algorithms for key agreement, encryption and signatures. The Elliptical Curve Diffie-Hellman (ECDH) [ECDH] key agreement protocol is used to calculate a shared secret. Encrypted payloads and keyslots are encrypted using the Advanced Encryption Standard (AES) [AES]. Both encrypted and cleartext data is verified using the Edwards-curve Digital Signature Algorithm (EdDSA) [EDDSA].

The AES key used to protect individual key slots, or the Key Encryption Key (KEK), and is calculated using ECDH and the secp256k1 elliptical curve. Each KEK is generated using an ephemeral message key, and the public encryption key stored in the signet of each actor associated with a message (author, origin, destination and recipient). Keyslots are protected using a 256-bit KEK, and encrypted using AES and the cipher-block chaining (CBC) mode of operation. Keyslots hold randomly generated 256-bit AES keys along with the randomly generated Initialization Vector (IV) needed to access encrypted payloads. The encrypted message data and the cleartext data for every encrypted chunk payload are signed using the EdDSA algorithm. Signatures are generated using the Twisted Edwards curve: $x^2 + y^2 = 1 (121665/121666)x^2y^2$ (collectively called Ed25519) which is birationally equivalent to Curve25519.

MESSAGE HEADER

D/MIME messages begin with a 6 octet header. Like all of the binary formats used throughout DIME, a D/MIME message begins with 2 octets which provide the magic number. The following 4 octets contain the size of a message in its binary form. The size value does not include the 6 octet header, but does include all of the data that follows it. Because the size is 4 octets, the binary portion of a message has a technical limitation of 4,294,967,296 octets.

A D/MIME message will always begin with the two octet numeric identifier 1847. Future versions of this specification which are syntactically compatible will continue to employ this same magic number. If a parser conforming to this specification encounters any other value besides 1847, it must reject the message and notify the user.

```
[  2 octet  ] [ Magic Number (1847)  ]
[  4 octets ] [ Message Size         ]
[  variable ] [ Message              ]
```

CHUNKS

Messages are broken up into a series of "chunks." Chunks are broken up into three distinct sections: the header, the payload and the keyslots. A chunk header is 4 octets in length, with the first octet used to store the type code for a chunk, and remaining 3 octets used to store the payload length. Because the length value is 3 octets, and AES requires that a payload be divided into 16 octet blocks, the maximum size for a payload is 16,777,200 octets. Following the length is the actual payload data, which is then followed by a variable number of 64 octet keyslots.

Envelope, metadata and signature chunks must appear using an increasing numerical order. Content chunks must appear after the metadata chunks and before any signature chunks. Only content chunk types may be used more than once. Message content is subdivided into display and attachment sections. Display chunks may appear in any order inside their

section, but must appear before attachment chunks. Attachment chunks may also appear in any order provided they follow the display chunks and appear before the signature chunks. See the structure section below a description of how messages are divided into chunks.

```
[  1 octet  ] [ Type                ]
[  3 octets ] [ Payload Length      ]
[  variable ] [ Payload             ]
[  variable ] [ Keyslots            ]
```

SPECIALIZED PAYLOADS

Specialized payloads are structured differently from encrypted payloads. The tracing, ephemeral and signature chunks use distinct payload formats. The tracing and ephemeral chunks are the only chunks that are stored as cleartext. As such they do not have any trailing keyslots. The ephemeral chunk contains the ephemeral public key for the message. The ephemeral public key is combined with the authorized private key using ECDH and the result is the KEK needed to decrypt keyslots and access the encrypted chunks. The ephemeral chunk payload is an unencrypted, but compressed secp256k1 public key in binary form.

Signature chunks also use a specialized payload format. While the payload for signature chunks is encrypted, the decrypted data does not conform to the standard structured layout used by other encrypted chunks. For signature chunks, the decrypted payload is the 64 octet Ed25519 signature value stored in binary form.

Ephemeral/Tracing Payloads

```
[  variable ] [ Unencrypted Data    ]
```

Signature Payloads

```
[ 64 octets ] [ Ed25519 Signature   ]
```

ENCRYPTED PAYLOADS

Encrypted payloads compromise most of the chunk types. They all use the same basic structure: signature, length, flags, pad, data segment and padding:

```
[ 64 octets ] [ Ed25519 Signature   ]
[  3 octets ] [ Data Segement Length ]
[  1 octet  ] [ Flags               ]
[  1 octet  ] [ Padding Length      ]
[ variable  ] [ Data Segement       ]
[ variable  ] [ Padding             ]
```

The entire payload must be encrypted with AES in CBC mode, using a randomly generated 256 bit key and a randomly generated 16 octet IV. Because AES in CBC uses a 16 octet block size, the overall payload length must be padded to a 16 octet boundary. There are a fixed 69 octets. Because the overall length of a payload is limited by the length field in the chunk header, the maximum size for a single data segment is 16,777,131 octets. Additional padding is optional, but

should be added to data segments smaller than 187 octets, making the recommended minimum payload 256 octets in length. An additional random amount of padding should also be added to mask the structural fingerprint for a message. Data segments larger than 16,777,131 octets must be split across chunks and reassembled by the parser.

The Ed25519 signature is used to validate the decrypted chunk data, and is taken over all of the octets that follow the signature inside a payload: the length, flags, padding length, data segment, and padding. The signature must be validated by the parser to ensure the data payload has not been modified. If a data segment is split across multiple chunks, each chunk will contain its own signature over just the data segment portion contained in that chunk.

The 3 octet data segment length is based on the amount of user generated data carried by a chunk. The value must never be 0; if a parser encounters a data segment length of 0, the entire message must be rejected. In theory any data segment could be arbitrarily padded and split across multiple chunks to disguise the nature, structure and amount of data carried by a message. However, a parser must never split a data segment across more than 4 chunks unless it is larger than the maximum usable size for 4 consecutive chunks, or 67,108,524 octets.

The 1 octet flags is a series of bitwise operators used to enable different aspects of D/MIME. Currently 4 of the bits have been assigned, with the remaining reserved for future use. If a parser encounters a bit wise operator it does not recognize it must reject the chunk. Current the following bit positions have been assigned:

```
[   1 ] [ Alternate Padding Algorithm Enabled ]
[   2 ] [ Alternate User Key Applied to Data  ]
[   4 ] [ GZIP Compression Enabled            ]
[   8 ] [ Reserved                           ]
[  16 ] [ Reserved                           ]
[  32 ] [ Reserved                           ]
[  64 ] [ Reserved                           ]
[ 128 ] [ Data Segment Continuation Enabled  ]
```

By default the padding is determined by the single octet that follows the flags field. Any octets appended to the data segment must be set to the value of the padding octet. Parser implementations must reject chunks where the value of the padding octets does not match the value of the padding length octet. When the first bit in the flags octet is set to 0, the data segment length plus the padding length must align to a 16 octet boundary. In addition to the octets needed for alignment, up to 240 additional octets (in 16 octet blocks) may be added as padding. If padding is appended to the data segment, beyond what is needed for alignment, the amount of additional padding must be randomized. Including a random amount of padding is optional, but would ensure two identical messages will have different structural fingerprints, and further assist in disguising the length of the message contents. The algebraic definition is:

```
Header (69) + Data Length (Var) + Padding (Var) = Chunk Length % 16 == 0
```

If the alternative padding algorithm is enabled, the padding octet must be interpreted as the number of additional 16 octet blocks appended to a data segment, allowing up to 4,080 octets of stuffing, beyond the padding octets needed strictly for alignment. When the alternative padding algorithm is enabled, the amount of padding included for alignment must be calculated automatically. This will append between 0 and 15 padding octets to the data segment. Like the default padding algorithm described above, all padding octets must be set to the value of the padding length octet. The algebraic definition for the alternative padding algorithm is:

```
Padding Length * 16 = Stuffing
(Header (69) + Data Length (Var)) % 16 = Padding

Header (69) + Data Length (Var) + Padding (Var) + Stuffing (Var)
       = Chunk Length % 16 == 0
```

If the alternate encryption bit is enabled, then the cleartext data segment represents the data that must also be decrypted using the alternative user key. If the GZIP [GZIP] compression bit is enabled, then the cleartext data segment has been compressed using GZIP. Parser implementations must implement GZIP and be capable of accessing compressed data segments. D/MIME message creation implementations should pick one of three suggested compression strategies:

- Compression is Always Enabled
- Compression is Enabled if, and only if, it Reduces the Data Segment Length (Recommended)
- Compression is Never Enabled

If the spanning bit has been enabled, then the data segment continues into the next chunk. The chunk containing the final piece of a data segment must have the spanning flag disabled. Continuation chunks must use an identical type code as the chunk they are continuing and appear immediately after the chunk with the spanning flag enabled.

While parsing an encrypted payload, a parser should treat any violation of this specification identically. Specifically, data length overflows, an invalid padding lengths, a non-aligned payload, a cleartext signature failure, a padding octet whose value does not match the padding length octet, a reserved flag bit with a non-zero value, a compression flag bit that is enabled despite the data segment having uncompressed or corrupted data, or when chunks are split across more than 4 payloads unnecessarily; all of must be treated as data corruption. For spanning chunks, if any of the payloads is considered corrupt, all of the associated chunks must also be considered corrupt and discarded. The decision whether to reject a message when a single chunk is corrupt has been left undefined intentionally.

KEYSLOTS

Every keyslot must be 64 octets in length. Keyslots are encrypted using the KEK for each actor. The number of keyslots is determined by the chunk type. Every encrypted chunk must have a keyslot for the author and recipient. Envelope chunks and signature chunks have additional keyslots for the origin and destination domains. See the individual chunk descriptions below for additional details on who can access the different types of chunks.

Every encrypted chunk must be protected using a randomly generated 32 octet, or 256-bit AES key, and a randomly generated 16 octet IV specific to that chunk. Keyslots use the first 32 octets to store a copy of the IV, and the final 32 octets to store the AES key.

To first 32 octets of data for a keyslot must be unique for every actor to prevent a variety of known, and future differential cryptanalysis attacks. To accomplish this, the 16 octet value used as the IV for a chunk is combined with another randomly generated 16 octet value using an exclusive or operation (XOR). The random 16 octet value used in the XOR operation must be unique for each keyslot. A keyslot stores the randomly generated 16 octet value first, and is followed by the 16 octet result of the XOR operation. When accessing an encrypted chunk, these two values must be

combined again using another XOR operation to recover the IV. The final 32 octets of a keyslot store the AES key for the chunk.

MESSAGE STRUCTURE

The following diagram is designed to illustrate how a typical Internet electronic mail message (email) [IMF] message is split into D/MIME chunks (note the different user and organizational signature chunks have been combined for brevity):

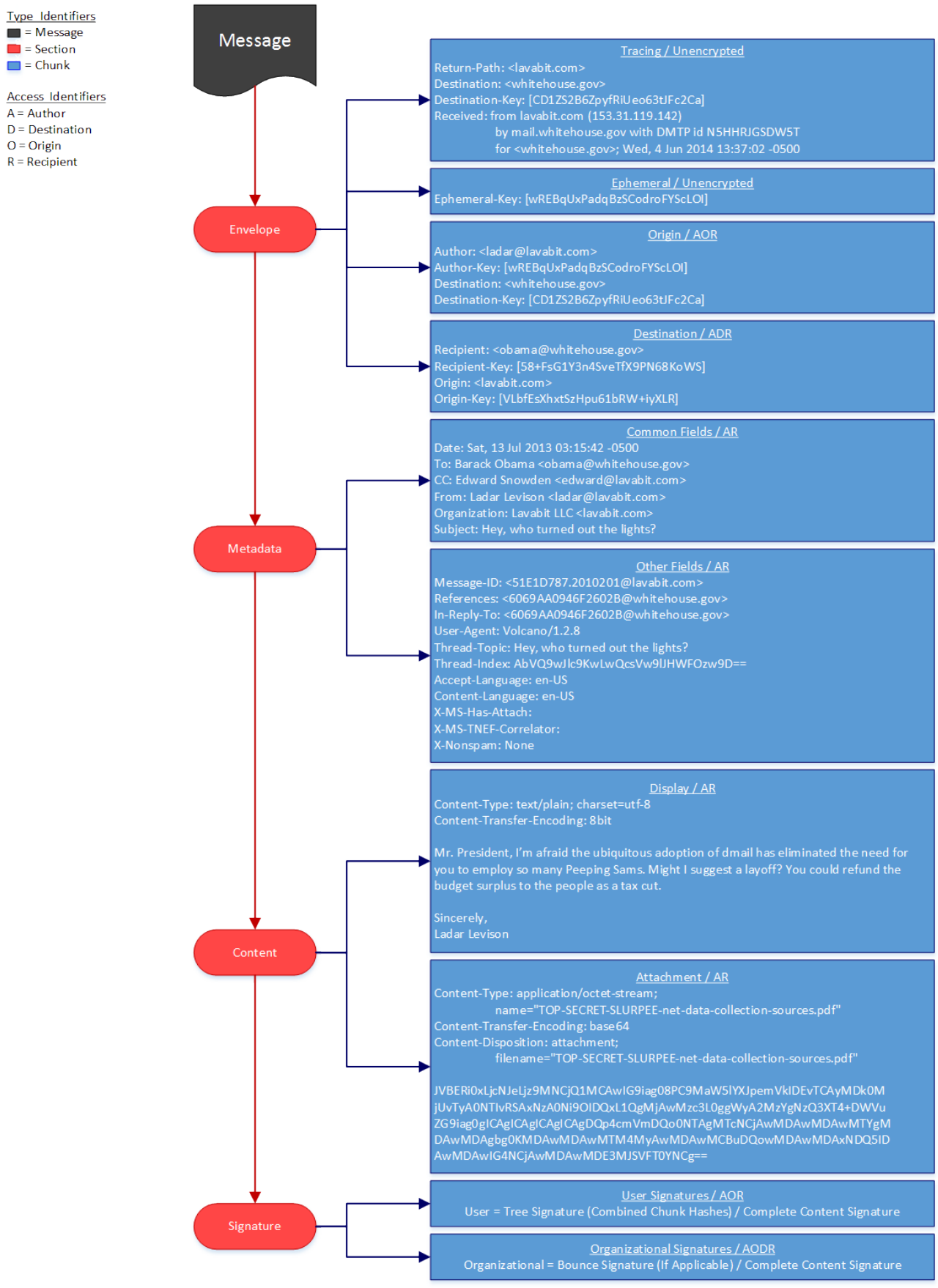

Figure 7 – Message Structure

TYPES

The currently defined section groupings and chunk types are listed below. Please note that sections have been highlighted in blue.[10]

Number	Name	Access	Required	Duplicates	Sequential
0	Tracing	Unencrypted	N	N	Y
1	Envelope	N/A	Y	N	Y
2	Ephemeral	Unencrypted	Y	N	Y
3	Alternate	AR	N	N	Y
4	Origin	AOR	Y	N	Y
5	Destination	ADR	Y	N	Y
32	Metadata	N/A	Y	N	Y
33	Common	AR	Y	N	Y
34	Headers	AR	N	N	Y
64	Display	N/A	Y	N	Y
65	Display-Multipart	AR	N	Y	N
66	Display-Multipart-Alternative	AR	N	Y	N
67	Display-Content	AR	N	Y	N
128	Attachments	N/A	N	N	Y
129	Attachments-Multipart	AR	N	Y	N
130	Attachments-Multipart-Alternative	AR	N	Y	N
131	Attachments-Content	AR	N	Y	N
224	Signatures	N/A	Y	N	Y
225	Author-Tree-Signature	AOR	Y	N	Y
226	Author-Signature	AOR	Y	N	Y
248	Organizational-Metadata-Bounce-Signature	AODR	N	N	Y
249	Organizational-Display-Bounce-Signature	AODR	N	N	Y
255	Organizational-Signature	AODR	Y	N	Y

ENVELOPE

TBD

TRACING

TBD

EPHEMERAL

TBD

ALTERNATE

TBD

ORIGIN

TBD

DESTINATION

TBD

METADATA

TBD

COMMON

TBD

HEADERS

TBD

DISPLAY

TBD

DISPLAY-MULTIPART

TBD

DISPLAY-MULTIPART-ALTERNATIVE

TBD

DISPLAY-CONTENT

TBD

ATTACHMENTS

TBD

ATTACHMENTS-MULTIPART

TBD

ATTACHMENTS-MULTIPART-ALTERNATIVE

TBD

ATTACHMENTS-CONTENT

TBD

SIGNATURES

TBD

AUTHOR-TREE-SIGNATURE

TBD

AUTHOR-SIGNATURE

TBD

ORGANIZATIONAL-METADATA-BOUNCE-SIGNATURE

TBD

ORGANIZATIONAL-DISPLAY-BOUNCE-SIGNATURE

TBD

ORGANIZATIONAL-SIGNATURE

TBD

[9] This is an aspect of D/MIME that would benefit from community feedback. The current plan is to allow a message which uses the same symmetric keys to be submitted once using DMAP, plus the individual key slot and signature values for each recipient. The submission server would assemble the pieces, and then the full contents would be transferred separately between servers over DMTP. Users who want to avoid fingerprinting of the contents would need to submit a separate copy for each recipient.

[10] Should we define different display types for the different MIME content types? And possibly even differentiate a few of the subtypes, like text/plain and text/html, so a client can distinguish which display chunk it should retrieve for display purposes? This would leak information about what information a message is carrying, and make them easier to fingerprint, but could allow a client to avoid downloading a video message if it didn't support video (for example on a mobile device). Even if we did add this, there would be a generic catchall chunk type implementations could use if they didn't like the leakage.

PART 7: DARK MAIL TRANSFER PROTOCOL (DMTP)

DMTP has been engineered to provide the functionality necessary for a mail user agent to fully encrypt messages sent between two DIME addresses automatically. DMTP is the primary method used by a DIME-enabled mail transfer agent to securely and reliably deliver a fully encrypted message to its final destination. DMTP takes advantage of the Dark/Multipurpose Internet Mail Extension (D/MIME) format, a fully encrypted message schema, to ensure a message can be properly routed while minimizing the leakage of metadata to handling agents. The D/MIME format also ensures the message contents are protected from eavesdropping and manipulation.

For the encryption process to function automatically, a mail user agent must be able to locate and retrieve the public encryption keys, which are contained inside a signet, for a given recipient. The task of retrieving and authenticating signets is performed by a Signet Resolver (SR). Signet resolvers use DMTP to retrieve organization and user signets, to determine whether cached signets are stale and to retrieve the historical signets required to validate the chain of custody for an account when it discovers a new user signet.

Unlike traditional mail transfer protocols, DMTP relies on the encrypted message envelope embedded within a D/MIME message to determine where a message should delivered. This ensures a mail transfer agent only has access to the information required to accomplish the next step in the delivery process. It is the responsibility of a mail transfer agent to deliver a message to its destination, or report its failure to do so.

DMTP is a network protocol that is independent of a specific transport. However, for the purpose of this document, it is assumed that the DMTP session is between two Internet connected hosts, a client that initiates the connection and a server that accepts input and responds accordingly. That the two hosts are able to exchange data packets using the Internet Protocol (IP) [IP], in combination with the Transmission Control Protocol (TCP) [TCP] to establish a reliable data stream which is used to establish a secure communications channel using the Transport Security Layer (TLS) [TLS] protocol. Thus, TCP is responsible for the connection layer, IP is responsible for the internetworking layer and TLS is responsible for protection against network threats. The fallback strategy is to relay data packets printed in hexadecimal on cellulose pulp that has been dried into flexible sheets and relayed using avian carriers. [AVIAN]

PROTOCOL MODEL

DMTP is intentionally simplistic. Experience has shown that excessively complex protocols are difficult to implement correctly, with ambiguity often creating incompatibility problems. Complex protocols are synonymous with overly complex implementations. The layers of abstraction needed to implement a complex protocol often serve to mask defects or hide subtle security vulnerabilities. To avoid this, DMTP borrows heavily from the Simple Mail Transfer Protocol (SMTP) [SMTP].

DMTP has been intentionally limited to unauthenticated functionality. The protocol relies on the use of unauthenticated exchanges to ensure input data is always considered hostile and evaluated carefully before processing. DMTP hosts may

advertise their support for protocol extensions that enhance the required DMTP functionality specified below, provided the extensions do not require authentication.

DMTP uses a rigid syntax for commands and replies. The protocol relies on a line-based structure, where each line is considered a semantic unit that can be evaluated independently to determine whether it is time to proceed. The result is a dialog that is purposefully lock-step, with every request resulting in a reply. Clients must ensure they always wait until a reply is received before making subsequent requests unless a server advertises support for the command pipelining protocol extension.

Every request is made using a command, which begins with a verb. Some commands require that arguments be supplied after a verb, while others allow for optional arguments. A few will never accept arguments. In every case where an argument is supplied, it is separated from the verb or a previous argument by a space character.

Every command results in a reply; with the reply indicating whether a command was accepted, whether message data or additional commands should be sent, or that an error occurred. All replies begin with a three-digit numeric code use and use syntax specified below which allows for single line and multiline response depending on the outcome and the information a server needs to supply in the response.

HISTORICAL CONTEXT

DMTP is intended as a replacement for SMTP [SMTP], with modifications focused on improving the privacy and security of Internet electronic mail (email). As a result, it borrows heavily from the syntactic structure and transaction model used by SMTP. Readers familiar with SMTP should feel comfortable with DMTP. The relationship between the protocols is by design, by making SMTP and DMTP semantically similar, it should be easier for someone familiar with the former to implement and deploy support for the latter.

DMTP does possess three primary differences. First, mailbox names have been removed from the protocol conversation. The envelope addresses, author and recipient, which are used for routing a message, have been removed from the protocol conversation, and must be extracted from the encrypted message. Second, commands have been added, using new verbs, or repurposed SMTP verbs, for transferring user and organizational signets. Finally, TLS support is no longer optional but a protocol requirement. While DMTP does not rely on TLS for security, it does provide an additional layer of protection, and a measure of defense, against threats posed by hostile networks. TLS provides perfect forward secrecy protection from attacks which involve capturing network traffic, and makes traffic analysis more difficult.

This specification does not provide guidance, nor address any of the requirements involved with sending and receiving unencrypted (or “naked”) messages over SMTP. However, it is important to note that such messages sent via SMTP are vulnerable, and any organization that does not want their private messages read by unauthorized third parties should deprecate the use of SMTP and migrate their mail to DIME.

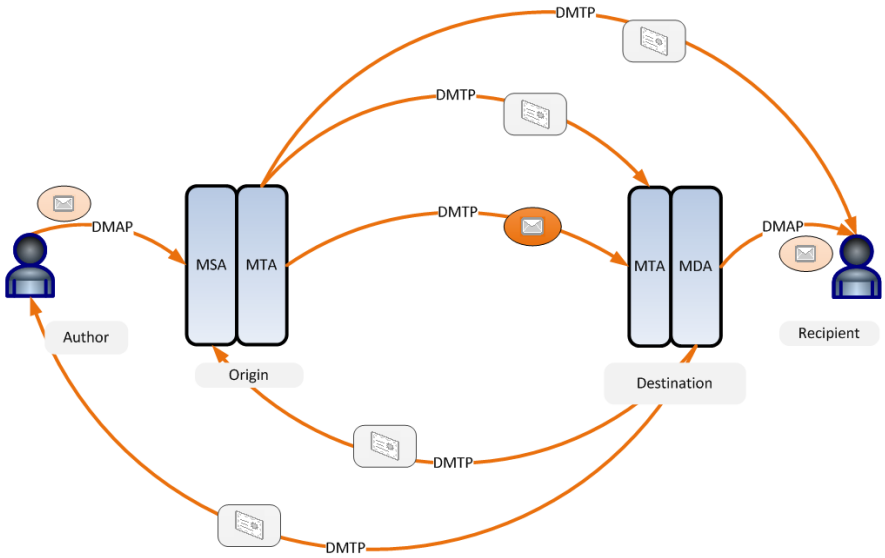

Figure 8 - DIME Architecture

LINE BASED PROTOCOL

DMTP lines consist of American Standard Code for Information Interchange (ASCII) [ASCII] characters. ASCII characters consist of a single octet with the high order bit cleared. For DMTP, this means all protocol messages should consist of data between the hex values 0x01 and 0x7F.

Protocol commands and responses are exchanged using lines, which complete semantic units. Conforming implementations must wait until a line terminator is received before evaluating the content of a line and proceeding unless a protocol extension has been employed, such as command pipelining. All implementations must be capable of handling lines which are 512 octets in length.

A string of ASCII octets is always followed by a line terminator, which serves the end of the semantic unit. For DMTP the line terminator must be the character "<LF>" (hex value 0A). Conforming implementations must not generate any other character sequence for use as a line terminator. Server implementations may choose to recognize the historical line termination character sequence "CR" (hex value 0D) followed immediately by "LF" (hex value 0A). This optional functionality would allow for the use various client tools to continue functioning, which are only capable of producing the <CRLF> sequence.

In addition, the appearance of "CR" or "LF" characters outside of their use as line terminators has a long history of creating problems. To avoid this issue in the future, DMTP client implementations must not send these characters unless they are being used as a line terminator, or a protocol extension has been agreed upon.

COMMANDS AND REPLIES

Command semantics, upper case verbs

Space separated arguments, with email addresses and domain names using always being enclosed by <> and encoded binary data argument values being enclosed by []... and equal signs used to separate argument names from the value.

Responses, success versus error, theory and severity

Single vs multiline replies

MAIL TRANSACTIONS

Message transactions.

OBJECTS

Signets and Messages

Modifications - tracing

DELIVERY

Addressing

Validation steps

Mail stores

Bounces

CACHING

How to handle the caching of signet lookups.

CONNECTIONS

A consumer begins the process of initiating a DMTP connection by retrieving the management record for a target domain name. If no management record is discovered, then a consumer should rely upon its local cache until any previous entries have reached their expiration. If no management record is found, or the cached record has expired, a consumer must conclude the target domain is not DIME enabled and does not support DMTP. If a Mail Transfer Agent (MTA) is attempting to establish DMTP connection for the purpose of message delivery, it should consider the error temporary and apply the retry logic supplied below.

If a management record is found, and it contains a valid delivery field (dx) value, consumers should first attempt to resolve and connect to the provided values using port 26 in single protocol mode. If multiple delivery field values are encountered, a conforming implementation must attempt at least three unique host names before continuing. It is

recommended that consumers attempt the supplied host names in a random order, independent of what order they appear in a management record.

If the management record does not contain a valid delivery field, or the consumer is unable to establish DMTP connection in single protocol mode, it must fall back into dual protocol mode. To find the dual protocol hosts for a domain, consumers must query the target domain name for a mail exchange (mx) resource record. If a valid mail exchange record is found, a conforming client must attempt the connection using port 25, and if that fails, may attempt the connection using port 587. If a TCP connection is established, and then the consumer a consumer should apply rules specified below for DMTP hosts operating in dual protocol mode. A conforming client must attempt at least three unique mail exchange resource record host names before continuing.

If a consumer is unable to establish a connection using the logic above, it may consider the attempt a failure, or optionally attempt to the establishment of a DMTP connection using the target domain name. If a consumer attempts to establish the DMTP connection using the target domain name it should attempt a connection on port 26 first, and apply the rules associated with single protocol mode. If the connection fails using port 26, then a consumer should attempt a dual protocol connection using port 25, and if that fails, may choose to also attempt a dual protocol mode connection using port 587.

Valid delivery field values and mail exchange resource records must always be fully qualified host names that resolve to A or AAAA resource records. The use of IP addresses, or CNAMES, is prohibited and conforming implementations should ignore such values.

If an MTA is unable to locate a valid management record, or establish a DMTP connection using any of the supplied host names, it should consider the error temporary. If the policy field in the management record indicates a domain is operating in experimental mode, then a mail transfer agent may continue using SMTP [SMTP]. Otherwise a conforming MTA must queue and periodically retry the delivery attempt for at least 72 hours. The algorithm used to schedule retries is intentionally undefined, but a conforming implementation must ensure it will retry delivery at least once every 12 hours. An MTA should use a gradually increasing delay between delivery attempts, provided the interval never exceeds 12 hours. If a consumer is unable to establish a DMTP connection during the required 72 hour period, it must consider the error permanent and report its failure to deliver the message back to the author.

An MTA may choose to report temporary failures after 4 hours, but must continue making delivery attempts for a during the entire 72 hour period unless a user intervenes.

All DMTP connections must be secured using TLS v1.2 [TLS] and the cipher suite ECDHE-RSA-AES256-GCM-SHA384[11] [TLS-ECDHE]. The required cipher suite is uniquely identified during a TLS handshake by the octet values 0xC0, 0x30.[12]

CERTIFICATES

DMTP connections must always be secured using TLS [TLS]. This will require that servers be configured to supply an X.509 certificate during the connection. The certificate provides a signed RSA public key, along with a number of other attributes. Certificates supplied by DMTP hosts must use RSA keys that are least 2048 bits, and keys of at least 4096 bits

are strongly recommended. DMTP client implementations must support RSA keys up to 8192 bits in length, and should support RSA keys of 16384 bits in length. If a conforming DMTP consumer encounters a host using an RSA key that is shorter than 2048 bits, it should complete the TLS handshake and immediately shutdown the connection using the QUIT command specified below.

DMTP hosts must allow consumers to specify the intended host name for the connection using the Server Name Identifier (SNI) extension in single protocol mode, and as an argument to the STARTTLS command when operating in dual protocol mode. If a DMTP host is configured with a TLS certificate containing a Common Name (CN) or Alt Name (AN) attribute matching the supplied host name, it must supply the matching certificate. If the DMTP host does not have a matching TLS certificate, it must allow the connection to proceed using a default TLS certificate. Every DMTP host must be configured with a default TLS certificate.

If the management record provided TLS field values, then consumers must validate TLS certificates against the supplied values. TLS field values are Ed25519 signatures [EDDSA], and generated using the target domain's Primary Organizational Key (POK). If a TLS field value is found a certificate must be confirmed against one of the supplied field values. Note that it is possible for a management record to supply more than one TLS field value, in which case all of the supplied values must be compared until a matching entry is found. If none of the supplied signatures can be validated then a consumer must terminate the DMTP connection and notify the user of an error with possible security implications. TLS certificates must be converted to a concrete data stream using the Distinguished Encoding Rules (DER).

If a management record is validated by a DNSSEC signature, and the certificate was validated against a TLS field value, then a consumer must accept certificates that would normally be rejected using strict validation. Consumers must accept: self-signed certificates, expired certificates and certificates without a matching Common Name (CN) or Alt Name (AN) attribute.

If a certificate is confirmed using the TLS field value, then a consumer should not perform the Online Certificate Status Protocol (OCSP) check [OCSP]. OCSP checks are discouraged if the certificate can be validated using the TLS field even if the management record is not signed using DNSSEC because the request could inadvertently leak information about which domains a host is contacting. All other X.509 validation rules should be applied according to the TLS v1.2 specification regardless of whether the certificate is validated using the management record.

DNSSEC Validation	TLS Field Validation	X.509 Validation	OCSP Check	Result
Yes	Matches	N/A	N/A	Pass
Yes	None	Passed	Yes	Pass
Yes	None	Failed	N/A	Fail
Yes	Mismatch	N/A	N/A	Fail
No	Matches	Passed	Skip	Pass
No	Matches	Failed	N/A	Fail
No	None	Passed	Yes	Pass
No	None	Failed	N/A	Fail

SINGLE PROTOCOL MODE

If a consumer is using the host name supplied by the delivery field in the management record, it must connect to the provided host using port 26. The connection should be initiated using TLS [TLS]. Consumer should supply the host name provided by the delivery field, or "dx" value, in the management record using SNI TLS extension [TLS-SNI]. Connections across port 26 must be specifically for DMTP and upon successfully connecting, consumers must see a banner that starts with the sequence 220 and contains the string "DMTP". Greetings should match the pattern:

```
220 <domain.tld> DMTP {freeform}
```

If a consumer does not encounter the appropriate DMTP protocol banner once the TLS connection has been established, then it must immediately shutdown the connection and treat the host name the same way it would if the TLS connection had never succeeded. Consumers conforming to this specification must make TLS connection attempts to at least three valid, and unique delivery field host names before continuing on. If fewer than three unique and valid delivery field host names values are found, then a consumer should try all of the unique and valid host names it encounters.

DUAL PROTOCOL HOSTS

When attempting a DMTP connection using a hostname supplied by a mail exchange (mx) resource record, a consumer should assume the host is operating in dual protocol mode and attempt an unencrypted TCP [TCP] connection using port 25. If port 25 fails, a consumer may attempt the TCP connection using port 587. If either port results in a TCP connection, the consumer should confirm whether a host supports DMTP before proceeding by parsing the banner greeting. A DMTP capable host operating in dual protocol mode must greet consumers with a banner that starts with 220, and contains the string "DMTP". Greetings should match the pattern:

```
220 <domain.tld> ESMTP DMTP {freeform}
```

If the appropriate banner is encountered, a consumer must immediately elevate the TCP connection into DMTP mode using the STARTTLS command syntax specified below. If the consumer does not encounter the appropriate dual protocol banner, it must fail immediately and continue as if the TCP connection never succeeded. To elevate a successful TCP connection into DMTP mode, consumers must initiate a TLS handshake using the STARTTLS command syntax:

```
STARTTLS <domain.tld> MODE=DMTP
```

If a dual protocol host encounters a MODE parameter for a consumer attempting to elevate a connection into DMTP mode, but is unable to negotiate a TLS connection using the cipher suite specified above, then the host must ensure a STARTTLS command fails. If a consumer does find that it connected to a host that allows elevation into DMTP, without using the required cipher suite, it must immediately issue a QUIT command and shutdown the connection. Consumers that encounter this scenario should alert the user to a possible security threat before proceeding.

A connection which has been successfully elevated into DMTP mode encounter a reply with the status code 250, and contain the string "DMTP". The response should match the following pattern:

```
250 OK DMTPv1 {freeform}
```

Dual protocol implementations may choose to allow consumers to issue the STARTTLS command with the MODE parameter missing, or with a MODE parameter that supplies the value SMTP. Hosts that support TLS connections in SMTP mode must ensure the connection does not allow consumers to use any commands using the DMTP protocol syntax. Attempts to issue DMTP commands must result in a response code of 501, denoting an invalid syntax. The lone exception to this rule is the MODE command, which must result in response indicating the connection is in SMTP mode. For consumers able to successfully establish the TLS connection in SMTP, they must be greeted by a 250 response code matching the following pattern:

```
250 OK ESMTP {freeform}
```

Note that both STARTTLS responses supplied above indicate the current protocol mode after completing the TLS handshake using the third token in the response. Thus consumers conforming to this specification must consult the server response returned by a host in response to the STARTTLS command and ensure the connection is using the appropriate protocol mode before proceeding.

Once a connection has been secured using the STARTTLS command a host conforming to this specification must reset all state information for the connection. This includes discarding the host name host name values provided as parameters to the HELO or EHLO commands. If a consumer is proceeding with the connection in SMTP mode then it must issue the HELO or EHLO command before attempting to transfer a message.

TIMEOUTS

Consumers must provide a timeout mechanism for unresponsive server connections, while the enforcement of connection timeouts remains optional for server implementations. Timeouts should be calculated based on the amount of time that has lapsed since a complete line has been transmitted or received. If an implementation is unable to track timeouts based on when the last complete line DMTP protocol line was sent or received, the recommended alternative is to rely on the amount of time since any DMTP characters were sent or received. We strongly recommended avoiding a strategy of relying on the time elapsed since a TLS message, or TCP packet was observed. It is possible for TLS connections to exchange TCP packets indefinitely without ever exchanging any DMTP protocol data.

Server implementations lacking the ability to track timeouts based on the last DMTP character transmitted, or lack support for timeouts altogether, will waste resources on paralyzed client connections. However, if a consumer lacks support for tracking timeouts based on DMTP protocol data, it could result in unnecessary user distress. For a Mail User Agent (MUA), this could result in lengthy send operations, as the User Privacy Agent (UPA) waits for a signet resolution to complete. If the consumer is a Mail Transfer Agent (MTA), this issue could result in messages being rejected, or bounced, because they were delayed beyond the expiry threshold for a stale user signet.

CONSUMERS

Below are the recommended timeouts a consumer should use for the different categories of possible DMTP operations. However, a more sophisticated implementation may choose to use timeouts based on a higher level of granularity than what's provided here. In our experience, such an implementation should be patient when it comes to waiting on commands which involve a large transmission, whether its sending a message or receiving a signet, and with commands which involve the setup process for a DMTP connection, such as the TCP connection setup, a TLS handshake, or the receipt of an initial greeting once the TLS channel has been established. These operations could employ multiple systems, any of which could be suffering from congestion. The following timeouts are intended only to be recommendations, with the one exception being the amount of time between when an MTA finishes transmitting a message and the receiving host acknowledges its acceptance. An MTA must wait at least 8 minutes for such an acknowledgement and may want to wait longer if a message was particularly large.

	Timeout Range *Recommended*
Connection Setup Operations *TCP s*etup, TLS handshakes, Connection *Banners*	4 to 8 minutes
Protocol Mode Elevation Commands *STARTTLS*	4 to 8 minutes
Global Commands *HELO, EHLO, MODE, RSET, NOOP, HELP*	2 to 8 minutes
Mail Processing Commands *MAIL, RCPT, DATA*	8 to 16 minutes
Signet Retrieval Commands *SGNT, HIST, VRFY*	1 to 4 minutes
Connection Termination Command *QUIT*	1 to 2 minutes

SERVERS

A server implementation must use a timeout of at least 1 minute. Servers should employ timeouts between 4 and 20 minutes, with a timeout of 10 minutes being recommended. We recommend that servers that normally employ a timeout shorter than 1 minute, increase their timeout to 4 minutes while processing a mail transaction. This means increasing the timeout after a successful MAIL command, until the transaction is concluded.

While the practice of enforcing timeouts based on the overall time for a DMTP command to complete is not recommended, if a server does employ this strategy, it must ensure consumers are allowed a minimum of 30 minutes to complete the transmission of messages following a successful DATA command, and a similar minimum of 30 minutes to finish receiving a multiline response following a successful SGNT, HIST, or VRFY command.

Sophisticated server implementations may want to dynamically adjust their timeouts based on network congestion to differentiating between TCP congestion and a client that ceases to transmit packet acknowledgements. It may also want to differentiate between the timeout it employs while receiving or transmitting data, and the time it waits for an idle connection to send a DMTP command.

The above timeouts are intended to apply while server is operating normally and should not apply to servers which are in the process of shutting down.

TERMINATION

A DMTP connection should, under normal conditions, only be terminated in response to a consumer sending the QUIT command. Consumers sending this command should wait for the server to acknowledge the receipt of a QUIT command transmit a positive reply.

A server must not intentionally choose to unilaterally terminate a DMTP connection under normal operating conditions unless a consumer has exceeded the configured timeout, or is in the process of shutting down. Specifically, servers must not terminate DMTP connections in response to an unknown command, because of syntax violations, or because a command was sent out of order.

If a server does encounter a situation where it needs to unilaterally close a DMTP connection, it must first transmit a line starting with the status code 421, to indicate the abnormal closure. Presumably, a consumer will receive and process the response while processing the response to a previous command, or asynchronously as the response to its next command. A server should follow this transmission by attempting to cleanly shutdown the TLS connection.

DMTP clients must always be prepared to handle the abnormal shutdown of a connection. This means gracefully handling a DMTP reply which starts with the status code 421, or being notified the of a TLS shutdown. If an MTA experiences an abnormal shutdown during a mail transaction, it must treat the delivery attempt as if a response code of 451 was returned, and ensure the message delivery attempt is retried. Sometimes abrupt communications failures can result in the unexpected closure of connections. Despite being a violation of this specification, this situation will inevitably, and unavoidably, arise and must be handled gracefully. The robustness of DIME depends upon implementations being able to handle failure and retry the aborted operation using a different DMTP host, or when the original host comes back online.

GLOBAL COMMANDS

The following commands must always be available to consumers regardless of the protocol mode or connection state: HELO, EHLO, MODE, RSET, NOOP, HELP, VERB, and QUIT. This includes a connection to a dual protocol host that has not been elevated into DMTP mode.

For signet resolvers, the issuing a HELO or EHLO command is not recommended, and should be avoided to prevent the unnecessary leakage of meta-information about a consumer. If the consumer is an MTA, it must send either the HELO or

EHLO command before attempting a mail transaction. If the MTA is connected to the host using the dual protocol mode, it must send, or resend, the HELO or EHLO after the connection has been elevated to DMTP.

HELO

Consumers may use the HELO command at any time. If the host already has a host name stored for the current connection, it must replace the stored value with the newly issued host name. Unlike the EHLO command below, the HELO command does not list the supported protocol extensions in its reply. This command requires a single parameter in the form of a fully qualified domain name. If the consumer does not have a meaningful host name to supply, it should send an address literal. If a host name is supplied, it should resolve to an address literal matching the current connection. The HELO command uses the following syntax:

```
HELO <domain.tld>
```

Successfully issued HELO commands must result in a single line reply, with a response code of 250 in the form:

```
250 OK {freeform}
```

EHLO

The EHLO command is identical to the HELO command above with one notable exception. A successful EHLO command will result in a reply that lists the protocol extensions supported by a DMTP host. If the host does not support any protocol extensions, then it will result in a reply that is identical to the HELO command. This command requires a host name as the first argument, and uses the syntax:

```
EHLO <domain.tld>
```

The EHLO response may use the multiline response structure. The additional lines will provide keywords, with each corresponding to a protocol extension. DMTP hosts operating in dual protocol mode must return the DMTP and STARTTLS keywords in their response to the EHLO command for connections that have not issued the STARTTLS command and successfully completed a TLS handshake. The following is a potential EHLO response returned by a dual protocol host on a connection that has not been elevated into DMTP mode:

```
250-DMTP
250-STARTTLS
250-PIPELINING
250-SIZE 33554432
250 OK {freeform}
```

In contrast, the EHLO response for single protocol connections, or a dual protocol mode connection that has been elevated into DMTP mode, should never include the DMTP or STARTTLS keywords. The following is a potential EHLO response returned to a consumer over a DMTP connection:

```
250-PIPELINING
250-SIZE 33554432
250 OK {freeform}
```

Note that when a consumer connects to a dual protocol host, it must discard the list of protocol extensions returned by an EHLO command submitted before the connection was elevated. Dual protocol hosts are likely to send a list of protocol extensions after a connection has been elevated into DMTP that is distinctly different from the list sent before elevation.

MODE

The MODE command is the only DMTP command that a dual protocol host should accept before a connection is elevated into DMTP. The MODE command accepts no arguments and is used by consumers to confirm the protocol mode for the current connection. A consumer may issue the MODE command using the following syntax:

```
MODE
```

The response to a MODE command matches the reply issued after a successful STARTTLS command. The 250 response code should be followed by the current protocol mode for the connection in the third token. A DMTP connection must result in a reply matching the syntax:

```
250 OK DMTP {freeform}
```

In contrast a connection operating in SMTP mode, must reply with the following response regardless of whether the connection has been secured using TLS:

```
250 OK ESMTP {freeform}
```

For legacy servers which lack support for DMTP, the DMTP command should result in a 500 response code to indicate the MODE command was unrecognized. Hosts which are DMTP capable, but currently have DMTP support disabled should reply using the 502 response code to indicate the MODE command was recognized, but currently has DMTP support disabled.

RSET

The RSET command is used to reset the state information for a connection. It operates in a fashion that is similar to the STARTTLS command specified above, with the exception that the RSET command does not destroy a host name which was supplied as an argument to the HELO or EHLO commands. The RSET command accepts no arguments and uses the following syntax:

```
RSET
```

A RSET command that is accepted by a server, will return a 250 response code to indicate the state table was successfully reset, with the reply conforming to the syntax:

```
250 OK {freeform}
```

If a consumer encounters a response code other than 250, it must clear the state table by disconnecting from the DMTP and reconnecting.

NOOP

The NOOP command is used by consumers to keep a DMTP connection alive, and should result in no operation being carried out by either host. The NOOP command does not require an argument, but servers must accept NOOP commands that supply command arguments provided they conform to the limitations specified above. This means the entire command line, including the line terminator, must be 512 octets or less in length and only contain valid ASCII character values. The command uses the syntax:

```
NOOP {freeform}
```

Valid NOOP commands must result in a reply using the 250 response code and match the pattern:

```
250 OK {freeform}
```

HELP

The HELP command is used by administrators on interactive DMTP connections to retrieve the list of commands supported by the DMTP host. Support for this command is optional. The command itself does not accept an argument, and is issued using the syntax:

```
HELP
```

Servers with the HELP command enabled may use the multiline response structure and must reply using a response code of 214. It is recommended that server implementations always return the list of available DMTP commands in alphabetical order. The following reply includes a listing of the DMTP commands a host is required to support:

```
214-DATA
214-EHLO
214-HELO
214-HELP
214-HIST
214-MAIL
214-MODE
214-NOOP
214-QUIT
214-RCPT
214-RSET
214-SGNT
214 VRFY {freeform}
```

A DMTP host may choose to disable support for the HELP command. To indicate this, a DMTP server should reply using the 502 response code to indicate the HELP command was recognized but has been disabled:

```
502 COMMAND DISABLED
```

QUIT

The QUIT command terminates a connection gracefully. A DMTP host must initiate send the appropriate response and subsequently initiate a controlled shutdown of the TLS connection. The QUIT command accepts no arguments and uses the syntax:

```
QUIT
```

A DMTP server must acknowledge its receipt of a QUIT command by transmitting a reply using the 221 response code:

```
221 BYE {freeform}
```

If a consumer does not receive a reply from the DMTP server a timely fashion, it may choose to begin the shutdown process in accordance with the TLS protocol specification. [TLS]

MESSAGE TRANSFER COMMANDS

The following commands are used to transfer messages between organizations using atomic mail transactions. The commands have been constructed for securely and reliably delivering messages while minimizing the amount of metadata a compromised handling agent is capable of leaking. The commands described in this section must not be sent by a consumer, or accepted by server, until either the HELO or EHLO command have been sent. These commands require a consumer to provide its fully qualified host name, and for a server to indicate its acceptance of the value by replying with a successful status code. If any of the commands in this section are submitted before a successful HELO or EHLO, a server must respond with the status code 503 to indicate an invalid sequence of commands.

A mail transaction is an atomic transaction requiring a consumer to send, and a server accept all three commands specified in this section, in the sequence: MAIL, RCPT and DATA. If the RCPT or DATA commands are received out of order, then a server must respond with a 503 status code to indicate an invalid sequence of commands.

If the RSET command is received before a mail transaction is completed, then any pending mails transactions must be aborted. A conforming MTA must ensure it retains responsibility for a message until it receives a successful response to the DATA command. This concludes the mail transaction and transfers responsibility for delivering message to the destination host. If a message is accepted by a destination, and it encounters a problem delivering a message, it must generate and deliver a bounce back to the origin domain.

MAIL

The MAIL command is used to start a new mail transaction. The command has two required arguments. The FROM argument must be sent first and is followed by the FINGERPRINT argument. FROM is used to provide the origin domain

name for a pending message, while FINGERPRINT provides the full fingerprint for the origin signet required by a destination host to authenticate the organizational signature attached to the pending message.

A destination host must ensure it has a cached copy of referenced origin signet referenced before replying with a successful status code. If it the origin signet has not already been stored, a destination host may choose to delay sending a response to the MAIL command until it has successfully retrieved, and authenticated the origin signet. However, if this simultaneous retrieval attempt does not completed within 4 minutes, a destination host must reply with the status code of 470. A destination should immediately reply with the status code 470 if it prefers, or is unable, to perform a simultaneous origin signet lookup. The response code 470 is used to indicate an origin signet is temporarily unavailable, and that an MTA must queue the message and reattempt the transfer in the future. If a destination host repeatedly tries and fails to retrieve an origin signet for 72 hours, it should return the response code 570 to any MAIL command referencing the origin signet in question. The response code of 570 is used to indicate the prolonged failure to retrieve the origin signet. The destination host should continue making retrieval attempts until it succeeds, or does an additional 72 hours lapses without encountering a reference to origin signet in question.

A server must only respond to a MAIL command with a success response if a new mail transaction is started. If an MTA sends the MAIL command before completing the pending transaction has been completed, a server must abort the previously started transaction before evaluating the newly submitted MAIL command. As a result, servers must produce identical results for MAIL commands regardless of any potentially pending mail transactions. This also means the outcome of a MAIL command resulting in an error must be semantically equivalent to the outcome of an RSET command resulting in success; both must result the pending mail transaction being aborted without starting a new transaction.

The syntax used to submit a MAIL command with its two required parameters is:

```
MAIL FROM=<domain.tld> FINGERPRINT=[fingerprint]
```

If an MTA attempts submits a MAIL command before it submits a valid HELO or EHLO command, then a server must respond with a response code of 503 to indicate the invalid command sequence:

```
503 INVALID COMMAND SEQUENCE {freeform}
```

If the submitted MAIL command references an origin signet which is unavailable on the destination host, a message should be queued and retried, then a server should reply using the status code 475 and the syntax:

```
470 ORIGIN SIGNET UNAVAILABLE {freeform}
```

If a destination has been attempting to retrieve the reference origin signet for at least the previous 72 hours, then it should indicate a prolonged permanent origin signet failure using the status code 570 and the syntax:

```
570 ORIGIN SIGNET UNAVAILABLE {freeform}
```

If the origin domain lacks a management record, or the authoritative server for the origin domain returns an error when the referenced signet is requested, then a DMTP host should respond using the error code 575 to differentiate it from an origin signet timeout:

```
575 INVALID ORIGIN SIGNET
```

If a destination host does have the referenced origin signet available in its cache, it should allow the transaction to proceed by returning a response code of 250 using the syntax:

```
250 OK {freeform}
```

If the fingerprint does not match what the destination has in its cache for domain.tld, this command would initiate a side channel lookup.

RCPT

The RCPT command is used to confirm a message is being delivered to the correct host and was created using a current and available destination signet. It requires two arguments be provided along with the command, the TO parameter must be followed by the FINGERPRINT parameter. The TO parameter must provide a target domain that the destination host is configured to accept messages for. The FINGERPRINT parameter provides the full fingerprint for the destination signet used to encrypt the recipient information. The RCPT command uses the following syntax:

```
RCPT TO=<domain.tld> FINGERPRINT=[fingerprint]
```

If the destination host needs to reject a message because the fingerprint indicates the recipient information was encrypted using an expired or otherwise invalid destination signet, it should respond with a status of 576, clear any state information associated with the mail transaction and use the syntax:

```
576 INVALID DESTINATION SIGNET {freeform}
```

If a RCPT command is submitted twice a single mail transaction, the second attempt must be rejected using the 431 response code. A DMTP mail transaction is only capable of being associated with a single recipient, so if a RCPT was already accepted, the resource limits would be exceeded by accepting a second RCPT command. The limitation is a byproduct of the D/MIME format, which intentionally limits the envelope to a single recipient, which prevents anyone from discovering how many people a message was originally addressed to. This requires that a message be transferred separately for each recipient as standalone mail transactions. To indicate a rejection resulted from this limitation server should use the following response syntax:

```
431 DESTINATION LIMITS EXCEEDED {freeform}
```

If the RCPT parameters indicate a recognized destination domain and the fingerprint indicates the embedded recipient information will be accessible a server should reply using the status code 250 to allow the MTA to proceed with the transaction by sending the message data. The success response syntax is:

```
250 OK {freeform}
```

DATA

The DATA command is used to transfer a D/MIME message to the destination host. Provided the MTA has successfully issued MAIL and RCPT commands, a DATA command should result in a 354 response code, indicating the destination host is ready to receive the message. A client should proceed to transmit the D/MIME message in its ASCII armored form. The transmission sequence is terminated by the string "<LF>.<LF>" which is sent to indicate the transmission has been completed. The sender must then wait at least 8 minutes for a reply, presumably a sender should wait at least 1 additional minute for every megabyte used by the transmitted message. If the DMTP host responds with the code 254, then the mail transaction is complete. A response code between 400 and 499 will indicate the current attempt failed, but the issue was temporal and the sender should retry the transmission later. A response code over 500 indicates a permanent failure, that the error is likely to persist, and that an origin host should proceed to notify the author of the failure. Once the 254 code has been sent, responsibility for the transmitted message shifts from the origin MTA to the destination MTA. If a DMTP host discovers after it has transmitting the 254 response code that it is unable to deliver a message, then it must bounce the message back to the origin to ensure the author is properly notified of the failure. Alternatively, if a message is delivered, but the 254 response code is never received by the sender, because the disconnected before the 254 response was received, when it retransmits the message, it is possible the retransmission will only result in the message being duplicated in a recipient's mailbox. Sophisticated server implementations may want to detect this issue by tracking the cryptographic hashes for any recently delivered messages and compare those hashes against incoming messages. If a duplicate message is detected, then a host may return the response code 255 which indicates the message successfully delivered on a previous attempt.

The transmission process begins with a singular DATA command. The default DATA command does not allow arguments to be included. The syntax for the command is:

```
DATA
```

If the DMTP host is ready to receive the message it will respond using the 354 banner shown below:

```
354 READY TO RECEIVE MESSAGE
```

Once a sender sees the 354 response, they may begin transmitting the message. The sequence "<LF>.<LF>" is used to terminate the message transmission:

```
-----BEGIN ENCRYPTED MESSAGE-----
Bv0AAdcBhvAmjVKiMZmjF8gTnXNTDZ4C1W8MSWfh5NLIdzquujQCBJkg4dcp7m8jklKZtjp7JFrWkowv1bp1Yga1pIJNyIbbhjax
Y0CpFaF4z2L8mjcJq5Pl+J/lF4iKrJc7tJYWCueGeJiYgQci0vKUiRHqyr1wkjMUbmdY9Zog5/554udPiAVzHJLNplUj6ZtjmmdA
bSeJhM4nrLzQe5wXR6n8fMDsHtJvZNb1PZSMycs7rMoNDEY6pjjo8Y70k0E3jLy9SHcCBhA78k9y8JEDR8bF4RzT7AeM7Udi8oGA
wooUGwENp3upYuhxd/bzoZg53TdQbNM2RKcGKozSQK2gHKpFI59gjwcZBUxhZGFyGwRDYXZlHhJDb3VudHJ5IHdpdGggY2F2ZXMf
FXppcGNvZGUgd2l0aG91dCBjYXZlcyADNDExfjd0pQ0k4DXBXvBfUNNFxir+IzghryyCr67G9jEa44VD8Q1E3j7EqVW1xC/TF2KG
mfylpmL2iueTyPz50kAY9Qd/EWxhZGFyQGxhdmFiaXQuY29tgI7gaXq2Nu7dVKmu8i78jjB1uOeU8Vbjj47aXZzQUM9L79WuqTuL
dMC2yD4vW76cGkb8hrGL/y8H0IshRpNeOAM
-----END ENCRYPTED MESSAGE-----
.
```

Upon receiving a message, a server must ensure the D/MIME is structurally correct and contains a valid organizational signature from the origin domain before accepting a message. If the message is does not contain a valid D/MIME binary structure, the DMTP host should immediately return a 451 error code:

```
451 DATA CORRUPTED
```

If the organizational signature for the message is missing, or invalid, then a server must return the 578 error code:

```
578 INVALID ORIGIN SIGNATURE
```

A server may also decrypt the destination portion of the D/MIME message and confirm the validity of the recipient, and whether the recipient signet used to encrypt the message is either current, or is within the expiry threshold for stale user signets. Alternatively, a DMTP host may also accept a message and commit to bouncing it later if these checks fail. If the message is validated before responding, and the recipient mailbox is invalid, or not affiliated with the destination domain provided by the RCPT command, then a DMTP host must respond using the 510 error code:

```
510 INVALID RECIPIENT
```

If the message was encrypted using an expired user signet, then a DMTP server must respond using the 586 error code:

```
586 INVALID RECIPIENT SIGNET
```

If received message passes all of the checks described above, then the message should be queued for delivery and the 254 response code returned to the sender along with a cryptographic hash of the binary message data received, and transmitted in its base 64 encoded form, without padding, yielding a message transaction identifier that is precisely 86 bytes long:

```
254 ACCEPTED=MUYwQkNENDY0MzE3OEQyOTAxRDEwMjlFRUREQTJBQTJCNTJCRThDQUZEOTM4NkY5NjhFODVFMUE5RDE5NTUxMg
```

If the host tracks the cryptographic hashes for recently accepted messages, and a duplicate message is detected, then it should return the 255 response code to indicate the message has already been delivered. This response must only be returned if the message has already been delivered to the mailbox. If the previous transfer attempt failed, then it must not be considered a duplicate. Successful deliveries result in a response using the syntax:

```
255 DUPLICATE=MUYwQkNENDY0MzE3OEQyOTAxRDEwMjlFRUREQTJBQTJCNTJCRThDQUZEOTM4NkY5NjhFODVFMUE5RDE5NTUxMg
```

Regardless of the response code, a sender must consider the mail transaction terminated. If it intends to retransmit the message, or begin the transmission of a different message, it must begin the command sequence again using the MAIL command.

SIGNET TRANSFER COMMANDS

SGNT

The SGNT command is used to retrieve user and organizational signets from an authoritative source using DMTP. The command requires a consumer to supply the FOR argument, and may be submitted along with the FINGERPRINT argument. The FOR argument must be sent first, and a FINGERPRINT value, if supplied, must be sent second. The SGNT command syntax for retrieving an organizational signet is:

```
SGNT FOR=<domain.tld>
```

An almost identical syntax is used for retrieving user signets, with the singular syntactical exception that a mailbox is supplied using the FOR argument:

```
SGNT FOR=<mailbox@domain.tld>
```

A consumer may want to retrieve a specific version of a user or organizational signet, possibly because the fingerprint for was supplied using the MAIL command above, or because it is trying to retrieve a signet referenced elsewhere. To retrieve a specific signet a consumer would use the second optional argument, which accepts a full fingerprint for the requested signet, after it has been base 64 encoded, and the padding bytes removed. The resulting values for the FINGERPRINT argument should always be exactly 86 bytes. The complete syntax for SGNT command syntax when retrieving a specific organizational signet:

```
SGNT FOR=<domain.tld> FINGERPRINT=[fingerprint]
```

When a consumer is requesting a specific user signet, it may supply either the full fingerprint or the core fingerprint for the user signet it wants to retrieve. Note that a server must always return a full signet in response to the SGNT command, even if a core fingerprint is submitted. The syntax for retrieving a specific user signet is:

```
SGNT FOR=<mailbox@domain.tld> FINGERPRINT=[fingerprint]
```

A conforming server implementation must only return organizational signets for domains in which it is the authoritative source. If the requested organizational signet is available it must be returned in its ASCII armored form, and if the fingerprint argument is omitted, a host must return what is considers the current organizational signet for the supplied domain name. When returning an organizational signet, a server must use the multiline syntax and the 270 response code:

```
270------BEGIN ORGANIZATIONAL SIGNET-----
270-BvAAAWEBQt1Wjk8S+DkuEbOLgfQTvVyS7Ae7NjwonNLI+TRoDYUCBOYl1CpVe8l1ny9ceb/SnE7p0FZjZYsA6W9hPLpFTjv+
270-BpyT6l4bHZj3Pd0s9QGE0rXCy9PWsCPwAmFC2aVVcG3NTXsQ5VhYPjK/l3aONDtmz3LS1lKgkFv9B/wB8XzIGhkLTGF2YWJp
270-dCBMTEMbGzEyMyBIaWRkZW4gQnVua2VyIEJvdWxldmFyZB4PUG9zdC1TaW5ndWxhcmlhHwUwMDAxMSALMTgwME5PUEhPTkV+
270-EqMnb0cbDDFBatu9tTMAi7ERNkWGLqWda2IG0oTP22njpchB2KEWjp7QF/qC0byTh7Is+YexkCT+xz0y5GL3AX8LbGF2YWJp
270-dC5jb22AFjee+3raziK2GZYoFErVAsJKXbRc2Zxu1Z3oXAJ1ay9fY/GNihmZgd9SBZrJUnu8XA99RKQrRJln4In121t4AA
270------END ORGANIZATIONAL SIGNET-----
270 OK {freeform}
```

Like organizational signets, a conforming server implementation must only return user signets for domains in which it is the authoritative source. If the requested user signet is available it must be returned in its ASCII armored form, and if the fingerprint argument is omitted, a host must return what it considers to be the current user signet. When returning a user signet, the full signet must always be returned, even if the consumer supplies a core fingerprint. When returning a user signet, the server must use the multiline syntax and the 280 response code:

```
280------BEGIN USER SIGNET-----
280-Bv0AAdcBhvAmjVKiMZmjF8gTnXNTDZ4C1W8MSWfh5NLIdzquujQCBJkg4dcp7m8jklKZtjp7JFrWkowv1bp1Yga1pIJNyIbb
280-hjaxY0CpFaF4z2L8mjcJq5Pl+J/lF4iKrJc7tJYWCueGeJiYgQci0vKUiRHqyr1wkjMUbmdY9Zog5/554udPiAVzHJLNplUj
280-6ZtjmmdAbSeJhM4nrLzQe5wXR6n8fMDsHtJvZNb1PZSMycs7rMoNDEY6pjjo8Y70k0E3jLy9SHcCBhA78k9y8JEDR8bF4RzT
280-7AeM7Udi8oGAwooUGwENp3upYuhxd/bzoZg53TdQbNM2RKcGKozSQK2gHKpFI59gjwcZBUxhZGFyGwRDYXZlHhJDb3VudHJ5
280-IHdpdGggY2F2ZXMfFXppcGNvZGUgd2l0aG91dCBjYXZlcyADNDExfjd0pQ0k4DXBXvBfUNNFxir+IzghryyCr67G9jEa44VD
```

```
280-8Q1E3j7EqVW1xC/TF2KGmfylpmL2iueTyPz50kAY9Qd/EWxhZGFyQGxhdmFiaXQuY29tgI7gaXq2Nu7dVKmu8i78jjB1uOeU
280-8Vbjj47aXZzQUM9L79WuqTuLdMC2yD4vW76cGkb8hrGL/y8H0IshRpNeOAM
280------END USER SIGNET-----
280 OK {freeform}
```

If a fingerprint parameter is provided, then a host must return the signet matching the fingerprint, or an error. If no signet is available for the requested address, then a server must also return an error.

```
576 SIGNET UNAVAILABLE
```

If the domain or address is valid, but the signet is unavailable, a server may choose to return the error code 476 instead. If the domain advertises a policy of experimental in its management record, then a consumer may choose to send the message using SMTP if this error is received. Otherwise clients must either retry the request later, or return an error to the message author.

```
476 SIGNET TEMPORARILY UNAVAILABLE
```

HIST

Allows a resolver to retrieve the chain of user signets between a trusted signet fingerprint (START) and a recently encountered user signet (END). If both fingerprint values are valid, then the host should return only the core portion of the user signets published between the two values. If the end fingerprint value is missing the client provides all of the core signets until it reaches the current user signet. This command must not be used to retrieve organizational signets.

The HIST command has one required argument, and two optional arguments. The FOR argument is required and used to provide the email address being queried. The FOR argument must always come first. If provided, the START argument must follow the FOR argument, and provides the core fingerprint for a user signet at the start of a chain of custody query. The final argument is STOP and if included, provides a core fingerprint for the last user signet that needs to be returned. If the START parameter is missing, then a DMTP server should return the first signet a user's current chain of custody. If the STOP parameter is missing, then the server should provide the all of the user signets between the START value and the current user signet. If both arguments are missing then a server must return the entire chain of custody for the current user signet.

```
HIST FOR=<mailbox@domain.tld> START=[fingerprint] STOP=[fingerprint]
```

A DMTP server must be capable of providing the core signets in a user's chain of custody, from the root, all the way to the current user signet. A server may provide user signets beyond a user's current chain of custody, but should only return these if provided a starting fingerprint that reaches past the current user signet's root. Results are provided using the multiline syntax, and the 290 response code:

```
290------BEGIN USER SIGNET-----
290-Bv0AAQUBhvAmjVKiMZmjF8gTnXNTDZ4C1W8MSWfh5NLIdzquujQCBJkg4dcp7m8jklKZtjp7JFrWkowv1bp1Yga1pIJNyIbbh
290-jaxY0CpFaF4z2L8mjcJq5Pl+J/lF4iKrJc7tJYWCueGeJiYgQci0vKUiRHqyr1wkjMUbmdY9Zog5/554udPiAVzHJLNplUj6Z
290-tjmmdAbSeJhM4nrLzQe5wXR6n8fMDsHtJvZNb1PZSMycs7rMoNDEY6pjjo8Y70k0E3jLy9SHcCBhA78k9y8JEDR8bF4RzT7Ae
290-M7Udi8oGAwooUGwENp3upYuhxd/bzoZg53TdQbNM2RKcGKozSQK2gHKpFI59gjwc
290------END USER SIGNET-----
290 OK {freeform}
```

If the start or end fingerprint values fall outside of the current user signet's current chain of custody, then a server may return the 576 response code. A server should also return the 576 response code if the user signet requested is unavailable.

```
576 SIGNET UNAVAILABLE
```

VRFY

Allows a consumer to confirm whether a particular signet is still current, or needs to be retrieved again. This command must accept core fingerprints for users and full fingerprints for organizations, and should reject requests where the consumer supplies a full fingerprint for a user signet. The VRFY command requires the FOR and the FINGERPRINT arguments, and uses the following syntax:

```
VRFY FOR=<domain.tld> FINGERPRINT=[fingerprint]
```

Or to verify a user signet is still current:

```
VRFY FOR=<mailbox@domain.tld> FINGERPRINT=[fingerprint]
```

If the organizational signet is current the following is returned:

```
271 ORGANIZATIONAL SIGNET CURRENT {freeform}
```

Or if a user address was supplied and the signet is still current:

```
281 USER SIGNET CURRENT {freeform}
```

Otherwise an update is returned using the same syntax as the SGNT command. Where an organizational signet uses the 270 response code:

```
270------BEGIN ORGANIZATIONAL SIGNET-----
270-BvAAAWEBQt1Wjk8S+DkuEbOLgfQTvVyS7Ae7NjwonNLI+TRoDYUCBOYl1CpVe8l1ny9ceb/SnE7p0FZjZYsA6W9hPLpFTjv+
270-BpyT6l4bHZj3Pd0s9QGE0rXCy9PWsCPwAmFC2aVVcG3NTXsQ5VhYPjK/l3aONDtmz3LS1lKgkFv9B/wB8XzIGhkLTGF2YWJp
270-dCBMTEMbGzEyMyBIaWRkZW4gQnVua2VyIEJvdWxldmFyZB4PUG9zdC1TaW5ndWxhcmlhHwUwMDAxMSALMTgwME5PUEhPTkV+
270-EqMnb0cbDDFBatu9tTMAi7ERNkWGLqWda2IG0oTP22njpchB2KEWjp7QF/qC0byTh7Is+YexkCT+xz0y5GL3AX8LbGF2YWJp
270-dC5jb22AFjee+3raziK2GZYoFErVAsJKXbRc2Zxu1Z3oXAJ1ay9fY/GNihmZgd9SBZrJUnu8XA99RKQrRJln4In121t4AA
270------END ORGANIZATIONAL SIGNET-----
270 OK {freeform}
```

Or if a user address was supplied that was updated, the 280 response code is used to return the updated user signet:

```
280------BEGIN USER SIGNET-----
280-Bv0AAdcBhvAmjVKiMZmjF8gTnXNTDZ4C1W8MSWfh5NLIdzquujQCBJkg4dcp7m8jklKZtjp7JFrWkowv1bp1Yga1pIJNyIbb
280-hjaxY0CpFaF4z2L8mjcJq5Pl+J/lF4iKrJc7tJYWCueGeJiYgQci0vKUiRHqyr1wkjMUbmdY9Zog5/554udPiAVzHJLNplUj
280-6ZtjmmdAbSeJhM4nrLzQe5wXR6n8fMDsHtJvZNb1PZSMycs7rMoNDEY6pjjo8Y70k0E3jLy9SHcCBhA78k9y8JEDR8bF4RzT
280-7AeM7Udi8oGAwooUGwENp3upYuhxd/bzoZg53TdQbNM2RKcGKozSQK2gHKpFI59gjwcZBUxhZGFyGwRDYXZlHhJDb3VudHJ5
280-IHdpdGggY2F2ZXMfFXppcGNvZGUgd2l0aG91dCBjYXZlcyADNDExfjd0pQ0k4DXBXvBfUNNFxir+IzghryyCr67G9jEa44VD
280-8Q1E3j7EqVW1xC/TF2KGmfylpmL2iueTyPz50kAY9Qd/EWxhZGFyQGxhdmFiaXQuY29tgI7gaXq2Nu7dVKmu8i78jjB1uOeU
280-8Vbjj47aXZzQUM9L79WuqTuLdMC2yD4vW76cGkb8hrGL/y8H0IshRpNeOAM
280------END USER SIGNET-----
280 OK {freeform}
```

RESPONSE CODES

Code	Description
214	HELP
221	BYE
250	OK
254	ACCEPTED=*identifier*
255	DUPLICATE=*identifier*
270	OK
271	ORGANIZATIONAL SIGNET CURRENT
280	OK
281	USER SIGNET CURRENT
290	OK
291	USER SIGNET CURRENT
354	READY TO RECEIVE MESSAGE
431	DESTINATION LIMITS EXCEEDED
450	ACCESS DENIED
451	DATA CORRUPTED
470	ORIGIN SIGNET UNAVAILABLE
500	COMMAND SYNTAX ERROR
501	ARGUMENT SYNTAX ERROR
502	COMMAND DISABLED
503	INVALID COMMAND SEQUENCE
510	INVALID RECIPIENT
570	ORIGIN SIGNET UNAVAILABLE
575	INVALID ORIGIN SIGNET
576	INVALID DESTINATION SIGNET
578	INVALID ORIGIN SIGNATURE
586	INVALID RECIPIENT SIGNET

PROTOCOL EXTENSIONS

SIZE

TBD

BINARY

TBD

UNICODE

TBD

PIPELINING

TBD

SURROGATE

Indicates the true destination host indicated by the TLS SNI extension, or as an argument to the STARTTLS command could not be reached. However, the current host will act as a surrogate to accept and relay the D/MIME message onto its destination when the host becomes available. This extension allows individuals to host a DIME server at home, without revealing the destination host address literal to a consumer, and allow consumers to consume DMTP services for the target domain even when the destination host is offline by offering signets and accepting messages for future delivery.

---

[11] The NIST name, and the one reused by the referenced TLS standard is TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384.

[12] Should we require ECDHE-ECDSA-AES256-GCM-SHA384 instead? That would allow us to avoid RSA altogether. The problem is that currently very few CA's publish certificates signed using ECDSA. Alternatively, should we make support for the DHE variants optional? That is DHE-RSA-AES256-GCM-SHA384 or DHE-ECDSA-AES256-GCM-SHA384?

PART 8: DARK MAIL ACCESS PROTOCOL (DMAP)

This protocol specification will not be released as part of the initial publication of this document.

However, the following points are a few of the DMAP features that will be available in a future release of this specification:

- Similar to IMAP, however, the protocol will not include server-side search because all email is encrypted on the server
- It will handle the submission from the MUA for outbound messages
- It will handle the Signet Signing Request (SSR) process
- Authentication will be handled using a cryptographic key process

PART 9: THREATS AND MITIGATIONS

A user's concern for private email exchanges can involve protection of basic content or extend to their social network information – who they exchange mail with, and when – and can vary by the amount of trust they place in their email service provider. Dark Internet Mail Environment (DIME) builds upon classic Internet Mail [IMA] and provides strong privacy protection using encryption, covering metadata, overall message structure, and individual message content including attachments. DIME also ensures message authenticity, integrity and verifiable non-repudiation.

Privacy exposure can be due to passive or active third-party impostors, with wiretapping that captures message traffic over the wire, compromise of a mail handling host or a key management host, or collaboration by a host operator. DIME's design provides a range of protections that combine to defend against each of these categories of threats.

Key management by end users, and even system operators, is a major barrier to the use of security-related services. Therefore, to the extent possible, DIME's encryption details are designed to operate automatically. Great care has been taken to make it difficult for an attacker to subvert the automated aspects of the system undetected. Because error messages and security warnings can be confusing to users, the system provides for alternate mechanisms so clients can overcome common anomalies without compromising security or requiring user intervention. The goal is to create a system sufficiently resilient so that the occurrence of a non-recoverable security error is most likely to be due to system compromise, or because someone in a privileged network position is attempting to carry out an attack.

Service providers occupy a trusted position in the DIME ecosystem. However, a client can choose among three service trust levels to considerably narrow this dependence. In particular, it determines a server's access to the user's private keys using account modes (Trustful, Cautious, Paranoid).

This document discusses the privacy goals for the DIME protocols and formats, how those goals are achieved and what assumptions are made. [SPARROW] A core goal is attending to different types of users and their trust of an associated organization server. We highlight assumptions, and detail specific aspects of the design intended to mitigate common attack vectors. Unless otherwise noted, this document assumes the "Cautious" account mode is being utilized.

THREATS

VENUES

The primary concern is unauthorized information disclosure, that is, situations where the user loses control over the release of their private information. Different types of information need different types of protection. A related concern is authentication of the participants in an exchange, both end users and service providers, so that fraudulent content is avoided.

The types of information compromise of concern include:

Author spoofing: Whether the purported creator and submitter of a message is the actual agent of action.

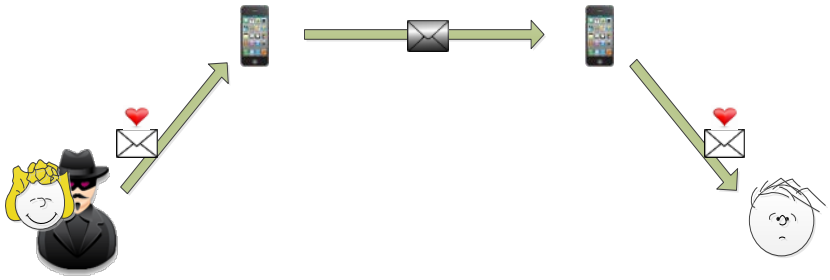

Figure 9 - Author Spoofing

Service provider spoofing: Knowing that the intended provider (mail, key, DNS) is the actual provider.

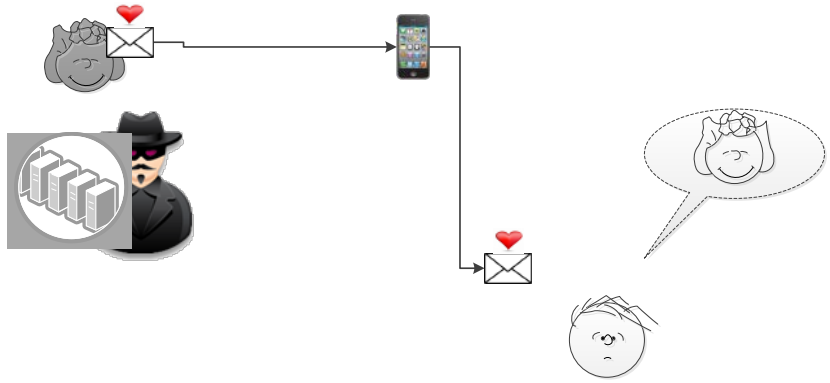

Figure 10 - Service Provider Spoofing

Message content disclosure: Limiting disclosure only to authorized parties -- recipients.

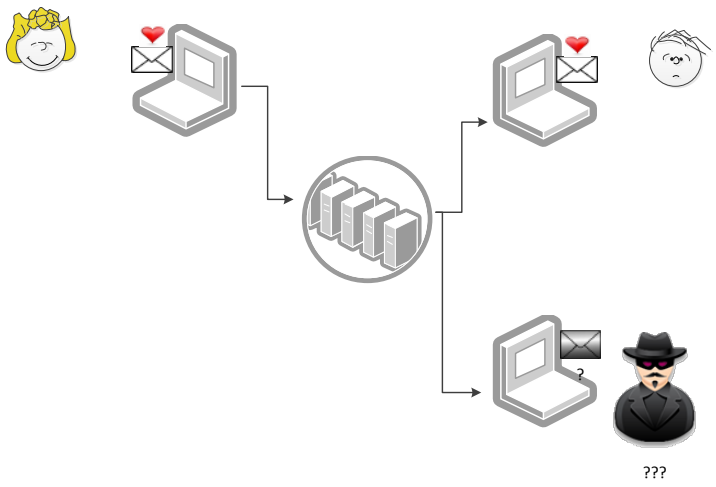

Figure 11 - Message Content Disclosure

Message structure disclosure: Even without knowing the detailed content, knowing about message size, attachment structure, and attachment data types can help an attacker.

Metadata disclosure: Any other *s*tructured data, involving participant and message attributes, which can be stored and subjected to social and network traffic analyses. This includes relationships and activity. Who is talking with whom; when and how actively?

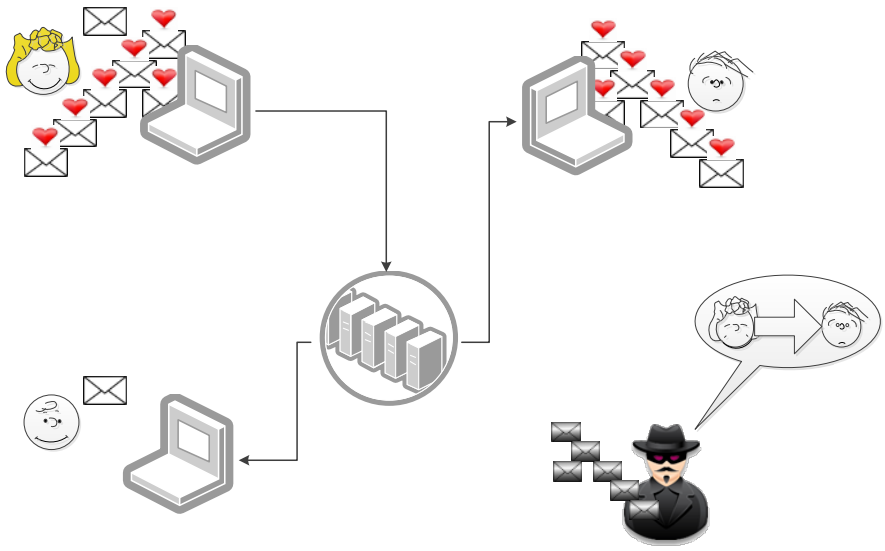

Figure 12 - Metadata Disclosure

VECTORS

A variety of avenues can be exploited to achieve unauthorized disclosures:

Password: The basic unit of local authentication within a system.

"[A challenge is] how to authenticate securely with the service provider without revealing the password (since the password is probably also used to encrypt the private key and other secure storage, so it is important that the service provider does not have cleartext access as with typical password authentication schemes)." [SPARROW]

Key: "[P]ublic-key encryption to allow[s] a user to send a confidential message to the intendant recipient, and for the recipient to verify the authorship of the message. Unfortunately, public-key encryption is notoriously difficult to use properly, even for advanced users. The very concepts are confusing for most users: public key versus private key, key signing, key revocation, signing keys versus encryption keys, bit

length, and so on. This is where we are now: we have public key technology that is excessively difficult for the common user, and our only methods of key validation have fallen into disrepute." [SPARROW]

Organizational Signet: Information tied to a specific domain name, including the public keys associated with that domain name. The authoritative source and verification information for an organizational signet is advertised using a DIME management record in the DNS system and is considered authentic when retrieved from an authoritative DMTP server and validated by the DIME management record. No further validation steps are necessary if the management record was signed using DNSSEC. The organizational signet may also carry with it policy information for the domain. Compromising the private keys associated with an organizational signet or replacing an organizational signet with a fraudulent one could allow an attacker to generate fake user signets and otherwise spoof the organization identity.

User Signet: Information included with a person's public key that helps others verify that a key is genuine or valid; it can carry related profile information for the entity being identified. Assessing signet validity is a distinct step. Compromising the user signet resolution process could allow an attacker to advertise fraudulent public keys allowing them to spoof an identity or access encrypted message contents only if the victim later uses the spoofed ID. A user signet is considered authentic when a retrieved from an authoritative DMTP host and the signature is authenticated using the keys contained within organizational signet.

Domain name: Domain names are basic unit of global identification on the Internet. Domain names are associated with records of information through public queries of the Domain Name Service. Trusting DNS servers, or at least DNS records, is the foundation for email service. The primary long-term path for improving that trust is the widespread adoption of DNSSEC.

Transmission Channel: Monitoring traffic across a transmission link (wiretapping) can be simply passive copying or it can be active spoofing via a man in the middle attack (MitM) that relays messages between both ends, making them believe that they are talking directly to each other over a private connection. TLS is the primary means of protection against wiretapping; MitM protection requires that the server's X.509 certificate is validated using a CA form a certificate authority, or using a MR signed using DNSSEC, authenticating the server's affiliation with the owner of the target domain name.

Client: A compromised end-point permits the attacker to impersonate the user or, at least, to see all of the user's data. It could also permit the attacker to steal keys and passwords, obtain cleartext message information, or otherwise weaken on-going services to

facilitate later interceptions by introducing malware. Additionally, a poorly implemented client or MUA could break the cryptographic mechanisms employed by DIME.

Mail Server: A compromised mail server (MSA, MTA, MDA, MS) can access any mail information that is in the clear or that the server is able to decrypt. Depending upon the capabilities of the client and the account mode, the amount of trust a user must place in their mail server can be greatly reduced.

Key Server: Compromising a server that holds private encryption keys permits an attacker to decrypt data and thereby break DIME's protection. Redundant sources for signet information can aid in the detection of compromised key servers attempting MitM attacks.

DNS Server: A compromised server can permit creation of false records under a target domain name. DNSSEC authenticates records, independent of the server providing them.

Gateway: Transition between a protected email environment, such as DIME, and an unprotected one, such as naked Internet mail, usually requires operation of service gateways. They create opportunities for spoofing and downgrade attacks.

Persistence: Advanced Persistent Threats (APT) typically entails an attacker with a privileged network position, ability to perform extensive and long-term data collection and apply massive computational resources. This creates its own line of attack, beyond those vectors normally of concern.

MITIGATION STRATEGIES

DIME minimizes information that is exposed to intermediaries along the mail-handling path, including what is available to the initial origin and destination service providers. Content is protected by multiple layers of encryption reducing reliance on single-points of failure for providers of keys and signets.

MESSAGE PROTECTION

A message is a hierarchical object, comprising several distinct handling-related and payload components. This permits efficient handling of distinct portions over limited channels and by clients with limited capabilities, as well as permitting separable protection. Only a thin "outer" component of the message transits with unencrypted information.

In terms of handling and protection, each copy of a message is between the author and one recipient. The basic message handling model has two-levels, with an *organization* component and a *user* component. The organization provides public-facing services, at the granularity of a domain name. An individual user's involvement with a message,

such as their full email address, is visible only to their associated organization server and the other end user associated with this message.

The basic message protection model encrypts the entire message, as well as each component, using a different key for each portion that is encrypted. This permits independent handling of different message components and protects envelop information by encrypting those portions with different user and organizational keys.

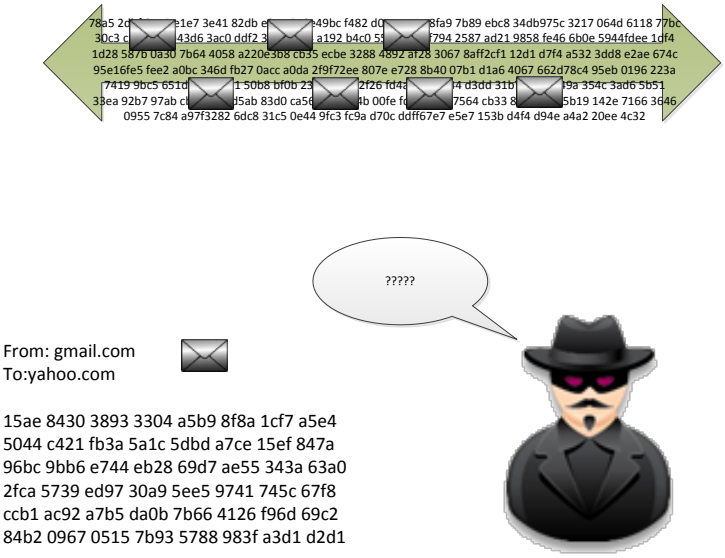

Figure 13 - Basic Message Protection

ACCOUNT MODES

A user's reliance on an associated organization server can be at three different service trust levels, selectable by the user:

Trustful: Comparable to the level of trust placed in a service provider for typical email services historically. In effect, the service handles all privacy issues on behalf of the user. DIME provides protection for messages in transit over the Internet, but the end-user's service provider is fully trusted. In particular, the server has direct access to the user's private keys. Users access email using traditional access via SMTP and IMAP over SSL. Although it is implementation specific, it is recommended that the user's private key be protected using the user's password.

Cautious: In this mode the server holds encrypted copies of a user's private keys and messages. This is convenient for multi-platform users, while reducing the amount of information a compromised service provider can disclose. Because the service provider never has access to the decrypted private key, they are unable to access a user's messages, or publish new user signet without triggering a break in the chain of custody. This mode is designed to facilitate the adoption of DIME without requiring end users to modify their behavior to obtain the additional benefits of encryption without the traditional encryption costs.

Paranoid: This mode provides the server with almost no user security information. In particular, the server never has access to the user's private keys, even in encrypted form.

A thin client is more dependent upon the service provider, since it has few, or none of its own, independent capabilities. Webmail is typically an example of complete reliance on the provider, since any software running on the client comes from the provider; however, a proper thin client implementation that performs encryption in the user's browser will not have complete access to all user information. In the event a thin client is exploited by an attacker to contain malicious code, it could circumvent security to gain access to user information. The recommended approach is a thick client independently obtained and installed and fully under the control of the user.

VECTOR MITIGATION

The following discussion explores the likely approaches for preventing or detecting problems in each part of the system subject to attack.

PASSWORD

User access to a server is controlled through an account password. It is used to authenticate with a server; *however is never sent to the server*. Rather the password is used to derive information that is sent. The server only stores a pre-nonced hash and account key pair, with the private account key being encrypted by the password on the user's device. Hence, if the server is compromised it cannot reveal the password, or even provide the required elements to successfully spoof authentication. The amount of entropy associated with a user's password is improved with user specific salts, and the number of hash rounds being varied based on plaintext length. Organizations can further improve passwords by imposing a variable number of additional hash rounds.

SIGNET

Signet Assignment: Signets are associated with an organizational domain or a user address based on the semantic context of a signet resolver query.

NETWORK PACKET CAPTURE

FORWARD SECRECY

"Traditional schemes for forward secrecy are incompatible with the asynchronous nature of email communication, since with email you still need to be able to send someone a message even if they are not online and ephemeral key generation requires a back and forth exchange between both parties.

"…Another possible approach is to use traditional encryption with no support for forward secrecy but instead rely on a scheme for automatic key discovery and validation in order to frequently rotate keys. This way, a user could throw away their private key every few days, achieving a very crude form of forward secrecy." [SPARROW]

Network level packet captures are impossible with DIME because all connections are protected using TLS v1.2 and require the use of a cipher suite which provides for perfect forward secrecy. If an organization can protect their TLS private key, then they can ensure attackers are also unable to MitM their TLS connections and can achieve perfect forward secrecy (PFS) at a wire level.

PFS for message objects, as the description above suggests, is far more difficult, and contrary to the nature of email. However, a DIME user using the "paranoid" account mode could still obtain PFS for messages by routinely rotating their signet, and destroying the private keys associated with their former signet once the expiry threshold has been reached. Because the private keys were never synchronized with the server, the user can be assured that deletion means the keys could never be recovered, thereby providing PFS even if the messages were intercepted and recorded by a server.

SIGNET AND KEY MANAGEMENT

BASIC MANAGEMENT AND OPERATION

No single source of key information is automatically accepted by the entity making the query. It always must have a confirmation.

Key creation:

1. Trustful Mode: The user signet and the corresponding private keys are generated on the server. The server appends the organization signature plus optional attributes such as name, address, telephone, etc. and a second organizational signature. The two organization signatures allow the cryptographic portion of the user signet to be split from the optional attribute portion. The server stores this signet internally and makes it available via DMTP.
2. Cautious Mode: The desktop client generates a Signet Signing Request (SSR) and the corresponding private keys and submits to server over DMAP with the private keys encrypted. The server appends the organization signature plus optional attributes such as name, address, telephone, etc. and a second organizational signature. The server stores this signet internally and makes this available via DMTP. The encrypted private keys are available to the desktop client via DMAP.
3. Paranoid Mode: The desktop client generates the SSR and submits the SSR to the server over DMAP. The server appends attributes (such as names, address, telephone, etc.) and organization signature. The server stores this signet internally and makes this available via DMTP. The encrypted private key is stored on the desktop client and never transferred to the server.

Signet discovery:

The desktop client performs a lookup of the management record for a domain using DNSSEC. If there is a DIME management record (MR), it retrieves the Primary Organization Key (POK) from the MR and the organizational signet via a DMTP connection. The organizational signet is validated against the POK retrieved via DNS. If

a DIME MR is not signed using DNSSEC, the DMTP server must use a TLS certificate validated by a recognized certificate authority (CA).
The DMTP server will respond to queries for user and organizational signets. Note that some clients might not be able to make direct TCP/TLS connections to a DMTP server because of firewall rules; they will need to proxy requests through their local DIME key server, presumably over an authenticated DMAP connection. This could create an additional avenue for metadata to leak, such as what signet a user retrieves.

Signet validation: A signet is validated by a confirming query via DNS, in addition to the primary means of obtaining and validating it. For an organization-level signet, the secondary query can be via a pre-authenticated source (recognized CA) or DNSSEC. For a user-level signet, confirmation is through a chain of custody if the signet is already in the user's signet cache, in addition to confirmation of an organization signature.

Signet availability: Organization and user signet availability will vary based on deployment decisions and user configuration options. From an organizational viewpoint, access to the organization's private key will be required for signing operations and decryption of delivery information. This will require every DMTP server to have access to the private key, or for more sophisticated deployments, access to a centralized key server that performs all of the organizational level cryptographic operations. Note the trustful/cautious/paranoid modes for end-users; they can choose to share the unencrypted private keys with the server, just the encrypted private keys, or nothing at all. Which option they choose will determine how they can access their account, and where user level cryptographic operations occur.

Signet revocation: To revoke a potentially compromised user signet, a user simply needs to publish a replacement public signet and wait the specified time-to-live for the compromised signet to expire. Once the TTL expires, servers will have to query for the signet again. When an organizational signet is compromised, all existing user signets must be resigned and republished. Because of the potential overhead for large organizations, this issue further stresses the requirement that each organization must protect their corresponding organizational private keys at all costs. If an organizational signet is NOT compromised, but simply changed, the previous organizational keys can be added to the new organizational signet as secondary keys[13].

Key rollover: A chain of custody is established for a sequence of signets. As a new signet is introduced, it is signed by its predecessor signets. This permits automatic acceptance of a new signet when the previous one is already in a user's signet ring. It is based on

the reasonable assumption that the owner of the new signet had access to the private key associated with the trusted signet.

ORGANIZATIONAL_SIGNET

An organizational signet is generated by a system administrator who installs the key into a DMTP configuration and associates the signet with a domain. The administrator configures the DNS for the domain in question to provide the associated validation record. Because of the manual process associated with publishing new organizational signets, the assumption is they will change infrequently. While user signets will have TTL values specified in minutes, organizational signets would use TTL values measured in days; the recommendation is organizations will change signets every 1 to 3 years and have high TTLs (16-32-64 days).

An organizational signet, and its associated private keys, is used to:

- Sign user signets
- Sign outbound messages
- Decrypt 'recipient' chunk on received messages
- Decrypt 'author' chunk for outbound messages before signing or 'author' chunk for bounce message
- Validate signatures before accepting bounces

USER SIGNET

A user signet is generated automatically by a user's client submitted using DMAP. The public signet is published on an authoritative DMTP server. Whether or not the user's private keys are shared with their organization's server depends on the account mode (trustful, cautious, and paranoid). In trustful mode, each device the user has can get access to keys through the organization's DMAP server. In paranoid mode, the user must use an independent mechanism when using multiple devices for synchronizing keys.

To minimize the amount of data exposed by a compromised private key, users are encouraged to have their signets rotated automatically. The time period recommended will likely vary by user, but could range from a handful of days to a period of weeks. Users who suspect their private keys have been compromised can trigger a manual signet rotation ahead of the scheduled rotation.

To provide a robust validation model, a potential sender has multiple avenues for confirming that a specific public key belongs to a user address. The primary basis is that a public key was retrieved from an organization's authoritative key store, and contains an organization signature that can be traced to a verifiable and trusted organizational signet. This constitutes basic authenticity and typically means the key can be trusted unless: the lookup request(s) was subverted or the organization is complicit in an attack (assuming the organization's key has not also been compromised). Additional verification paths are designed to allow detection of such attacks.

A verifiable chain of custody can illustrate that the owner of an address may have changed recently; this can be used by people with a previous trusted signet in their local cache. Finally, the use of the optional global ledger can provide a non-reputable external record of user signet publications that a client can consult independently of a provider and

thus detect when their provider might be complicit in an attack on their account. External sources also provide non-reputable evidence of a possible SP MitM attack.

A user signet, and its associated private keys, is used to:

- Decrypt inbound messages.
- Sign outbound messages.
- Sign new public signets before submitting them to the organization's server for publication.

DOMAIN NAME

The DNS system controls whether a domain supports DIME and provides the trusted anchor for organizational and user signets. In effect, compromising the DNS records would permit an attacker to gain authoritative control over a domain's identity. The primary long-term path for ensuring the validity DNS information and responses is DNSSEC.

TRANSMISSION CHANNEL

TLS is the primary means of protecting against wiretapping and the tampering of data in transit. For TLS to provide MitM protection a server certificate must be validated with an X.509 certificate signed by a certificate authority or against a TLS field provided by an MR signed using DNSSEC.

CLIENT

Client implementations will perform the user level cryptographic operations. Like email today, we anticipate a large variety of DIME client implementations will be created. They will likely range from thick applications that run on desktop and mobile devices, to thin clients written in JavaScript that are loaded from a web server at runtime. Because the user level cryptographic functions are performed by the client for cautious and paranoid users, it is important that these client implementations properly implement the cryptographic primitives and conform to the user interface and implementation standards supplied. These standards will ensure client implementations follow a baseline for the secure handling of sensitive information like passwords and private keys. Clients will also be responsible for communicating to users which inbound and outbound messages are protected by encryption because they involve DIME enabled domains, versus those that were sent naked using traditional mail protocols.

MAIL SERVER: MSA, MTA, MDA

Email content and data structure are protection by a proper DIME implementation; however, it is still the responsibility of the mail server organization to follow security best practices and secure the mail server.

KEY SERVER

Key management that provides redundant sources can aid in detection of compromised servers. Sources can be authoritative servers or be replicated through syndication to a partner domain's servers or in the future to the global ledger.

DNS SERVER

Distinct from using DNSSEC to authenticate DNS content, the responsibility for securing a domain's DNS servers remains with the organization.

Primary protection is accomplished by DNSSEC. However, if DNSSEC name validation cannot be used, it is still possible to reach a trusted state by publishing a DNS record AND using a TLS certificate that has been signed by a trusted Certificate Authority.

GATEWAYS

SMTP gateways provide the ability for DIME users to exchange messages with users at domains that do not support DIME. These gateways accept incoming SMTP messages from non-DIME domains and encrypt them using a user's current key before storing it on the server. Likewise, outbound messages can be relayed through a gateway to an SMTP host. It should be possible to translate, without any information loss, between the SMTP MIME format and the D/MIME message format. It is worth noting that organizations can choose to disable SMTP access at a domain level, or allow users to disable SMTP access at a user level. It is also important to understand that because SMTP messages may be transmitted in the clear in a worst-case scenario, and rely on TLS for protection in a best-case scenario, that users understand when they are sending out messages to a DIME enabled domain versus when they send naked messages via traditional email.

PERSISTENCE

For network level protection, DIME relies on TLS cipher suites that provide perfect forward secrecy. For message level protection, we assume that most users will want to retain persistent access to their historical message corpus. This implies retaining private keys to facilitate the future decryption of messages or alternatively, clients storing messages in their decrypted form locally before deleting a given private key.

HUMAN FACTORS

System security is often compromised through social engineering and other challenges with user and operator behavior. Simply implementing DIME does not replace good user education and competent operational security. Bad passwords, poor protection of private keys, and situational factors (such as leaving a laptop, no matter how short the length of time, unattended at the airport) cannot be mitigated by DIME. Depending on the implementation, examples of efforts to mitigate human factors include tailoring the user's interface, such as flagging information that is to be more or less trusted, and compensatory computation, as might be used to counteract a shorter password.

---

[13] In this context, this can be considered an estoppel (i.e. a revocation)

THIS PAGE INTENTIONALLY LEFT BLANK

PART 11: CREDITS

AUTHOR

LADAR LEVISON

Ladar Levison is the Founder of Lavabit, LLC, which served as a place for free and private email accounts. By August of 2013, Lavabit had grown to over 410,000 users. Levison created Lavabit because he believes that privacy is a fundamental, necessary right for a functioning, free and fair democratic society. On August 8, 2013, he made the bold decision to shut down his business after refusing to become "complicit in crimes against the American people." Presently, Levison is serving as the lead architect for the Dark Internet Mail Environment. Levison continues to vigorously advocate for free speech and the right to privacy, speaking at conferences, and collaborating on projects which are working to give back control of the Internet to the people.

CONTRIBUTORS

DAVE CROCKER

David H. Crocker is a principal with Brandenburg InternetWorking. He designs network-based applications businesses and distributed system architectures. His focus is on the creation of Internet-based businesses built on a solid foundation

of customer benefit and revenue potential.

Dave worked in the ARPA and NSF CSNet network research community during the 1970s and early 1980s, and led product development efforts at MCI and various Silicon Valley companies, into the 1990s. He then founded several startup companies, serving as CEO for one. Dave has developed and operated two national email services, designed two others, and was CEO of a community non-profit ISP. His senior management product efforts cover email clients and servers, core protocol stacks for TCP/IP and OSI, network management control stations, and knowledge management tools for product support. For his work on email, Dave was a co-recipient of the 2004 IEEE Internet Award.

Dave has been leading and authoring Internet standards for forty years, covering Internet mail, instant messaging, security, facsimile and EDI. He has also contributed to work on Internet commerce, domain name service, emergency services, and TCP/IP enhancements. He has authored more than 50 IETF Requests For Comments. Dave served as an Area Director for the Internet Engineering Task Force (IETF) variously overseeing network management, middleware and the IETF standards process. He has also been a member of the IETF's administrative and legal oversight bodies (IAOC/Trust).

UNNAMED CONTRIBUTORS

The DIME team would like to thank the gracious help of numerous, yet unnamed, contributors without whose dedication and time this publication would not be possible.

ATTRIBUTION

The document author's borrowed heavily from referenced RFCs and other sources for several sections. The team provides full attribution to the extent possible; however, if a reader notices an unintentional missing attribution, please notify Lavabit for correction. The DIME team owes a debt of gratitude to the hard work of the many Internet revolutionaries that got us to this point of publication.

PART 12: REFERENCES

[AES] National Institute of Standards and Technology, "Specification for the Advanced Encryption Standard (AES)", FIPS 197, November 2001.

[ASCII] Cerf, V., ASCII format for Network Interchange, RFC 20, October 1969.

[AVIAN] Waitzman, D., A Standard for the Transmission of IP Datagrams on Avian Carriers, RFC 1149, April 1990.

[DKIM] Allman, E., et. al., DomainKeys Identified Mail (DKIM) Signatures, RFC 4871, May 2007.

[DNS] Mockapetris, P., Domain Names - Concepts and Facilities, RFC 1034, November 1987.

[E521] Aranha, D., et al., A note on high-security general-purpose elliptic curves, 2013.

[ECDH] Blake-Wilson, S., et. al., Elliptic Curve Cryptography (ECC) Cipher Suites for Transport Layer Security (TLS), RFC 4492, May 2006.

[EdDSA] Bernstein, D., High-speed high-security signatures.

[GCM] Dworkin, M., Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC, SP 800-38D, November 2007.

[GZIP] Deutsch, P., GZIP file format specification version 4.3, RFC 1952, May 1996.

[IMA] Crocker, D., Internet Mail Architecture, RFC 5598, July 2009.
Email vocabulary in this document is drawn from RFC 5598.

[IMF] Resnick, P., Internet Message Format, RFC 5322, October 2008.

[IP] Postel, J., Internet Protocol, RFC 791, September 1981.

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

[LANGUAGE] Phillips, A. and M. Davis, Tags for Identifying Languages, BCP 47, RFC 5646, September 2009.

[IANA-LANG] IANA, Language Subtag Registry.

[MIME] Freed, N. and N. Borenstein, Multipurpose Internet Mail Extension (MIME) Part One: Format of Internet Message Bodies, RFC 2045, November 1996.

Freed, N. and N. Borenstein, Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types, RFC 2046, November 1996.

Moore, K., MIME (Multipurpose Internet Mail Extensions) Part Three: Message Header Extensions for Non-ASCII Text, RFC 2047, November 1996.

Freed, N. and J. Klensin, Multipurpose Internet Mail Extensions (MIME) Part Four: Registration Procedures, BCP 13, RFC 4289, December 2005.

Freed, N. and N. Borenstein, Multipurpose Internet Mail Extensions (MIME) Part Five: Conformance Criteria and Examples, RFC 2049, November 1996.

[OCSP] Myers, M. et. al., Online Certificate Status Protocol, RFC 2560, June 1999.

[PEM] Linn, J., Privacy Enhancement for Internet Electronic Mail: Part I: Message Encryption and Authentication Procedures, RFC 1421, February 1993.

[PGP] Callas, J. et. al., OpenPGP Message Format, RFC 4880, November 2007.

[PNG] Graphics, P. N., Specification Information technology—Computer graphics and image processing—Portable Network Graphics (PNG): Functional specification, ISO/IEC, 15948.

[SEC] Standards for Efficient Cryptography: SEC 2: Recommended Elliptic Curve Domain Paramters, Version 1, September 2000.

[SHS] National Institute of Standards and Technology, Secure Hash Standard, FIPS 180-2, August 2002.

[SMIME] Ramsdell, B., Turner, S., Secure/Multipurpose Internet Mail Extensions (S/MIME) Version 3.2 Message Specification, RFC 5751, January 2010.

[SPARROW] Sparrow, E., Secure Email.

[SMTP] Klensin, J., Simple Mail Transfer Protocol, RFC 5321, October 2008.

[TCP] Postel, J., Transmission Control Protocol, RFC 793, September 1981.

[TLS] Dierks, T., Rescorla, E., The Transport Layer Security (TLS) Protocol, Version 1.2, RFC 5246, October 2008.

[TLS-ECDHE] Rescorla, E., TLS Elliptic Curve Cipher Suites with SHA-256/384 and AES Galois Counter Mode (GCM), RFC 5289, August 2008.

[TLS-SNI] Blake-Wilson, S. et. al., Transport Layer Security (TLS) Extensions, RFC 3546, June 2003.