Network Working Group
Internet-Draft
Intended status: Informational
Expires: 6 February 2026

J. Harvey
B. Kaliski
A. Fregly
S. Sheth
Verisign Labs
5 August 2025

Considerations for Integrating Merkle Tree Ladder (MTL) Mode Signatures into Applications
draft-harvey-cfrg-mtl-mode-considerations-02

Abstract

This document provides design considerations for application designers on how to add Merkle Tree Ladder (MTL) Mode signatures into their applications. It provides general considerations relevant to any signature algorithm in addition to specific considerations for MTL mode such as for grouping and ordering messages to be signed, computing and signing ladders, and forming signatures exchanged between application components, including handling caching between the signer and the verifier.

Status of This Memo

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

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

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

This Internet-Draft will expire on 6 February 2026.

Copyright Notice

Table of Contents

1. Introduction

This document provides general design considerations for integrating Merkle Tree Ladder mode [I-D.harvey-cfrg-mtl-mode] signature algorithms into applications, including design decisions that would typically be made by the application. Documents specifying how to integrate MTL mode into particular applications, e.g., [I-D.fregly-dnsop-slh-dsa-mtl-dnssec] for DNSSEC, may provide more

detailed considerations specific to their applications. In the following, the term "application" refers to an application integrating MTL mode.

The design considerations in this document assume that the MTL mode signature algorithms follow the specification of MTL mode in [I-D.harvey-cfrg-mtl-mode], and in particular that the Merkle tree ladders to be signed are implemented using the "binary rung strategy" defined in Section 6.5 of [I-D.harvey-cfrg-mtl-mode].

The document is organized into two parts. Section 2 provides general considerations that apply to MTL mode but are not specific to MTL mode when integrating cryptographic schemes into an application. Section 3 providers considerations that relates to MTL mode specifically as, unlike general signature algorithms, MTL mode is not "drop-in" and impacts the logic and workflow for signing messages and verifying signatures.

As discussed in [MTL-MODE] MTL mode can help reduce the signature size impact in "message series signing" applications. In these applications, a signer continuously signs new messages and publishes the messages and their signatures. A verifier then continuously requests selected messages and verifies their signatures. For example, the signed messages could be web Public-Key Infrastructure certificates [RFC5280], Domain Name System Security Extensions (DNSSEC) records [RFC4033] or certificate timestamps [RFC9162].

Other potential applications include message signing, code signing, payments, blockchain transactions, identity management, and time-stamping. MTL mode can also help reduce signature computation impact in applications where signing operations for multiple signature operations can be batched, including the applications already mentioned as well as security protocols where one party may negotiate multiple security associations with other parties concurrently, e.g., TLS [RFC8446] (see also [I-D.ietf-tls-batch-signing]) or IKE v2 [RFC7296].

2. General Considerations

Integrating a signature algorithm into an application involves specifying algorithm identifiers, key formats, and signature formats within the context of the application. As would be the case if a conventional signature algorithm were being integrated, the general design considerations include:

1. Algorithm instantiations to be integrated.
2. Algorithm identifiers for each of these instantiations.
3. Key formats for public keys.

4. Key formats for private keys (optional).
5. Signature formats.

2.1. Algorithm Instantiations

MTL mode provides a collection of signature instantiations that combine MTL mode with the underlying SLH-DSA signature schemes (see [I-D.harvey-cfrg-mtl-mode] Section 10 - SLH-DSA-MTL Instantiation Table).

An application can select any number of the signature instantiations in the table.

2.2. Algorithm Identifiers

Algorithm identifiers indicate the algorithm associated with a public key, private key, and/or signature.

Depending on the application’s conventions, the algorithm identifier for a selected MTL mode instantiation can be defined by the application itself or can be drawn from a separate registry that supports multiple applications.

As one example of the latter, PKIX-based applications base their algorithm identifiers on object identifiers in the SMI Security for PKIX registry [IANA-SMI-NUMBERS] maintained by IANA (see also [RFC7299]). As another example, DNSSEC draws its algorithm identifiers from the DNSSEC Algorithm Numbers registry [IANA-DNSSEC-NUMBERS] also maintained by IANA. [I-D.fregly-dnsop-slh-dsa-mtl-dnssec] includes a request to add algorithm identifiers for MTL mode instantiations to this registry.

2.3. Specifying Key Formats for Public Keys

An application can use the public key byte format specified in this document, i.e., the SLH-DSA public key format [FIPS205-IPD] for interoperability with other implementations of MTL mode combined with SLH-DSA. An application can also use a different format if preferable for consistency with application conventions. An example of the latter choice for a conventional algorithm is given in [RFC7518], which uses object notation for representing RSA public keys when used in JSON Web Signatures and JSON Web Encryption, overriding the ASN.1 format in the referenced cryptographic specification [RFC3447]. An application can have multiple formats, such as the key format representation in DNS where there can be a wire format (e.g. key byte format) and a readable presentation format (e.g. Base64 encoded).

2.4. Key Formats for Private Keys (Optional)

An application can also specify the format of the private keys for each MTL instantiation it has selected, if needed for interoperability.

An application can use the private key byte format specified in this document, i.e., the SLH-DSA private key format [FIPS205-IPD] for interoperability with other implementations of MTL mode combined with SLH-DSA, or it can use a different format if preferable.

2.5. Specifying Signature Formats

MTL mode specifies two related but separate signature formats: full signatures, which can be verified directly using a public key (Section 9.4 of [I-D.harvey-cfrg-mtl-mode]), and condensed signatures, which can be verified using a public key and an associated signed ladder (Section 9.5 of [I-D.harvey-cfrg-mtl-mode]). There can also be a wire format (e.g. signature byte format) and a readable presentation format (e.g. Base64 encoded) for each signature format.

An application that supports full signatures can use the full signature byte format specified in Section 9.4 of [I-D.harvey-cfrg-mtl-mode].

An application that supports condensed signatures can either use the condensed signature byte format specified in Section 9.5 of [I-D.harvey-cfrg-mtl-mode] or a combination of that byte format and either a URI or another explicit ladder identifier (see Section 9.1 of [I-D.harvey-cfrg-mtl-mode] Step 9 and Section 3.5 below).

An application that supports both full and condensed signatures in a given exchange between application components can specify the signature format according to its interoperability objectives. As an example, [I-D.fregly-dnsop-slh-dsa-mtl-dnssec] Section 4 uses a one-octet MTL-Type prefix to indicate whether a signature is in condensed (0) or full (1) format.

An application can convey the fields of the signature format separately from one another if convenient. For example, in an application where the message and signature are sent in the same data stream, the randomizer field R_mtl of the signature can be sent before the message while the other fields of the signature format are sent after the message. This way, a verifier can start its processing of the message as soon as it receives the randomizer (see Section 5.2 of [I-D.harvey-cfrg-mtl-mode] where the data value is computed from the randomizer and the message). If the randomizer

were sent after the message, then the verifier may need to store the message until it receives the randomizer, which could be inconvenient if the message is very long. A signer can similarly start its processing of a message to be signed as soon as it has generated and sent the randomizer (except in the optional case where the randomizer depends on the message).

Additional considerations for providing signatures are given in Section 3.4.

3. MTL Mode-Specific Considerations

Conventional signature algorithms can be considered "drop-in" replacements because they only affect the general considerations in Section 2. MTL mode is not "drop-in," however, and has several additional considerations for applications that can impact the logic and workflow for signing messages and verifying signatures. MTL-mode-specific design considerations include:

1. Generating randomizers including "optional random" inputs.
2. Grouping and ordering messages to be signed.
3. Computing and signing Merkle tree ladders during signature generation.
4. Forming the actual signatures exchanged between application components.
5. Enabling application components to obtain signed ladders when needed.
6. Caching signed ladders.

MTL mode does not necessarily affect how the application forms the message input to the signature algorithm or where the application puts the signature output. An application could change these aspects for a specific signature algorithm, but it does not need to do so for MTL mode.

In DNSSEC, for example, the message can still be the wire format representation of an RRSIG record (minus the signature field) followed by the wire format representation of the canonical sorted resource record set (RRset) that is being signed. The resulting signature can still be conveyed in the RRSIG record’s Signature field which is not part of the message that is signed [RFC4034]. For PKIX certificates, the message can still be the ASN.1 DER encoding tbsCertificate field of the certificate, and the signature can still be conveyed in the certificate’s signatureValue field [RFC5280]. And for OCSP responses, the message can still be the DER encoding of the tbsResponseData field of the OCSP response, and the signature can still be conveyed in response’s signature field [RFC6960].

3.1. Generating Randomizers Including "Optional Random" Inputs

MTL mode gives implementations two ways of setting the "optional random" OptRand input to PRF_msg during the MTL mode operations. As discussed in Section 4.4 and Section 5.1 of [I-D.harvey-cfrg-mtl-mode], implementations can set OptRand either to the fixed value PK.seed or to a value that is randomly generated for each call to OptRand.

Similarly, the underlying signature scheme, SLH-DSA, gives implementations two ways of setting the opt_rand input to PRF_msg during SLH-DSA signature generation operations. As discussed in Section 9.2 and Algorithm 18 of [FIPS205-IPD], they can likewise set opt_rand either to the fixed value PK.seed or to a value that is randomly generated for each call. The decisions on the optional random inputs are independent of one another.

An application can set OptRand and opt_rand according to its own security and operational objectives. In addition, an application integrating MTL mode can use the same or a different SK.prf value for the MTL mode operations as for the underlying signature scheme, and can optionally include the message M in the input to the PRF_msg call.

3.2. Grouping and Ordering Messages to Be Signed

In MTL mode, messages to be signed with a given private key are grouped into one or more ordered message series each associated with a separate series identifier and evolving Merkle node set (Section 3 of [I-D.harvey-cfrg-mtl-mode]). Messages in a message series are assigned sequential 4-byte indexes, starting with 0, corresponding to the order in which the messages are added to the series and their associated leaf nodes are added to the node set.

An application can group messages to be signed into one or more message series and order the messages within each series according to its own operational objectives.

As one approach, all messages to be signed with a given private key can be grouped into a single message series and ordered according to when they are ready to be signed. This approach is similar to how typical stateful hash-based signature schemes process messages, as they also have only a single "series" per private key and process messages sequentially.

Examples of other approaches for this purpose include:

1. Grouping messages into multiple series each with a limited number of messages that is less than MTL mode’s 2^32 maximum. This approach has the advantage that the maximum authentication path length (and hence the condensed signature size) is a function of the maximum size of the Merkle tree. Limiting the maximum authentication path length can help the application stay within signature length constraints.
2. Grouping messages into multiple series based on when their signatures expire. Messages whose signature expires at one point in time can be organized into one group, while messages whose signatures expire at another time - which can include "refreshes" of messages in the first group - can be organized into another group. This approach has the advantage that the two groups can be evolved in parallel.

An application can group and order messages to be signed in MTL mode either the same way or a different way than the application organizes messages for other purposes. For instance, DNS defines a canonical ordering of domain names in a zone [RFC4034] and of RRsets per domain name [RFC8976]. However, RRsets can be signed with MTL mode in a different order than the canonical order.

3.3. Computing and Signing Ladders During Signature Generation

In MTL mode, ladders are computed and signed at certain points during the signature generation process for a message series (see Section 9.1 of [I-D.harvey-cfrg-mtl-mode] steps 5 & 6).

Examples of other approaches for this purpose include:

1. Computing a new ladder and signing it after a certain number of messages have been added to a message series.
2. Computing a new ladder and signing it after a certain time window (assuming at least one message has been added).
3. Computing a new ladder and signing it after a defined application event has occurred.

Computing and signing ladders occasionally , i.e., after a batch of messages, rather than after each and every message, can reduce the number of signing operations performed with the underlying signature algorithm and thus reduces the average computational overhead per message signed. This practice can also reduce the load on a hardware security module.

Examples of batch signing include: for DNSSEC, signing multiple RRsets being added to a zone, updated, and/or having their signatures refreshed at the same time; for PKIX certificates, signing multiple certificates being created and/or renewed at the same time; for OCSP,

signing multiple responses at the same time, including responses for multiple certificates; and for Certificate Transparency, signing time-stamps for multiple certificates (or pre-certificates) at the same time.

Other examples include signing the transcripts of multiple TLS handshakes in a batch, or signing multiple IKE v2 exchanges as a batch. In these examples, the computational impact of the underlying signature scheme at the signer can be reduced but not necessarily the size impact (because a given verifier would typically process signatures in succession and therefore would need a new signed ladder each time).

In addition, signatures on messages to be published in future time periods can also be pre-signed as part of a batch and held for later publication (e.g., future signed keys, certificates, status responses, etc., assuming that the content of the messages is known in advance). In this case, the signed ladder for the full batch can be published after the batch is signed, while the condensed signatures on the individual messages can be released gradually at their relevant time periods approach. The verifier can then obtain the signed ladder early and use it to verify the condensed signatures that are released later.

While messages can be signed (and pre-signed) in batches using other signature algorithms, MTL mode batch processing can be particularly efficient because the underlying signature scheme is applied only once per batch. Batch signing with MTL mode or using other signature algorithms needs to be considered carefully to ensure that new signatures are available in a timely manner while still gaining the benefits of batch signing.

The security analysis of SLH-DSA assumes that a given private key is used no more than 2^64 times. A proposal has been made to add new SLH-DSA instantiations that are more efficient assuming a lower limit on the number of uses (see [FD24] for analysis of the limits 2^10, 2^20, 2^30, 2^40, and 2^50). Signing messages in batches can help an application stay within these smaller limits.

3.4. Forming Signatures Exchanged Between Application Components

With a conventional signature scheme, the signature exchanged between application components when authenticating a message is the same as the initial signature that was generated on the message.

With MTL mode, in contrast, the signature that is exchanged for a given message can evolve over time. In particular, the signature exchanged between application components can include an authentication path relative to a ladder that was computed after the message was initially processed, i.e., added to the Merkle node set.

In addition, the MTL mode signature exchanged between application components can be either a condensed signature or a full signature.

There are three main cases to consider, described next.

In the following, it is assumed that the condensed or full signature is associated with a message that can also be sent from one component to another. Associating messages with signatures is part of the application logic and is independent of MTL mode. MTL mode only affects how the signature associated with the message is formed, including which message series it belongs to and its index within the message series.

3.4.1. Signer

A MTL mode signer has access to the evolving message series, the evolving Merkle tree, and all the signed ladders it has computed. The signer can therefore form condensed or full signatures for a message that include an authentication path relative to any subsequent ladder, i.e., any ladder for the same message series that was computed at or after the time the message was initially added to the Merkle node set. When the signer provides a condensed signature, the signer can include an authentication path relative to a subsequent signed ladder that is as current as possible. Doing so helps increase the likelihood that a verifier will be able to verify the signature with a ladder the verifier already holds.

When the signer provides a full signature, the signer can include a subsequent signed ladder that is as current as possible and an authentication path that can be verified relative to the signed ladder. Doing so helps increase the likelihood that a verifier who caches the signed ladder will be able to verify future signatures it receives with the ladder.

The signer in an application can be implemented as part of a provisioning system and/or with a cryptographic module that stores the private key for the underlying signature scheme.

3.4.2. Component providing signatures on behalf of a signer.

In some applications, a component acting on behalf of the signer forwards the signer's signatures to other components. As above, when such a component provides a condensed signature, it can include an authentication path relative to as current a subsequent signed ladder as possible, and when it provides a full signature, it can include as current a subsequent signed ladder as possible.

Because the component acts on behalf of the signer, the signer can provision the component to provide current authentication paths and signed ladders. Example approaches for this purpose include:

1. Provisioning the component with the per-message randomizers and the signed ladder for a batch of new messages when a batch is initially added to the message series and a new signed ladder is produced. The component can then compute the evolving Merkle node set from this information and form condensed and full signatures the same way that the signer can.
2. Provisioning the component with the per-message randomizers, the leaf nodes and the signed ladders. The component can then compute the rest of the evolving Merkle node set from the leaf nodes, without re-hashing the messages, and proceed as above.
3. Provisioning the component with initial versions of signatures on a batch of new messages when the batch is initially added to a message series, and updating previous signatures as needed when the message series evolves, so that their authentication paths remain as current as possible.

(Note that in all three approaches, the signer provisions only publicly available or computable information to the component. In particular, the signer does not provision the component with its private key.)

An advantage of the first and second approaches is their efficiency: the component only stores the randomizers, the Merkle node set and the signed ladders, not necessarily every condensed or full signature. The signer provisions new randomizers and a new signed ladder to the component when a new batch of messages is added to the message series and the resulting new ladder is computed and signed, again not necessarily a signature on every message.

An advantage of the third approach is its simplicity. It does not involve changes to the protocol interaction between the signer and the component, nor new logic at the component, assuming that the signer already has a way to provision the component with new signatures. Indeed, if the component only supports condensed signatures, then the component does not even need to be "MTL mode aware," as it can simply provide its most current version of a condensed signature to other components.

An example of a component providing signatures on behalf of a signer in DNSSEC would be an authoritative name server, which is provisioned by a DNS zone administrator to provide copies of DNSSEC signatures that may have been generated elsewhere by the zone administrator. For PKIX certificates, an example would be a certificate directory managed by a certification authority that maintains copies of certificates issued by the CA. For OCSP, a content distribution network could act in this role. In each of these examples, the component could potentially implement one or more of the approaches described above for providing MTL mode signatures to other components.

3.4.3. Component providing signatures but not on behalf of a signer

In some applications, a component not acting on behalf of the signer provides the signer’s signatures to other components.

Again, when such a component provides a condensed signature, it can include an authentication path relative to as current a subsequent signed ladder as possible. When it provides a full signature, it can include as current a subsequent signed ladder as possible.

Because the component does not act on behalf of the signer, the signer cannot necessarily apply the provisioning approaches described in Section 3.4.2. However, the component itself can potentially "refresh" its copies of signatures periodically to keep their authentication paths relative to as current a subsequent signed ladder as possible. Example approaches for this purpose include:

1. Requesting an updated copy of a signature on a message periodically, e.g., based on a "time-to-live" or similar value, or according to some other schedule, thus potentially obtaining a signature that includes a newer authentication path.

2. Requesting an updated copy of a signature when encountering a newer signed ladder for the same message series where the stored signature's authentication path is not compatible with the newer ladder (i.e., the authentication path doesn't reach the ladder). The component can request a new version of a signature when the incompatibility is encountered; when it needs to provide the signature to another component; or according to some other schedule.

An advantage of the first approach is its predictability: the refresh schedule is not affected by which other signatures are encountered. This approach can be a good fit for components that do not expect to encounter many other signatures, for instance a web server that caches signatures on a relatively small subset of a message series (e.g., signatures on the certificates and/or OCSP responses for the websites hosted by the web server).

An advantage of the second approach is its precision: the refresh schedule adapts to updates to the message series. This approach can be a good fit for components that do expect to encounter many other signatures, for instance a recursive name server that caches signatures on a relatively large subset of a message series (e.g., on the DNS records requested by recursive name server's clients).

(The foregoing assumes the application is like DNSSEC where components can repeatedly request a potentially updated copy of a previously signed message. Note that such applications are also more likely to be the ones with components that provide the signer's signature but do not act on behalf of the signer, and thus to which the present case applies.)

An example of a component providing signatures but not on behalf of a signer in DNSSEC is a recursive name server. Such a server serves as a cache: it provides copies of DNSSEC signatures generated elsewhere, but does so by looking up these signatures itself rather than by being provisioned by a DNS zone administrator. Other examples include a web proxy or cache, a content distribution server, or a web hosting server interacting with other application components on behalf of a client. In each case the component is not necessarily being managed by the signer.

3.5. Enabling Application Components to Obtain Signed Ladders

Example approaches for application components obtaining ladders when needed include:

1. A service that takes as input a public key identifier, a series identifier, and the target index pair of the authentication path within a condensed signature, and returns as output an associated signed ladder. (This approach can be called an "implicit identifier" approach.)
2. A service that responds to a specified URI or other explicit ladder identifier by returning a signed ladder. (This approach can be called an "explicit identifier" approach.)
3. An option in the protocol interaction between application components whereby a component can request a full signature instead of a condensed signature. The signed ladder would be obtained along with the condensed signature as part of the full signature. (This approach can thus be called a "request / retry" or "flag" approach.)

In the implicit identifier approach, the ladder identifier is derived from information already available to the application components including the public key and the condensed signature. Information about the service itself, such as a URI for the service, can be configured into the application independently of a specific condensed signature or may be determined by the application at runtime.

In the explicit identifier approach, the ladder identifier is conveyed together with a condensed signature.

In the flag approach, the identifier can be either implicit or explicit. However, the identifier may not necessarily be provided when requesting a signed ladder. Rather, the application component can retry the same protocol exchange that initially returned the condensed signature to obtain a corresponding full signature instead.

The implicit or explicit identifier approach can use the signed ladder byte format defined in Section 9.6 of [I-D.harvey-cfrg-mtl-mode] to represent the signed ladder.

The explicit identifier approach can use a URI or other explicit identifier according to the application’s interoperability objectives. For example, following Information-Centric Networking concepts [RFC8793], the URI or other identifier can include an application-defined name (possibly a combination of one or more of the input fields listed in the first approach - public key identifier, series identifier, target index pair) and/or an implicit name (i.e., a cryptographic hash of the signed ladder). The URI or other identifier could also include a decentralized identifier ([DID]) or a domain name.

Note: Identifying a signed ladder by its cryptographic hash makes the signed ladder lookup implicitly verifiable. The property can be beneficial in applications where a time-stamp is applied to the combination of a condensed signature and a signed ladder identifier, rather than to a full signature - see Section 5, Security Considerations.

In the flag approach, the application can add a flag to the protocol interaction that indicates which signature format to return. [I-D.fregly-dnsop-slh-dsa-mtl-dnssec] Section 6 uses the mtl-mode-full EDNS(0) option to request that signatures be returned in full format. Otherwise, signatures are returned in condensed format. An extension could similarly be added to the TLS handshake [RFC8446] to request a full signature instead of condensed signature for all signed items returned in the handshake, and/or for particular items. Similarly, a flag could be added to a OCSP request to specify full vs. condensed signatures.

Note that a request for a full signature can be optimized by the following variation when an exchange includes multiple signatures (compare Section 6.2 of [I-D.fregly-dnsop-slh-dsa-mtl-dnssec]). Rather than returning all the signatures in full format, the response can include only enough signatures in full format to ensure that every other signature in the response can be verified. This optimization reduces the size impact while still ensuring that all the signatures returned can be verified without needing to look up any additional signed ladders.

An advantage of the implicit and explicit identifier approaches is that they do not involve changes to the protocol interaction between application components, such as a flag. Instead, the signed ladder lookup can be done "out of band." The approaches are also well suited to "store-and-forward" operations where messages (and signatures) are passed from one component to the next. However, these approaches involve an additional service that maintains signed ladders associated with identifiers for as long as the signed ladders may need to be looked up.

An advantage of the flag approach is that the interactions are all "in-band," thus avoiding additional services that need to be operated and kept coordinated with other application components. Indeed, in the flag approach, the component sending a signature knows that the component receiving it will be able to verify it without reference to other services. However, the flag approach involves protocol changes.

Another alternative to the approaches just described is a service that returns the current signed ladder associated with a given public key and series identifier. Such a service can maintain only the current signed ladder. However, because condensed signatures are formed relative to a specific signed ladder, it is possible that an older condensed signature (perhaps one that has been cached) may not be able to be verified relative to the current signed ladder. (A related observation is made in Section 3.4.3 about "refreshing" authentication paths.) Accordingly, this document sets forth the three approaches already described.

3.6. Caching Signed Ladders

In Steps 5 and 6 in Section 3 of [I-D.harvey-cfrg-mtl-mode], when a verifier receives an authentication path, if the verifier has a ladder that is compatible with the authentication path, then the verifier verifies the authentication path relative to the ladder. If not, the verifier requests a new signed ladder that the authentication path is compatible with. After verifying the signature on the ladder, the verifier can then store the ladder for future use. The more ladders the verifier maintains, the more likely the verifier will be able to verify a received condensed signature relative to a ladder the verifier already holds. (Although Section 3.4 describes providing as current an authentication path as possible, if a component doesn’t do so, then a verifier that has stored previously verified ladders can potentially still verify less-current authentication paths relative to one of its older ladders.)

An application’s design considerations on when and for how long to store verified ladders for future use, can include how much storage the verifier has available and how messages and grouped and ordered (see Section 3.2). For instance, if messages are grouped into multiple message series for a given public key, then it can be more beneficial to maintain at least one ladder per message series than to maintain additional ladders for the same message series.

4. IANA Considerations

This document makes no requests of IANA.

5. Security Considerations

[I-D.harvey-cfrg-mtl-mode] provides various security considerations for MTL mode that may also be applicable to applications that integrate MTL mode.

Applications that use a time-stamping service should consider how the approach for identifying signed ladders (see Section 3.5) can affect security. In particular, a time-stamp on a combination of a condensed signature and a URI or other identifier may demonstrate that the combination existed in a particular timeframe, but not necessarily that the identifier was associated with a specific signed ladder at that time (the actual signed ladder may have been different, or may not have existed). As a result, such a time-stamp on its own may not be sufficient to demonstrate that a valid signature existed at the specified time. This concern can be addressed by separately time-stamping an association between the identifier and the actual signed ladder. It can also be addressed by including a cryptographic hash of the actual signed ladder as part of the identifier, as the hash will uniquely identify the actual signed ladder from a cryptographic perspective. Alternatively, it can be addressed by including the actual signed ladder within the input to the time-stamp's signature computation, even though only the signed ladder's identifier is conveyed as part of the timestamp. In effect, the signed ladder would be "incorporated by reference"; both signer and verifier would need to incorporate it. For all three approaches, the application may need to maintain the signed ladder associated with a given identifier for as long as the time-stamped signature needs to be verified (i.e., even after the signer's key pair is no longer in use).

Pre-signing messages (see Section 3.3) carries the risk that the messages that have been pre-signed are no longer accurate at the time that their signature is to be published (e.g., a DNS record has subsequently changed or a certificate has been revoked). An application that uses pre-signing can manage precomputed signatures so that they are not released until the time set by the application's policy (and that a signature is not necessarily released at all if its associated message is no longer accurate).

As discussed in the security considerations of [I-D.harvey-cfrg-mtl-mode], due to MTL mode's randomizers, "individual messages that have been signed by the signer remain private until the signer publishes them." That privacy protection applies to the extent that an adversary does not already have other information about the messages. In particular, if a message is pre-signed as contemplated in Section 3.3, then an adversary may be able to predict the message based on other known messages (e.g., they may differ only in their validity periods). Nevertheless, even if an adversary can predict a message before it is published, the adversary cannot provide a valid MTL mode signature on the message (without access to the signer's private key) because a valid signature includes a message-specific randomizer, and the randomizer cannot be computed without the PRF key.

6. References

6.1. Normative References

[FIPS205-IPD]
National Institute of Standards and Technology (NIST), "Stateless Hash-Based Digital Signature Standard", FIPS PUB 205 (Initial Public Draft), DOI 10.6028/NIST.FIPS.205, 24 August 2023, <https://doi.org/10.6028/NIST.FIPS.205.ipd>.

[I-D.harvey-cfrg-mtl-mode]
Harvey, J., Kaliski, B., Fregly, A., and S. Sheth, "Merkle Tree Ladder (MTL) Mode Signatures", Work in Progress, Internet-Draft, draft-harvey-cfrg-mtl-mode-06, 15 March 2025, <https://datatracker.ietf.org/doc/html/draft-harvey-cfrg-mtl-mode-06>.

6.2. Informative References

[DID] Sporny, M., Longley, D., Sabadello, M., Reed, D., Steele, O., and C. Allen, "Decentralized Identifiers (DIDs) v1.0", W3C did-core, July 2022, <https://www.w3.org/TR/2022/REC-did-core-20220719/>.

[FD24] Fluhrer, S. and Q. Dang, "Smaller Sphincs+", Cryptology ePrint Archive Paper 2024/018, 2024, <https://eprint.iacr.org/2024/018>.

[I-D.fregly-dnsop-slh-dsa-mtl-dnssec]
Fregly, A., Harvey, J., Kaliski, B., and D. Wessels, "Stateless Hash-Based Signatures in Merkle Tree Ladder Mode (SLH-DSA-MTL) for DNSSEC", Work in Progress, Internet-Draft, draft-fregly-dnsop-slh-dsa-mtl-dnssec-04, 2 April 2025, <https://datatracker.ietf.org/doc/html/draft-fregly-dnsop-slh-dsa-mtl-dnssec-04>.

[I-D.ietf-tls-batch-signing]
Benjamin, D., "Batch Signing for TLS", Work in Progress, Internet-Draft, draft-ietf-tls-batch-signing-00, 13 January 2020, <https://datatracker.ietf.org/doc/html/draft-ietf-tls-batch-signing-00>.

[IANA-DNSSEC-NUMBERS]
Internet Assigned Numbers Authority (IANA), "DNS Security Algorithm Numbers", <https://www.iana.org/assignments/dns-sec-alg-numbers/dns-sec-alg-numbers.xhtml>.

[IANA-SMI-NUMBERS]
Internet Assigned Numbers Authority (IANA), "SMI Security for PKIX", <https://www.iana.org/assignments/smi-numbers/smi-numbers.xhtml#smi-numbers-1.3.6.1.5.5.7>.

[MTL-MODE] Fregly, A., Harvey, J., Kaliski, B., and S. Sheth, "Merkle Tree Ladder Mode: Reducing the Size Impact of NIST PQC Signature Algorithms in Practice", in Rosulek, M. (editor), Lecture Notes in Computer Science VOLUME 13871, CT-RSA 2023 - Cryptographers Track at the RSA Conference pages 415-441, DOI 10.1007/978-3-031-30872-7_16, 2023, <https://eprint.iacr.org/2022/1730.pdf>.

[RFC3447] Jonsson, J. and B. Kaliski, "Public-Key Cryptography Standards (PKCS) #1: RSA Cryptography Specifications Version 2.1", RFC 3447, DOI 10.17487/RFC3447, February 2003, <https://www.rfc-editor.org/info/rfc3447>.

[RFC4033] Arends, R., Austein, R., Larson, M., Massey, D., and S. Rose, "DNS Security Introduction and Requirements", RFC 4033, DOI 10.17487/RFC4033, March 2005, <https://www.rfc-editor.org/info/rfc4033>.

[RFC4034] Arends, R., Austein, R., Larson, M., Massey, D., and S. Rose, "Resource Records for the DNS Security Extensions", RFC 4034, DOI 10.17487/RFC4034, March 2005, <https://www.rfc-editor.org/info/rfc4034>.

[RFC5280] Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., and W. Polk, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 5280, DOI 10.17487/RFC5280, May 2008, <https://www.rfc-editor.org/info/rfc5280>.

[RFC6960] Santesson, S., Myers, M., Ankney, R., Malpani, A., Galperin, S., and C. Adams, "X.509 Internet Public Key Infrastructure Online Certificate Status Protocol - OCSP", RFC 6960, DOI 10.17487/RFC6960, June 2013, <https://www.rfc-editor.org/info/rfc6960>.

[RFC7296] Kaufman, C., Hoffman, P., Nir, Y., Eronen, P., and T. Kivinen, "Internet Key Exchange Protocol Version 2 (IKEv2)", STD 79, RFC 7296, DOI 10.17487/RFC7296, October 2014, <https://www.rfc-editor.org/info/rfc7296>.

[RFC7299] Housley, R., "Object Identifier Registry for the PKIX Working Group", RFC 7299, DOI 10.17487/RFC7299, July 2014, <https://www.rfc-editor.org/info/rfc7299>.

[RFC7518] Jones, M., "JSON Web Algorithms (JWA)", RFC 7518, DOI 10.17487/RFC7518, May 2015, <https://www.rfc-editor.org/info/rfc7518>.

[RFC8446] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018, <https://www.rfc-editor.org/info/rfc8446>.

[RFC8793] Wissingh, B., Wood, C., Afanasyev, A., Zhang, L., Oran, D., and C. Tschudin, "Information-Centric Networking (ICN): Content-Centric Networking (CCNx) and Named Data Networking (NDN) Terminology", RFC 8793, DOI 10.17487/RFC8793, June 2020, <https://www.rfc-editor.org/info/rfc8793>.

[RFC8976] Wessels, D., Barber, P., Weinberg, M., Kumari, W., and W. Hardaker, "Message Digest for DNS Zones", RFC 8976, DOI 10.17487/RFC8976, February 2021, <https://www.rfc-editor.org/info/rfc8976>.

[RFC9162] Laurie, B., Messeri, E., and R. Stradling, "Certificate Transparency Version 2.0", RFC 9162, DOI 10.17487/RFC9162, December 2021, <https://www.rfc-editor.org/info/rfc9162>.

Appendix A. Change Log

00: Initial draft of the document.

Acknowledgements

The authors would like to acknowledge the following individuals for their contributions to this document: Ondrej Sury for emphasizing the importance of protecting against malicious signers and intermediaries.

Authors' Addresses

J. Harvey
Verisign Labs
12061 Bluemont Way
Reston, VA 20190
United States of America
Email: jsharvey@verisign.com
URI: https://www.verisignlabs.com/

B. Kaliski
Verisign Labs
12061 Bluemont Way
Reston, VA 20190
United States of America
Email: bkaliski@verisign.com
URI: https://www.verisignlabs.com/

A. Fregly
Verisign Labs
12061 Bluemont Way
Reston, VA 20190
United States of America
Email: afregly@verisign.com
URI: https://www.verisignlabs.com/

S. Sheth
Verisign Labs
12061 Bluemont Way
Reston, VA 20190
United States of America
Email: ssheth@verisign.com
URI: https://www.verisignlabs.com/