Abstract

This document specifies the conventions for using the Hierarchical Signature System (HSS) / Leighton-Micali Signature (LMS) hash-based signature algorithm with the CBOR Object Signing and Encryption (COSE) syntax. The HSS/LMS algorithm is one form of hash-based digital signature; it is described in RFC 8554.

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 June 13, 2020.

Copyright Notice

Copyright (c) 2019 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust’s Legal Provisions Relating to IETF Documents (https://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of
the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

1. Introduction

This document specifies the conventions for using the Hierarchical Signature System (HSS) / Leighton-Micali Signature (LMS) hash-based signature algorithm with with the CBOR Object Signing and Encryption (COSE) [RFC8152] syntax. The LMS system provides a one-time digital signature that is a variant of Merkle Tree Signatures (MTS). The HSS is built on top of the LMS system to efficiently scale for a larger numbers of signatures. The HSS/LMS algorithm is one form of hash-based digital signature, and it is described in [HASHSIG]. The HSS/LMS signature algorithm can only be used for a fixed number of signing operations. The number of signing operations depends upon the size of the tree. The HSS/LMS signature algorithm uses small public keys, and it has low computational cost; however, the signatures are quite large. The HSS/LMS private key can be very small when the signer is willing to perform additional computation at signing time; alternatively, the private key can consume additional memory and provide a faster signing time. The HSS/LMS signatures [HASHSIG] are currently defined to use exclusively SHA-256 [SHS].
1.1. Motivation

Recent advances in cryptanalysis [BH2013] and progress in the development of quantum computers [NAS2019] pose a threat to widely deployed digital signature algorithms. As a result, there is a need to prepare for a day that cryptosystems such as RSA and DSA that depend on discrete logarithm and factoring cannot be depended upon.

If large-scale quantum computers are ever built, these computers will have more than a trivial number of quantum bits (qubits) and they will be able to break many of the public-key cryptosystems currently in use. A post-quantum cryptosystem [PQC] is a system that is secure against such large-scale quantum computers. It is open to conjecture when it will be feasible to build such computers; however, RSA [RFC8017], DSA [DSS], ECDSA [DSS], and EdDSA [RFC8032] are all vulnerable if large-scale quantum computers come to pass.

Since the HSS/LMS signature algorithm does not depend on the difficulty of discrete logarithm or factoring, the HSS/LMS signature algorithm is considered to be post-quantum secure. The use of HSS/LMS hash-based signatures to protect software update distribution will allow the deployment of future software that implements new cryptosystems. By deploying HSS/LMS today, authentication and integrity protection of the future software can be provided, even if advances break current digital signature mechanisms.

1.2. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

2. LMS Digital Signature Algorithm Overview

This specification makes use of the hash-based signature algorithm specified in [HASHSIG], which is the Leighton and Micali adaptation [LM] of the original Lamport-Diffie-Winternitz-Merkle one-time signature system [M1979][M1987][M1989a][M1989b].

The hash-based signature algorithm has three major components:
As implied by the name, the hash-based signature algorithm depends on a collision-resistant hash function. The hash-based signature algorithm specified in [HASHSIG] currently makes use of the SHA-256 one-way hash function [SHS], but it also establishes an IANA registry to permit the registration of additional one-way hash functions in the future.

2.1. Hierarchical Signature System (HSS)

The hash-based signature algorithm specified in [HASHSIG] uses a hierarchy of trees. The Hierarchical N-time Signature System (HSS) allows subordinate trees to be generated when needed by the signer. Otherwise, generation of the entire tree might take weeks or longer.

An HSS signature as specified in [HASHSIG] carries the number of signed public keys (Nspk), followed by that number of signed public keys, followed by the LMS signature as described in Section 2.2. The public key for the top-most LMS tree is the public key of the HSS system. The LMS private key in the parent tree signs the LMS public key in the child tree, and the LMS private key in the bottom-most tree signs the actual message. The signature over the public key and the signature over the actual message are LMS signatures as described in Section 2.2.

The elements of the HSS signature value for a stand-alone tree (a top tree with no children) can be summarized as:

```
u32str(0) ||
lms_signature  /* signature of message */
```

where, the notation comes from [HASHSIG].

The elements of the HSS signature value for a tree with Nspk signed public keys can be summarized as:
u32str(Nspk) ||
signed_public_key[0] ||
signed_public_key[1] ||
...
signed_public_key[Nspk-2] ||
signed_public_key[Nspk-1] ||
lms_signature /* signature of message */

where, as defined in Section 3.3 of [HASHSIG], a signed_public_key is the lms_signature over the public key followed by the public key itself. Note that Nspk is the number of levels in the hierarchy of trees minus 1.

2.2. Leighton-Micali Signature (LMS)

Subordinate LMS trees are placed in the hierarchy structure discussed in Section 2.1. Each tree in the hash-based signature algorithm specified in [HASHSIG] uses the Leighton-Micali Signature (LMS) system. LMS systems have two parameters. The first parameter is the height of the tree, h, which is the number of levels in the tree minus one. The [HASHSIG] includes support for five values of this parameter: h=5; h=10; h=15; h=20; and h=25. Note that there are 2^h leaves in the tree. The second parameter is the number of bytes output by the hash function, m, which is the amount of data associated with each node in the tree. The [HASHSIG] specification supports only SHA-256, with m=32. An IANA registry is defined so that other hash functions could be used in the future.

The [HASHSIG] specification supports five tree sizes:

- LMS_SHA256_M32_H5;
- LMS_SHA256_M32_H10;
- LMS_SHA256_M32_H15;
- LMS_SHA256_M32_H20; and
- LMS_SHA256_M32_H25.

The [HASHSIG] specification establishes an IANA registry to permit the registration of additional hash functions and additional tree sizes in the future.

The [HASHSIG] specification defines the value I as the private key identifier, and the same I value is used for all computations with the same LMS tree. The value I is also available in the public key. In addition, the [HASHSIG] specification defines the value T[i] as the m-byte string associated with the ith node in the LMS tree, where and the nodes are indexed from 1 to 2^(h+1)−1. Thus, T[1] is the m-byte string associated with the root of the LMS tree.
The LMS public key can be summarized as:

\[ \text{u32str(lms\_algorithm\_type)} \ || \ \text{u32str(ots\_type)} \ || \ I \ || \ T[1] \]

As specified in [HASHSIG], the LMS signature consists of four elements: the number of the leaf associated with the LM-OTS signature, an LM-OTS signature as described in Section 2.3, a typecode indicating the particular LMS algorithm, and an array of values that is associated with the path through the tree from the leaf associated with the LM-OTS signature to the root. The array of values contains the siblings of the nodes on the path from the leaf to the root but does not contain the nodes on the path itself. The array for a tree with height \( h \) will have \( h \) values. The first value is the sibling of the leaf, the next value is the sibling of the parent of the leaf, and so on up the path to the root.

The four elements of the LMS signature value can be summarized as:

\[ \text{u32str(q)} \ || \ \text{ots\_signature} \ || \ \text{u32str(type)} \ || \ \text{path}[0] \ || \ \text{path}[1] \ || \ ... \ || \ \text{path}[h-1] \]

### 2.3. Leighton-Micali One-time Signature Algorithm (LM-OTS)

The hash-based signature algorithm depends on a one-time signature method. This specification makes use of the Leighton-Micali One-time Signature Algorithm (LM-OTS) [HASHSIG]. An LM-OTS has five parameters:

- \( n \) - The number of bytes output by the hash function. For SHA-256 [SHS], \( n=32 \).
- \( H \) - A preimage-resistant hash function that accepts byte strings of any length, and returns an \( n \)-byte string.
- \( w \) - The width in bits of the Winternitz coefficients. [HASHSIG] supports four values for this parameter: \( w=1; w=2; w=4; \) and \( w=8 \).
- \( p \) - The number of \( n \)-byte string elements that make up the LM-OTS signature.
- \( \text{ls} \) - The number of left-shift bits used in the checksum function, which is defined in Section 4.5 of [HASHSIG].

The values of \( p \) and \( \text{ls} \) are dependent on the choices of the parameters \( n \) and \( w \), as described in Appendix B of [HASHSIG].
The [HASHSIG] specification supports four LM-OTS variants:

- LMOTS_SHA256_N32_W1;
- LMOTS_SHA256_N32_W2;
- LMOTS_SHA256_N32_W4; and
- LMOTS_SHA256_N32_W8.

The [HASHSIG] specification establishes an IANA registry to permit the registration of additional hash functions and additional parameter sets in the future.

Signing involves the generation of C, which is an n-byte random value.

The LM-OTS signature value can be summarized as the identifier of the LM-OTS variant, the random value, and a sequence of hash values (y[0] through y[p-1]) as described in Section 4.5 of [HASHSIG]:

\[ u32str(otstype) || C || y[0] || ... || y[p-1] \]

3. Hash-based Signature Algorithm Identifiers

The CBOR Object Signing and Encryption (COSE) [RFC8152] supports two signature algorithm schemes. This specification makes use of the signature with appendix scheme for hash-based signatures.

The signature value is a large byte string as described in Section 2. The byte string is designed for easy parsing. The HSS, LMS, and LMOTS components of the signature value format include counters and type codes that indirectly provide all of the information that is needed to parse the byte string during signature validation.

When using a COSE key for this algorithm, the following checks are made:
- The ‘kty’ field MUST be ‘HSS-LMS’.
- If the ‘alg’ field is present, it MUST be ‘HSS-LMS’.
- If the ‘key_ops’ field is present, it MUST include ‘sign’ when creating a hash-based signature.
- If the ‘key_ops’ field is present, it MUST include ‘verify’ when verifying a hash-based signature.
- If the ‘kid’ field is present, it MAY be used to identify the top of the HSS tree. In [HASHSIG], this identifier is called ‘I’, and it is the 16-byte identifier of the LMS public key for the tree.

4. Security Considerations

The Security considerations from [RFC8152] and [HASHSIG] are relevant to implementations of this specification.

There are a number of security considerations that need to be taken into account by implementers of this specification.

Implementations MUST protect the private keys. Compromise of the private keys may result in the ability to forge signatures. Along with the private key, the implementation MUST keep track of which leaf nodes in the tree have been used. Loss of integrity of this tracking data can cause a one-time key to be used more than once. As a result, when a private key and the tracking data are stored on non-volatile media or stored in a virtual machine environment, failed writes, virtual machine snapshotting or cloning, and other operational concerns must be considered to ensure confidentiality and integrity.

When generating an LMS key pair, an implementation MUST generate each key pair independently of all other key pairs in the HSS tree.

An implementation MUST ensure that a LM-OTS private key is used to generate a signature only one time, and ensure that it cannot be used for any other purpose.

The generation of private keys relies on random numbers. The use of inadequate pseudo-random number generators (PRNGs) to generate these values can result in little or no security. An attacker may find it much easier to reproduce the PRNG environment that produced the keys, searching the resulting small set of possibilities, rather than brute force searching the whole key space. The generation of quality
random numbers is difficult, and [RFC4086] offers important guidance in this area.

The generation of hash-based signatures also depends on random numbers. While the consequences of an inadequate pseudo-random number generator (PRNG) to generate these values is much less severe than in the generation of private keys, the guidance in [RFC4086] remains important.

5. Operational Considerations

The public key for the hash-based signature is the key at the root of Hierarchical Signature System (HSS). In the absence of a public key infrastructure [RFC5280], this public key is a trust anchor, and the number of signatures that can be generated is bounded by the size of the overall HSS set of trees. When all of the LM-OTS signatures have been used to produce a signature, then the establishment of a new trust anchor is required.

To ensure that none of tree nodes are used to generate more than one signature, the signer maintains state across different invocations of the signing algorithm. Section 12.2 of [HASHSIG] offers some practical implementation approaches around this statefulness. In some of these approaches, nodes are sacrificed to ensure that none are used more than once. As a result, the total number of signatures that can be generated might be less than the overall HSS set of trees.

A COSE Key Type Parameter for encoding the HSS/LMS private key and the state about which tree nodes have been used is deliberately not defined. It was not defined to avoid creating the ability to save the private key and state, generate one or more signatures, and then restore the private key and state. Such a restoration operation provides disastrous opportunities for tree node reuse.

6. IANA Considerations

IANA is requested to add entries for hash-based signatures in the "COSE Algorithms" registry and hash-based public keys in the "COSE Key Types" registry.

6.1. COSE Algorithms Registry Entry

The new entry in the "COSE Algorithms" registry [IANA] has the following columns:
Name: HSS-LMS

Value: TBD1 (Value between -256 and 255 to be assigned by IANA, with a preference for -46)

Description: HSS/LMS hash-based digital signature

Reference: This document (Number to be assigned by RFC Editor)

Recommended: Yes

6.2. COSE Key Types Registry Entry

The new entry in the "COSE Key Types" registry [IANA] has the following columns:

Name: HSS-LMS

Value: TBD2 (Value to be assigned by IANA)

Description: Public key for HSS/LMS hash-based digital signature

Reference: This document (Number to be assigned by RFC Editor)

6.3. COSE Key Type Parameters Registry Entry

The new entry in the "COSE Key Type Parameters" registry [IANA] has the following columns:

Key Type: TBD2 (Value to be assigned above by IANA)

Name: pub

Label: TBD3 (Value to be assigned by IANA)

CBOR Type: bstr

Description: Public key for HSS/LMS hash-based digital signature

Reference: This document (Number to be assigned by RFC Editor)

7. References

7.1. Normative References

7.2. Informative References


[DSS] National Institute of Standards and Technology (NIST), "Digital Signature Standard (DSS)", FIPS Publication 186-6, 2013.


Appendix A.  Examples

This appendix provides a non-normative example of a COSE full message signature and an example of a COSE_Sign1 message. This section is formatted according to the extended CBOR diagnostic format defined by [RFC8610].

The programs that were used to generate the examples can be found at https://github.com/cose-wg/Examples.
A.1. Example COSE Full Message Signature

This section provides an example of a COSE full message signature.

Size of binary file is 2560 bytes.

%( RFC Editor: This example assumes that -46 will be assigned for the HSS-LMS algorithm. If another value is assigned, then the example needs to be regenerated. %)

```
98{
  [  
    / protected / h'a10300' / {  
      \ content type \ 3:0  
    } / ,  
    / payload / 'This is the content.',  
    / signatures / [  
      [  
        / protected / h'a101382d' / {  
          \ alg \ 1:-46 \ HSS-LMS  
        } / ,  
        / kid / 4:'ItsBig'  
      ],  
      / signature / h'000000000000000000000000391291de76ce6e24d1e2a9b60266519bc8ce889f814de0f000ed3129de3ab56bfa3bf47d007d844af7db749ea97215e82f4f56cbb473812c6a042ae39539898752c8960a276ec8a9f3eb900e25bdfe0ab8773aac136ae2146d67c65bb68630450a5db2c7c6403b77f6a9bf4d30a0219db5cccced84d7514f3cbbd19220020bf3045b0e5c695b32864f16f97da02f0cbf2ea70456b700732e30b03d27b5f8f3d6c871442e63e84b10f559f5dc594a219424c84ccc337078665653fc74034042813b0f5b5154f2f2cb291ad05ace7ace8e6031b209b2f47712d1c01e34b165a2f2e07f5a521a85afb3d2a6288947bcb5e2265d67b0d6192eb2f643964e2783d84aeac343f8e3571e4f0fcbec94e80470aa7252dc1733a553907e66c7b9f0b88b159dca2730ee47f13e7e134d305e5f53fac640b784a9b0f183ef841217325626f4f87c8d8cb9ea0fabb174ee0b7076cfc3945037cefdff3f1e6b5174412174c095b77c239373ec4c46a96199b66cad2990cbf5bb1abfde9e9107c7f7289395b5f2a433598ede0b1969f23db949a9fb54d33831dae6461a3655f8ff9b16c4dfc4b68689b193a557721252ac8a18d51c995344cc10c4c4e9ccfbb4e418bed0f334af165339e67254f4f1e995521e1be8a5665d59b57cd130903b42d07087d6d6f4ef8fc1e9e9071bb7a123fdec3d37687cdaaf0f4bf308407406917c17885b9431ad0980d36a6f8a8262562daa34f8aa0731a8357c060db8e80efef61bc323890e640363b98d175d4d6ebff800a71cfc864ec02837ed9de0e79f0f040acafdf56805cb273e631a395d23e86acfc6ea618a5afe1faa361cbdb5fefe9ebd70c95919ec3128e0dfbe9ca0f89fc0357671c05d41e7a010892c42e8e2af62a040f4e2140cb0b0807548f199307a555af233b9424f209b89f161032e413b047ae5aab0aa15643bb4c643446d2c9829eb256e7375ce9639047a244f44da446b7359556f3ab3484c56511c68a140dc0531f653105800d9f20990d4ebdc5ceea918d7ae95c0d7ec69a00d6a936b25fc19b9dfe5561
```
400f046191136c367038d6a9d0e0ae30ddc4733712cd5a2ae3e5351e7fe5c17e08
5b68c5cf0c64e95df2ca6f03dbd4480a2ed140a0dbf29d9463d5b9e41e34e649
c89ad5e43939c834c4746309c886d61f2f92155ca161b0ac9668082a947b5aba94
b35357d13df6f022adeaebf56912f68ae5d3a60214fe6d004c4d9f0af9eb0bf961cd
9f72751d46899c28d87080ba2ead3e8193f51a789706e32aacee9f4b14eeeca91a25
2fe8a94b3ed3c93398abbe7d217948c8e797ceabd4d7df6756f3099f2543ed3cb2b2
ac2ab4d253039c9e07cfd323cc3f9aa17977e05c5bce0f5954d51f16f0f52d3f93166
af68a90293f35d27a3adaefc58b5b0c5042bfa626a7d2a5d2ccda4af637e2d20c05
f1ac0ea3c802632e4341933612e07b6e85006137a59e2f5cd58392b654cf9e11e7
02c7b9152e4049f4676958b5bfc2862e25aba6ea9cb9d06c9911693eff24f2151c3cb885461438b8a2b142ddc39badd516e6fe30847a96b74
3f2494dd90a84f3e3cb0b38a587a9472a7589e41f4bdf8b7914803b0a4d9ac13
1882455ece23e50324c5feea21792c8e20b2d9f17e4388e6e8930314dd
7e9c629c46dfd92bc57c5f5d243f159d64691745dcd1645679d069649791c949cbdf2a2f
897a29be8576619c7a76516e6c941087441a81f119a83353b2b4ddd725a81a9d1
f6fd2a804c88d84e3682065574282aef23eaf648cfa7967af0b98fbd618567476
5f5f39bc6726c9a3f1f746f3cc0feea929b9c05dded76143c63dcd7018ab1b0c108ea9
01be32b9d911b6da13a1528c32a9694c899a772f8e1e0017ecec3b437e73772ca
06c5fddac94ad3df8ef3c1f5955a68c1d8cb0e83bd34371239da996f7a17ab
6a397442d261e8df913d0afb1e5752b9c1abc917b06ac69579fb1b05e49e2
d181784ac6dd160dc717b73bd2b28ef5508d474665d2ae7f54814c7e206fa9e9e2ec533
85d14d52f7679d95e0a50524f9f20dc7275b04d71d9167e33cbe6c4811f0c5a157e8a
1f9d67960456256545dbd173ccdd97661e9595ce47d63ead96e6e006a5ce6f49c7
7f7fe2e3f91be8e778cacc864d8fe0315dc731b8b98795f9b891605f52e11de8v09
ab4f28ee93d1711e73645b410abc5b18538ce3d4bd1a3e535ac8f6016656688c99d6a
8ed565ba4b132908360a5d5526ca5e25916124a77b7e8655odde128cd71ee651d44
96696900a02979e42bcbcc32c3b1eobf9ffaa4d5352e0eb03382222d2f2c2de4ce9
0c49c1a359e92ef971cd6cb06047a333c2ebe827eb65d2f2811fdebe0b0f12b204ae
0dd8e418f3691a60ceb0cecfb6f45f47883d6b9f3290590e912667406d6bbd6b3c6
e56e60a666b0d8c93b86e4e6269537a7878686fc86e6c150657db1a89d18e6e6efe7d88ff5b739b8610e392811e097a6a1c481e0fbd3463edf3f65a4e12ab0d9f13
022df2e7fda8ee0face73766c8ff6f446995b516f43cd59c70feecbd60a25e82850417
157f4f3f73e2c7a6f05199641cd0d99a78130ef7ba3833502ad4f3064f641a43d0
35e2a0edc034649aa905f196c5c386e652763e8696bd264aa109da6c3ca4a4
24e3ebc9b2373d262e886ad724dd56ea285e8e4b60bec92d45d7070a38877874525f
ef1a8f71547b902061461631d3b930f62582285238b51f7aef231673a5efe5add8c
b0e9992b6b5c65b0de4f2bb21462694766c28e5e7c834d3bcb6f6db7fbd84552522f2
2e9744f6df3da7167469899d1ba9029b298c2897cd2b36349828cf01107618
ecf14f8a7a8b9d2889d87e38b8ca3787674be5243fca64ca8f3d1054f62baf9398b
143a3a98f78bb8c891e5000dbabf7f70b712c45784f4d1c14265b58a30df5ebe1f5ea6d
067c509160b78e8532aca770c7be366ec07e78081892b0000000ed1ce86ce437
918d43faba79d38569a41c128703f6fb7704deed9384a66fbc362c948646b3c984
8038e69a91f7d9367f079cdd355d7c03d605f0c3683090084941bc4ebcabec128
7c81a46e6a267b57640a087878e0c8d8d61e030816b7f5b5dcd5
6211d7ca20b81f117d129529a7570cf79cf52a7028a48538ec4d3b383d5d62d26
246595c4f73a525a5edc203524ebbd8cc82ce019bc4977c6898ff5f3d3d10b0ba
e71696ce93c6a525465b96e9d075e383bb7543c675842bafbfc7cdd8843b3276c
A.2. Example COSE_Sign1 Message

This section provides an example of a COSE_Sign1 message.

Size of binary file is 2552 bytes.

{{( RFC Editor: This example assumes that -46 will be assigned for the HSS-LMS algorithm. If another value is assigned, then the example needs to be regenerated. )}}

```
18{
    [  
    /protected / h‘a101382d’ / {  
        /alg/ 1:-46 / HSS-LMS /
    },  
    /unprotected / {  
        /kid/ 4:‘ItsBig’
    },  
    /payload/ ‘This is the content.’,
    /signature/ h‘00000000000000391291de76ce624d1e2a9b60266519bc8ce889f814deb0fc00edd3129de3ab99a5b5ac783bdf0fe689f57fb204f1992dbc1ce2484f316c74be3f2094cfa8e96a4a9548ced0f78ee5d549510d190f647320448ae27ecce77249802a0c39c645bf8db08573af52c9391fd0e217f245c752c176b81514eb6e3067ef0b329225eeaa88c7d21635e32ae842d3f89018c0b6f1b84e61eeac348b690d7c6265c19f9d868952d99826e4d417b5279dd674cd951c306016cfefee3bfc5e5a5ad08b5bf453bc93995f26efce7c0c15ba257471cf2d8470993e8bd47ef9b9cf309ef895226e92be60683459009611de8bb9a43217956a0ab295bbda0fe4a393e7c4a6cd8a5314d6b02b377406d5a5e589e91fefaaf24e4c1682ba1f633cc778449932e40da651f71d3c19e38c6348d98b0c508324c0bcf7c5f0a80c14b4af200a739f96cdda94daf86ce80c76158d4f5c3cd2ba9f1393df4e7556887f916854085242a05e6b6c776659ec3d0d2fedae3fd1608a701c226f5fd83c9b1ed3152ddac742630e33908ce8f1da6174abe8d3568c9b76149eb077761a15b8fb1b18c5f9d14e448e216f375e1f96a52d39619459b131026143e8809bad40f5ef666cd3da272431e68670c0b4b2c3801e1e9025b1ebed218e0956967158ccc274c704adcc8c23c149a89eda25478742ddaced15f23384453e54021000b5d557313d4f271758680ed65e7fd681fdd19fb9a748acbb2377aac1387fd8b80e618eb7d69a368729ca9e092af91ebe1ca58435fe62734d1d530b35d02093a201c889ad37558b610f1ab00179a11f881600e944cedc47a7a6ed828009d7c61ffe9add5a540640e8e85dc05647b57589eab18e792f4631af62d4588a1818167274273c697ae0735be5dada7e224e3b178b
```
Appendix B. Acknowledgements

Many thanks to Roman Danyliw, Elwyn Davies, Scott Fluhrer, Ben Kaduk, Laurence Lundblade, John Mattsson, Jim Schaad, and Tony Putman for their valuable review and insights. In addition, an extra special thank you to Jim Schaad for generating the examples in Appendix A.

Author’s Address

Russ Housley
Vigil Security, LLC
516 Dranesville Road
Herndon, VA 20170
US

Email: housley@vigilsec.com