Much Ado About Encodings

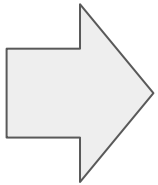
A sequel to the critically-acclaimed
“Great Private Key War of ‘25”

Mike Ounsworth
PQUIP 123, Madrid

LAMPS has had yet another centa-thread about PQ key formats …

116 replies when I counted on Jul 5, 2025

(plus a couple of spin-off threads)

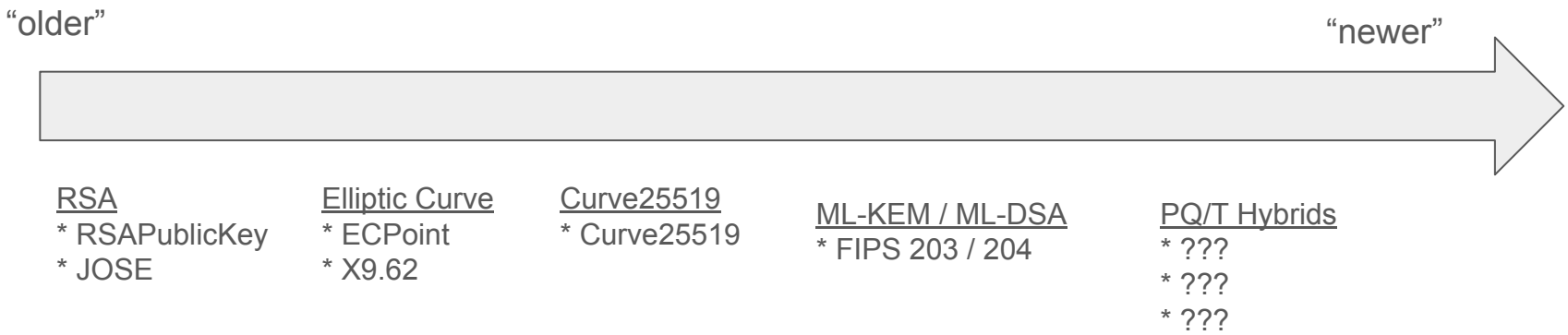


[lamps] Encoding of RAW OCTET STRING values Michael StJohns
[lamps] Re: [EXTERNAL] Encoding of RAW OCTET STRI… Mike Ounsworth
[lamps] Re: Encoding of RAW OCTET STRING values Nico Williams
[lamps] Re: [EXTERNAL] Encoding of RAW OCTET STRI… Viktor Dukhovni
[lamps] Re: [EXTERNAL] Encoding of RAW OCTET STRI… John Gray
[lamps] Re: [EXTERNAL] Encoding of RAW OCTET STRI… Viktor Dukhovni
[lamps] Re: [EXTERNAL] Encoding of RAW OCTET STRI… Viktor Dukhovni
[lamps] Re: [EXTERNAL] Encoding of RAW OCTET STRI… John Gray
[lamps] Re: [EXTERNAL] Encoding of RAW OCTET STRI… Peter Gutmann
[lamps] Re: [EXTERNAL] Encoding of RAW OCTET STRI… Bas Westerbaan
[lamps] Re: [EXTERNAL] Encoding of RAW OCTET STRI… Peter Gutmann
[lamps] Re: [EXTERNAL] Encoding of RAW OCTET STRI… Ilari Liusvaara
[lamps] Re: [EXTERNAL] Encoding of RAW OCTET STRI… Viktor Dukhovni
[lamps] Re: [EXTERNAL] Encoding of RAW OCTET STRI… Nico Williams
[lamps] Re: [EXTERNAL] Encoding of RAW OCTET STRI… David Benjamin
[lamps] Re: [EXTERNAL] Encoding of RAW OCTET STRI… Viktor Dukhovni
[lamps] Re: [EXTERNAL] Encoding of RAW OCTET STRI… Nico Williams
[lamps] Re: [EXTERNAL] Encoding of RAW OCTET STRI… Nico Williams
[lamps] Re: [EXTERNAL] Encoding of RAW OCTET STRI… Viktor Dukhovni
[lamps] Re: [EXTERNAL] Encoding of RAW OCTET STRI… Viktor Dukhovni
[lamps] Re: [EXTERNAL] Encoding of RAW OCTET STRI… Ilari Liusvaara
[lamps] Re: [EXTERNAL] Encoding of RAW OCTET STRI… Ilari Liusvaara
[lamps] Re: Tangential topic RSA private key form… Viktor Dukhovni
[lamps] Re: [EXTERNAL] Encoding of RAW OCTET STRI… Ilari Liusvaara
[lamps] Re: [EXTERNAL] Encoding of RAW OCTET STRI… Viktor Dukhovni
[lamps] Re: [EXTERNAL] Encoding of RAW OCTET STRI… Bas Westerbaan

This talk is advertising to come to LAMPS tomorrow.

But first, a history lesson

How many different ways can you encode each public key and signature type?



RSA Public Key

An RSA public key is two integers: *(n, e)*.

PKCS#1 (1991) or RFC2313 (v1.5, 1993) specified an ASN.1-based encoding [1]:

```
RSAPublicKey ::= SEQUENCE {
 modulus INTEGER, -- n
 publicExponent INTEGER -- e }
```

Most protocols will carry an RSA public key as an ASN.1 blob inside.

The exception is JOSE / JWK [2],

which carries “n”, “e” separately:

```
{
 "kty":"RSA",
 "kid":"1e9gdk7",
 "n":"w7Zdfmece8iaB0kiTY8pCtiBtzbp..",
 "e":"AQAB"
}
```

[1]: RFC2313 [2]: RFC7518

Elliptic Curve

… zoo of encodings: ECPoint vs X9.62.

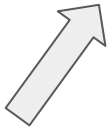
RFC3279 / RFC5480

```
ECPoint ::= OCTET STRING
```

The first octet of the OCTET STRING indicates whether the key is compressed or uncompressed. The uncompressed form is indicated by 0x04 and the compressed form is indicated by either 0x02 or 0x03 (see 2.3.3 in [SEC1]).

Multiple internal encodings possible

```
Ecdsa-Sig-Value ::=  SEQUENCE  {
     r      INTEGER,
     s      INTEGER
}
```



Not fixed-length … because ASN.1 …
The INTEGER type fluctuates by a few bytes since DER drops leading zeros.
Why didn't we choose a simple IntToBytes()?
… answer: because at the time, all crypto was ASN.1.

… a 1998 (RFC2528) / 2001 (RFC3279) decision that we're still living with, especially since it blurs the boundary between "crypto layer" and "protocol layer".

Elliptic Curve

… zoo of encodings: ECPoint vs X9.62.

X9.62 / SEC1:

Output: A signature $S = (r, s)$ on M consisting of a pair of integers r and s,

Integers encoded with:

2.3.7 Integer-to-Octet-String Conversion

Output: An octet string M of length $mlen$ octets.

Having both ASN.1 and non-ASN.1 encodings

…That was a mess, let's never do that again!

🎶🎶 Oh, give me the bytes, boys, and free my soul 🎶🎶

X9.62 is:

- Protocol-agnostic ✔
- Fixed-length ✔

Ed25519 / X25519

One encoding
to rule them all!

From RFC7748:

```
Alice's public key, X25519(a, 9): (u coordinate)

   8520f0098930a754748b7ddcb43ef75a0dbf3a0d26381af4eba4a98eaa9b4e6a
```

… A single 32-byte value. That’s all there is to it.

ML-DSA
(and similar for ML-KEM)

One encoding
to rule them all!

An ML-DSA public key has internal structure:

A 32-byte public random seed ρ.

the compressed polynomial vector $\mathbf{t}_1$.

But NIST gives us a binary encoding [1]:

Algorithm 22 pkEncode$(\rho, \mathbf{t}_1)$

Encodes a public key for ML-DSA into a byte string.

1: $pk \leftarrow \rho$
2: **for** i **from** 0 **to** $k-1$ **do**
3: $\quad pk \leftarrow pk \,||\, \text{SimpleBitPack}\,(\mathbf{t}_1[i], 2^{\text{bitlen}\,(q-1)-d} - 1)$
4: **end for**

[1]: FIPS 204

ML-DSA
(and similar for ML-KEM)

One encoding to rule them all!

When you need to transport that in a protocol, you treat the NIST encoding as an opaque blob

In X.509 [1]:

```
BIT STRING { mldsa_pk }
```

In JWT [2]:

```
{
   "kty": "AKP",
   "alg": "ML-DSA-44",
   "pub": "unH59k4Ru...DZgbTP07e7gEWzw4M"
}
```

[1]: draft-ietf-lamps-dilithium-certificates-12
[2]: draft-ietf-cose-dilithium-07

ML-DSA
(and similar for ML-KEM)

We **ARE NOT** making protocol-specific encodings of the ML-DSA key.

For example, you will **never** see:

```
SEQUENCE {
        rho BIT STRING,
        t1_0 BIT STRING,
        …,
        t1_k-1 BIT STRING
}
```

or

{ “rho”: <b64>, “t1_0”: <b64>, …, “t1_k-1”: <b64>}

If you do this, you will be sleeping in the IETF dog house!

Why do we prefer “one encoding to rule them all”?

Because in modern crypto architectures like a clean separation between:

- Protocol layer – ex.: X.509, JWT, etc, where you handle message data and wire encodings,
  and
- Crypto layer – ex.: openssl, python.cryptography, etc, where you have APIs for KeyGen(), Sign(), etc.

In this model, the protocol layer has no business needing to parse the internal components of the key or ciphertext material; just pass it opaquely to the crypto layer.

Why do we prefer “one encoding to rule them all”?

Modern thinking is that in certain contexts, KEMs want to bind a ciphertext / shared secret to the public key it was intended for:

Keeping Up with the KEMs: Stronger Security Notions for KEMs and Automated Analysis of KEM-based Protocols

Authors: Cas Cremers, Alexander Dax, Niklas Medinger | Authors Info & Claims

CCS '24: Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security
Pages 1046 - 1060 • https://doi.org/10.1145/3658644.3670283

X-Wing shared key (32 bytes):

SHA3-256 (

\./ /^\ (6 bytes)	ML-KEM-768 shared key (32 bytes)	X25519 shared key (32 bytes)	X25519 ciphertext (32 bytes)	X25519 public key (32 bytes)

)

RFC9180 (HPKE):

```
pkRm = SerializePublicKey(pkR)
kem_context = concat(enc, pkRm)

shared_secret = ExtractAndExpand(dh, kem_context)
```

Why do we prefer “one encoding to rule them all”?

There are discussions about whether signature schemes should also be binding public keys (for example within the ctx) to strongly prevent cross-key attacks.

What would happen if, for example you use an “x5c” to sign a JWT … would you pass in X.509’s or JWT’s encoding of the public key?

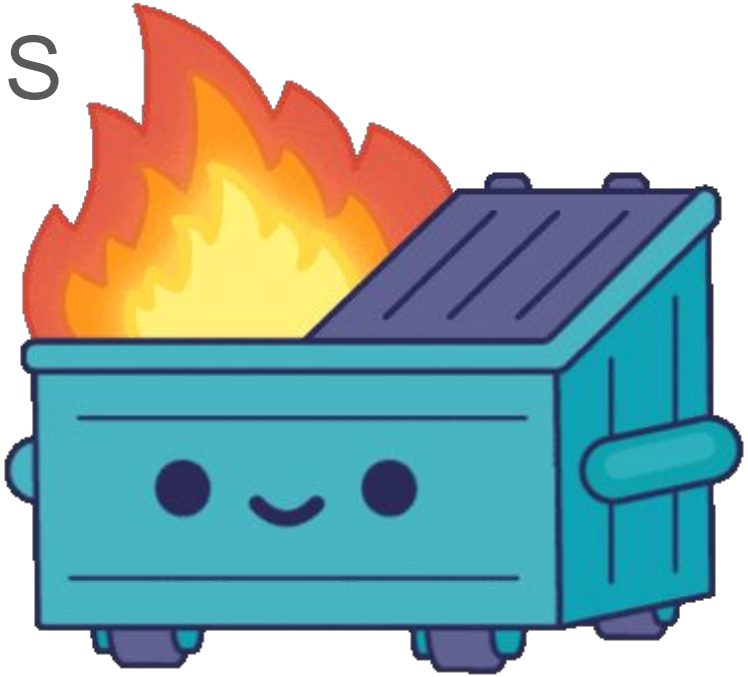
The more we do constructions that hash in the public key, the more it matters that the sender and receiver use the same encoding for the public keys and ciphertexts.

Having a single encoding for the algorithm, regardless of protocol, solves all these problems.

So why am I giving this presentation?

So why am I giving this presentation?

Because HYBRIDS

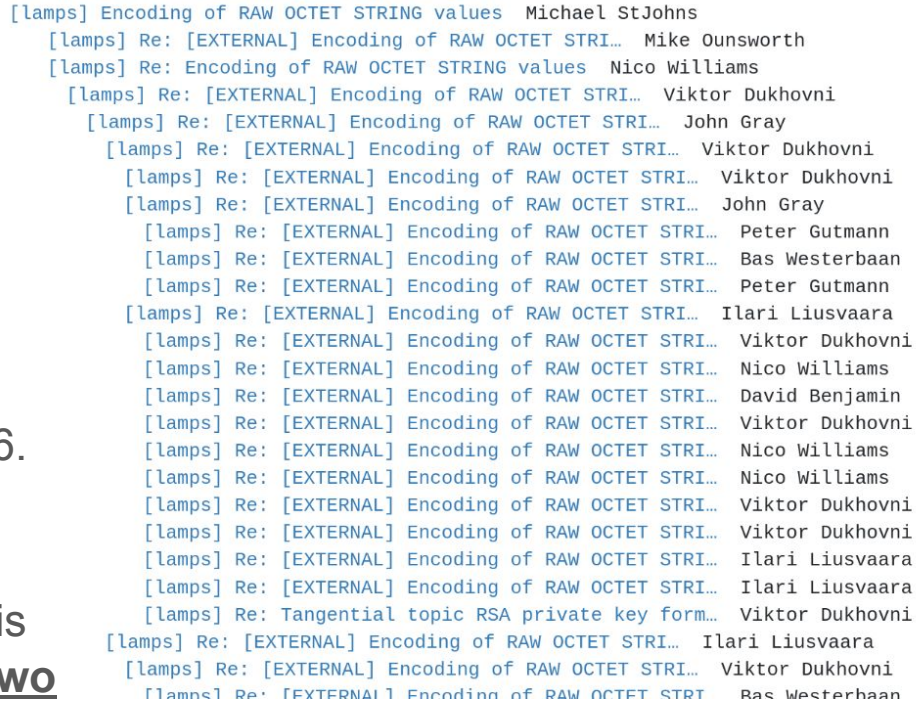


Technically,

how to encode

a hybrid on the wire.

How to encode a hybrid?



Specifically, I'm talking about "composite-type" hybrids [1], ex.:

- X-Wing
- TLS's MLKEM768X25519,
- LAMPS' id-MLKEM768-X25519-SHA3-256.

The core debate is whether a hybrid algorithm is **one algorithm** with one opaque encoding, or **two algorithms** that should be decomposable at the protocol layer?

[1]: draft-terminology

The core of the debate

Do hybrids need protocol-level separability to handle multiple underlying crypto modules?

Especially in PKIX-land, we expect to have to support cases where the two halves of the composite come from two different crypto modules.

ex.:

- ECDSA from your existing FIPS-certified module, and ML-DSA from your new not-yet-certified module.
- RSA from a smartcard or TPM, and ML-DSA from software.
- etc

Question: does supporting that necessitate composite pubkey, privkey, sig, ciphertext values that are protocol-level decomposable?

Encoding options for composite

1. “Raw bytes”:

```
Pubkey: mldsaPK || tradPK

Sig:    r || mldsaSig || tradSig
```

2. “ASN.1 SEQUENCE”:

Pubkey: SEQUENCE { mldsaPK OCTET STRING, tradPK <asn1_type> }

Sig: SEQUENCE { r OCTET STRING, mldsaSig OCTET STRING, tradSig <asn1_type> }

3. “Full SubjectPublicKeyInfo”:

Pubkey: SEQUENCE { mldsaPK SubjectPublicKeyInfo, tradPK SubjectPublicKeyInfo }

How to encode a hybrid?
Arguments for “One opaque encoding”

- The security and negotiability properties of a composite assume that it will be treated as one algorithm by the application layer and protocol layer, and only decomposed inside the crypto library layer.
  - Therefore, there is no reason to make it easily decomposable in a protocol-specific way.
- Having more than one possible encoding leads to problems with combiners that use combiners to achieve PK-BIND or CT-BIND.
  - Ex.: if encrypting a JWT payload for an ML-KEM public key in an X.509 certificate, should you use the JWT public key encoding on the X.509 public key encoding?

How to encode a hybrid?
Arguments for “Decomposable encoding”

- If you want to future-proof to handle components with variable-length values, then you end up needing to re-invent a length-tagged encoding, so why not use an existing one (DER)?
- Protocol-level decomposability means the protocol-layer can enforce policy on the components.
    - Point of debate: is there meaningful policy to enforce beyond what’s already specified by the spec for X-Wing, MLKEM768X25519, id-MLKEM768-X25519-SHA3-256?
- Fits more cleanly into crypto libraries which are already protocol-specific.
    - ex.: OpenSSL is fundamentally an X.509 library where public keys need to be in a an ASN.1 SPKI anyway.
    - Here an ASN.1-based encoding helps separate the two keys to feed them into separate underlying crypto modules.

My Opinionated Answer 🤔

As an author of the LAMPS Composite-ML-DSA and Composite-ML-KEM specs, we are choosing to define an opaque, protocol-agnostic encoding that allow these algorithms to be easily re-used across protocols.

Essentially, we are taking the view that Composites are an *algorithm*; they only happen to be in LAMPS because there is no dedicated Crypto WG, not because they are fundamentally part of X.509.

That means that Composite ML-DSA and Composite ML-KEM should have a unique opaque binary encoding which is intentionally not separable at the protocol layer.

Composite ML-DSA in JOSE

4.4. Encoding Rules
In each combination, the byte streams of the keys are directly concatenated, and the byte streams of the signatures are directly concatenated with the randomizer r.

Randomizer r || Signature of the 1st Algorithm || Signature of the 2nd Algorithm

Composite ML-DSA in TLS

Does not modify the public key or signature value encoding from the underlying Composite ML-DSA spec [I-D.ietf-lamps-pq-composite-sigs].

But what about Composite Private Keys?

- The arguments made so for are about public values: Public Keys, Signature Values, Ciphertexts. Do these arguments also apply to Private Keys?
- In particular, should Composite Private Keys allow

  ```
  CHOICE { SEED, EXPANDED, BOTH }
  ```

  for the ML-DSA / ML-KEM component?
  - This would force A) multiple valid encodings, B) debates about how to encode that CHOICE (ASN.1?)
- Main debate is:
  - SEED is the superior format
    vs
  - Cryptographic hardware is going through FIPS certification now that does not support seed-based priv. key export … will this hurt adoption of Composites?

This will be debated at LAMPS tomorrow.

Summary

- Since Bangkok, LAMPS has had another centa-thread about PQ key and signature encodings.
- This time, it's about whether Composites should be considered an atomic algorithm and encoded as raw bytes, or wrapped in ASN.1 to make them more easily decomposable.
- (I think?) we have settled on "raw bytes".
  - Although, this is "rough consensus" not "full consensus".
  - This aligns with the companion drafts for Composites in JOSE WG and TLS WG.
  - This is well-motivated by the historical struggles we've had with multiple encodings for ECDSA.
- The debate may not be settled for Private Keys.