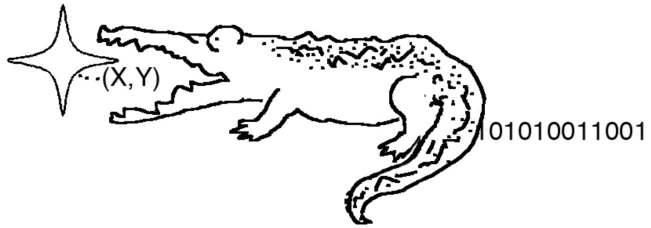
Kemeleon Encodings

Making ML-KEM public keys and ciphertexts indistinguishable from random

6 November 2024

IETF121: CFRG

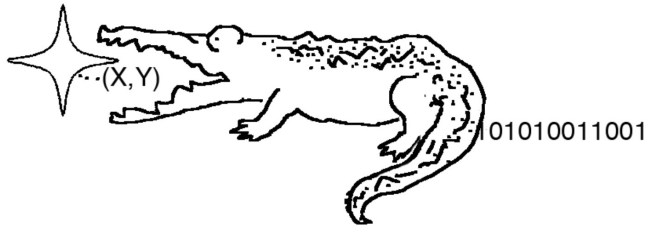
Previously: Elligator



- Mapping elliptic curve points to uniform random bytestrings.
- Used in protocols for censorship-resistance, hash-to-curve, VRFs, OPRFs, and more...
- Specified in RFC 9380.

Bernstein, Hamburg, Krasnova, Lange. Elligator: Elliptic-curve points indistinguishable from uniform random strings. CCS2013

Previously: Elligator



- Mapping elliptic curve points to uniform random bytestrings.
- Used in protocols for censorship-resistance, hash-to-curve, VRFs, OPRFs, and more...
- Specified in RFC 9380.

- As we transition to post-quantum algorithms, there is a need for a replacement.
- ML-KEM public keys and ciphertexts are distinguishable from random.

Bernstein, Hamburg, Krasnova, Lange. Elligator: Elliptic-curve points indistinguishable from uniform random strings. CCS2013

Kemeleon

ML-KEM/Kyber public keys

- vector of polynomials with coefficients mod q

$[\ a_1\ ][\ a_2\ ][\ a_3\ ]...[\ a_b\ ]$ $a_i \in \mathbb{Z}_q$ (q=3329, each a_i requires 12 bits)

↑ ↑ ↑ ↑

most sig. bit of each value biased towards 0

Günther, Stebila, Veitch. Obfuscated Key Exchange. CCS2024

Kemeleon

ML-KEM/Kyber public keys

- vector of polynomials with coefficients mod q

$[\ a_1\ ][\ a_2\ ][\ a_3\ ]...[\ a_b\ ]$ $a_i \in \mathbb{Z}_q$ (q=3329, each a_i requires 12 bits)

↑ ↑ ↑ ↑

most sig. bit of each value biased towards 0

- Encoding for public keys:
  1. accumulate into one big number
  2. rejection sampling: reject if msb is 1

$[\ A = a_1 + a_2 \cdot q + a_3 \cdot q^2 + \cdots + a_b \cdot q^{b-1}\ ]$ A is a number mod q^{b-1}

↑

most sig. bit still biased towards 0

Günther, Stebila, Veitch. Obfuscated Key Exchange. CCS2024

Kemeleon

ML-KEM/Kyber public keys

- vector of polynomials with coefficients mod q

$[\ a_1\ ][\ a_2\ ][\ a_3\ ]...[\ a_b\ ]$ $a_i \in \mathbb{Z}_q$ (q=3329, each a_i requires 12 bits)

↑ ↑ ↑ ↑

most sig. bit of each value biased towards 0

- Encoding for public keys:
  1. accumulate into one big number
  2. rejection sampling: reject if msb is 1

Encoded pks **~2.5% smaller** than regular
(19/28/38 bytes for ML-KEM-512/768/1024)

$[\ A = a_1 + a_2 \cdot q + a_3 \cdot q^2 + \cdots + a_b \cdot q^{b-1}\ ]$ A is a number mod q^{b-1}

↑

most sig. bit still biased towards 0

Günther, Stebila, Veitch. Obfuscated Key Exchange. CCS2024

Kemeleon

ML-KEM/Kyber ciphertexts

- vector of polynomials with coefficients mod q

22: $c_1 \leftarrow \mathsf{ByteEncode}_{d_u}(\mathsf{Compress}_{d_u}(\mathbf{u}))$
23: $c_2 \leftarrow \mathsf{ByteEncode}_{d_v}(\mathsf{Compress}_{d_v}(v))$
24: **return** $c \leftarrow (c_1 \| c_2)$

- compressed before being returned by Encap
- compressed ciphertexts are no longer uniformly distributed (compress performs some rounding)

Kemeleon

ML-KEM/Kyber ciphertexts

- vector of polynomials with coefficients mod q

22: $c_1 \leftarrow \mathsf{ByteEncode}_{d_u}(\mathsf{Compress}_{d_u}(\mathbf{u}))$
23: $c_2 \leftarrow \mathsf{ByteEncode}_{d_v}(\mathsf{Compress}_{d_v}(v))$
24: **return** $c \leftarrow (c_1 \| c_2)$

- compressed before being returned by Encap
- compressed ciphertexts are no longer uniformly distributed (compress performs some rounding)

- Encoding for ciphertexts:
  1. Decompress and "recover" randomness lost from compression
  2. Apply previous technique for public keys

Kemeleon Variants

1. No rejection sampling:
   - Use a technique from Elligator squared (Tibouchi) to avoid rejection (100% success probability), at a cost of slightly bigger output

Tibouchi. Elligator Squared: Uniform Points on Elliptic Curves of Prime Order as Uniform Random Strings. FC2014

Kemeleon Variants

1. No rejection sampling:
   - Use a technique from Elligator squared (Tibouchi) to avoid rejection (100% success probability), at a cost of slightly bigger output

2. Deterministic:
   - Public keys: Store randomness in secret key
   - Ciphertexts: Derive from the shared secret key K, a “new” key K’ (to be used as the shared secret) and a seed to be used to generate randomness for Kemeleon encoding

Kemeleon Variants

1. No rejection sampling:
   - Use a technique from Elligator squared (Tibouchi) to avoid rejection (100% success probability), at a cost of slightly bigger output

2. Deterministic:
   - Public keys: Store randomness in secret key
   - Ciphertexts: Derive from the shared secret key K, a "new" key K' (to be used as the shared secret) and a seed to be used to generate randomness for Kemeleon encoding

3. Faster:
   - Kemeleon encoding requires computing with large (~750 to 1500 byte) integers
   - At the cost of a higher rejection rate, we can compute with coefficients mod q-1 = 13*2^8.
   - Improved efficiency by using only ~237 to 475 byte integers
   - (same output size)

Which variant(s) are you interested in?

	Output size (bytes)	Success probability	Notes
	ML-KEM-512 / 768 / 1024 Public keys: 800 / 1184 / 1568 Ciphertexts: 768 / 1088 / 1568	Elligator: 0.50	
Kemeleon (original)	ML-KEM-512 / 768 / 1024 Public keys: 781 / 1156 / 1530 Ciphertexts: 877 / 1252 / 1658	ML-KEM-512 / 768 / 1024 Public keys: 0.56 / 0.83 / 0.62 Ciphertexts: 0.51 / 0.77 / 0.57	*Larger integer arithmetic* (~750 to 1500 byte integers)
No rejection	*Kemeleon* + 16 in all cases	1.00	*Larger integer arithmetic* (~775 to 1515 byte integers)
Faster	*Same as Kemeleon (original)*	ML-KEM-512 / 768 / 1024 Public keys: 0.49 / 0.29 / 0.53 Ciphertexts: 0.45 / 0.25 / 0.47	*Smaller integer arithmetic* (~237 to 475 byte integers)

WIP Draft: https://github.com/ssveitch/draft-kemeleon Contact: shannon.veitch@inf.ethz.ch