Kemeleon:

Elligator-like Obfuscation for ML-KEM

Felix Günther
IBM Research – Zurich

Michael Rosenberg
Cloudflare

Douglas Stebila
University of Waterloo

Shannon Veitch
ETH Zurich

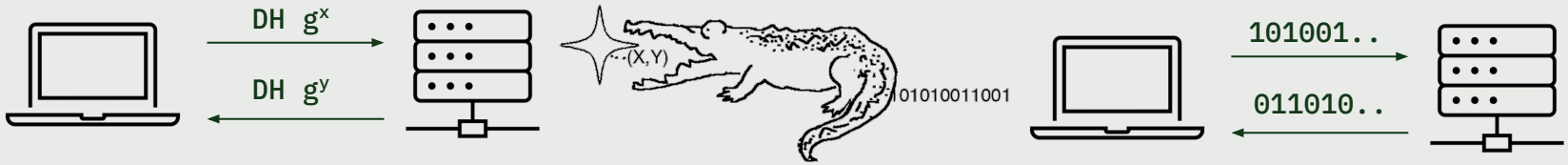
Uniform Representations

Internet protocols **hide metadata** to protect user privacy, dissuade protocol fingerprinting, and prevent network ossification

- TLS 1.3 Encrypted Client Hello, QUIC, obfs4, Shadowsocks, ...
- "Fully encrypted" protocols, with **obfuscated key exchange**

Some PAKEs need to operate on random bytestrings

Previously: Elligator maps elliptic curve public keys to random bytestrings



What about post-quantum key exchanges? Can use Saber or FrodoKEM.

What about standardized post-quantum key exchanges?

Byte Distribution of ML-KEM Public Keys

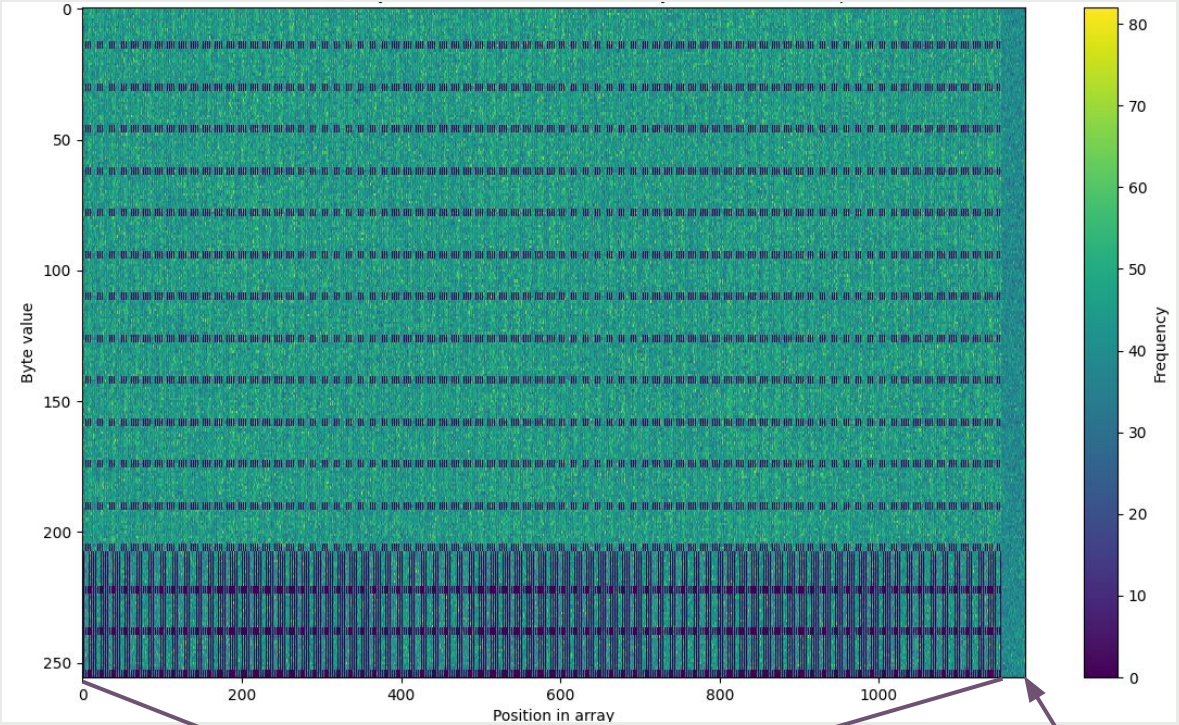
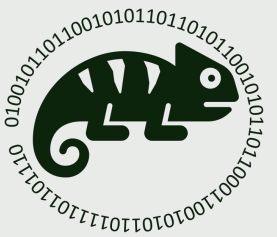


Image credit: Jack Wampler

Kemeleon: Rejection-Sampling Pubkeys



ML-KEM 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

Kemeleon encoding for public keys

1. Accumulate into one **big integer**
2. Rejection sampling: **reject if msb is 1**

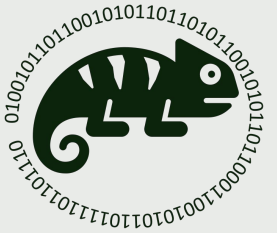
Encoded public keys **~2.5% smaller** than regular (19/28/38 bytes for ML-KEM-512/768/1024)

MLKEM-768 **likelihood of rejection is 17%**

$[\ A = a_1 + a_2 \cdot q + a_3 \cdot q^3 + \cdots + a_b \cdot q^{b-1}\ ]$ A is a number mod q^{b-1}

Most sig. bit still biased towards 0

Kemeleon: Rejection-Sampling Ciphertexts



ML-KEM ciphertexts

Vector of polynomials with coefficients mod q

Algorithm 14 K-PKE.Encrypt$(\text{ek}_{\text{PKE}}, m, r)$

22: $c_1 \leftarrow \text{ByteEncode}_{d_u}(\text{Compress}_{d_u}(\mathbf{u}))$
23: $c_2 \leftarrow \text{ByteEncode}_{d_v}(\text{Compress}_{d_v}(v))$

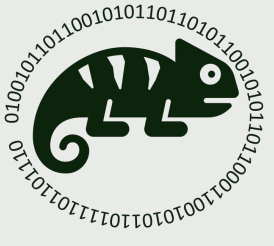
Compression step in Encap performs rounding which results in a non-uniform ciphertext distribution.

$$\text{Compress}_d : \mathbb{Z}_q \longrightarrow \mathbb{Z}_{2^d}$$
$$x \longmapsto \lceil (2^d/q) \cdot x \rfloor \bmod 2^d .$$

Kemeleon encoding for ciphertexts:

1. Decompress and "recover" randomness from ciphertexts
2. Use same rejection-sampling method as before

Kemeleon without Rejection



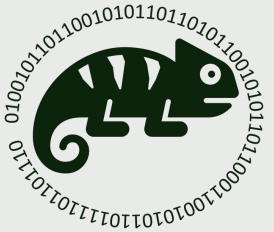
Applying techniques from Tibouchi 2014 (Elligator^2):

1. Take the big integer output from encoding with bytelength < n
2. Randomly pad it to n + 32 bytes (or alternative value, depending on security requirements)

Encoded public keys are **~ same size** as in standard ML-KEM.

Likelihood of rejection is **0%**

Using Kemeleon with ML-KEM: an OKEM



ML-Kemeleon.KGen()

```
1.  repeat
2.    (sk,pk) <-$ MLKEM.KGen()
3.    pk' <- Kemeleon.EncodePk(pk)
4.  until pk' != ⊥
5.  return (sk,pk')
```

ML-Kemeleon.Encap(pk')

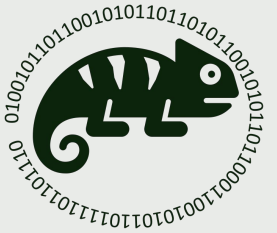
```
1.  pk <- Kemeleon.DecodePk(pk')
2.  repeat
3.    (c,K) <-$ MLKEM.Encap(pk)
4.    c' <- Kemeleon.EncodeCtxt(c)
5.  until c' != ⊥
6.  return (c',K)
```

- **IND-CCA**: IND-CCA of ML-KEM
- **SPR-CCA**: SPR-CCA of ML-KEM and ciphertext uniformity
- **ctxt uniformity**: SPR-CCA of ML-KEM
- **pk uniformity**: reduces to MLWE

\+ small loss from rejection rates in each case

Note: while Elligator is statistically uniform, Kemeleon relies on MLWE assumption.

Using Kemeleon: Dos and Don'ts



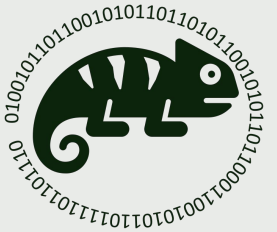
Dos!

- Append the seed as usual to the encoded portion of the public seed
- Consider a constant-time implementation for big integer arithmetic, if this is in your threat model (also, consider timing side channels due to rejection sampling)

Don'ts!

- Use randomness derived from the KEM shared secret to seed the encoding algorithm (i.e., careful with key separation)
- Reveal randomness used for the encoding algorithm (i.e., randomness must be kept secret)

What can you do with an OKEM?



Combine OKEMs

- Requires 1 statistical OKEM (DHKEM+Elligator) and 1 computational OKEM (ML-Kemeleon)

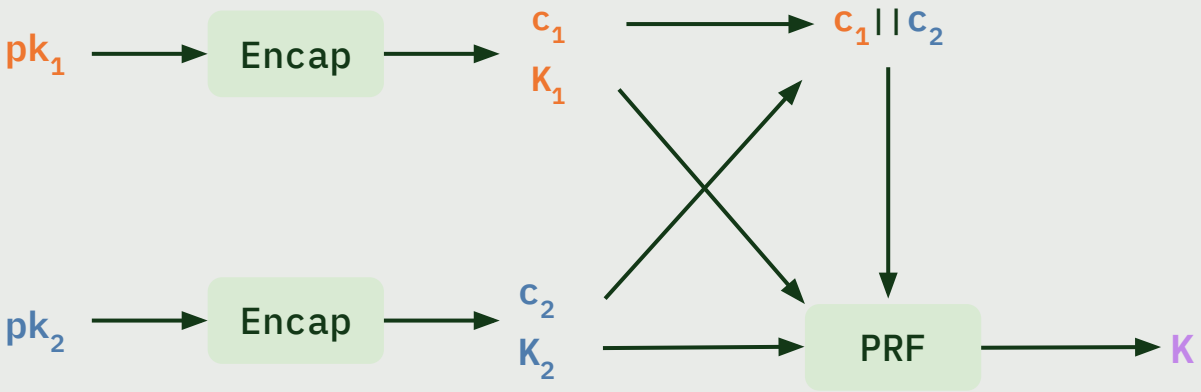
Make hybrid obfuscated key exchange

- Key agreement that looks like random. Think obfs4 (Tor bridge protocol)

Make hybrid password-authenticated key exchange (PAKE)

- First hybrid PAKE with security against adaptive corruptions

Hybrid KEMs: The Parallel Approach



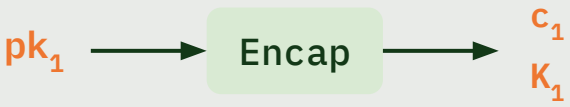
Approach used in hybrid TLS 1.3, Xyber, X-Wing, ...

- ☑ Hybrid IND-CCA
- ☒ Hybrid Obfuscation (also, SPR-CCA, which implies anonymity)

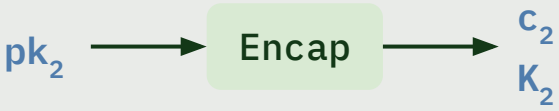
Outer-Encrypts-Inner Nested Combiner (OEINC)

Outer-Encrypts-Inner Nested Combiner (OEINC)

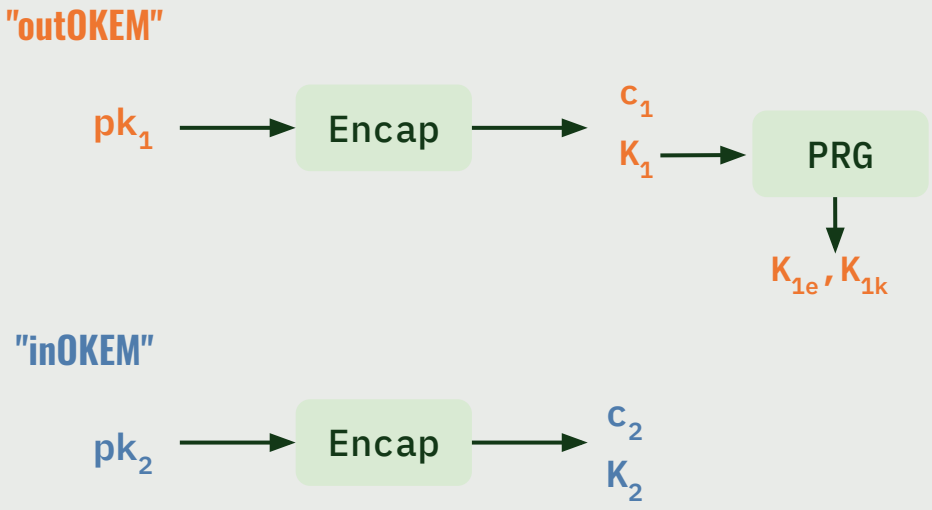
"outOKEM"



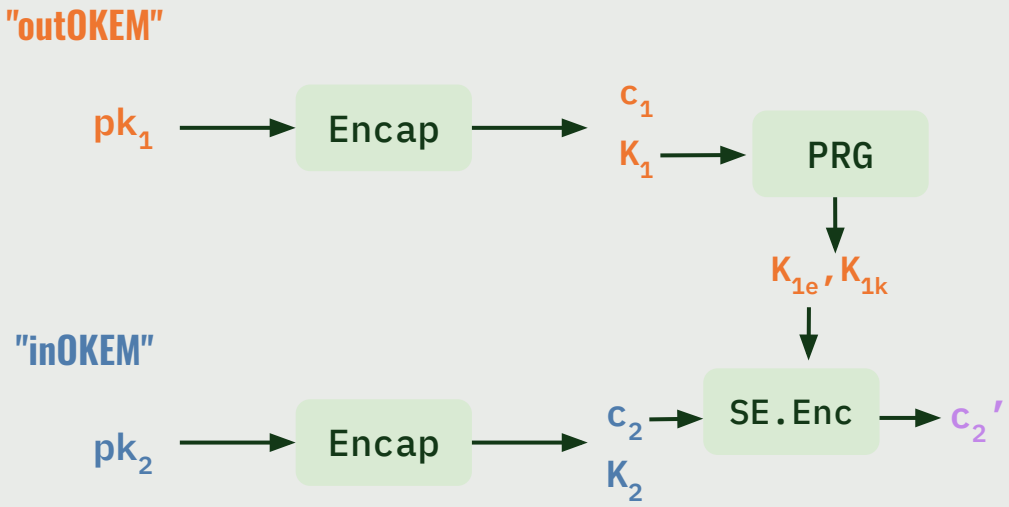
"inOKEM"



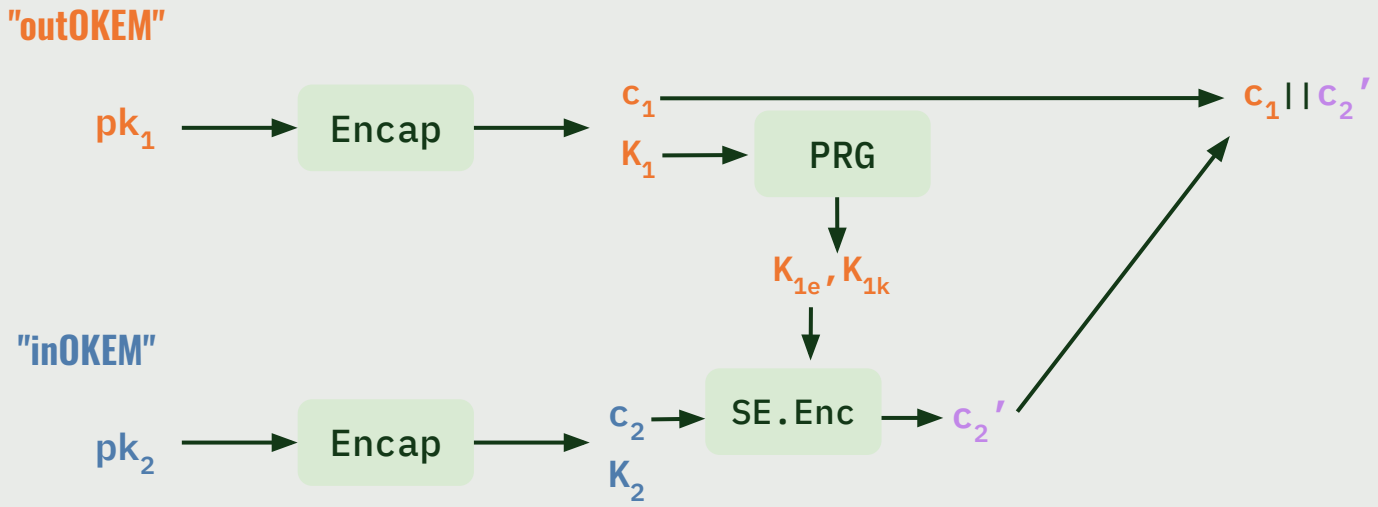
Outer-Encrypts-Inner Nested Combiner (OEINC)



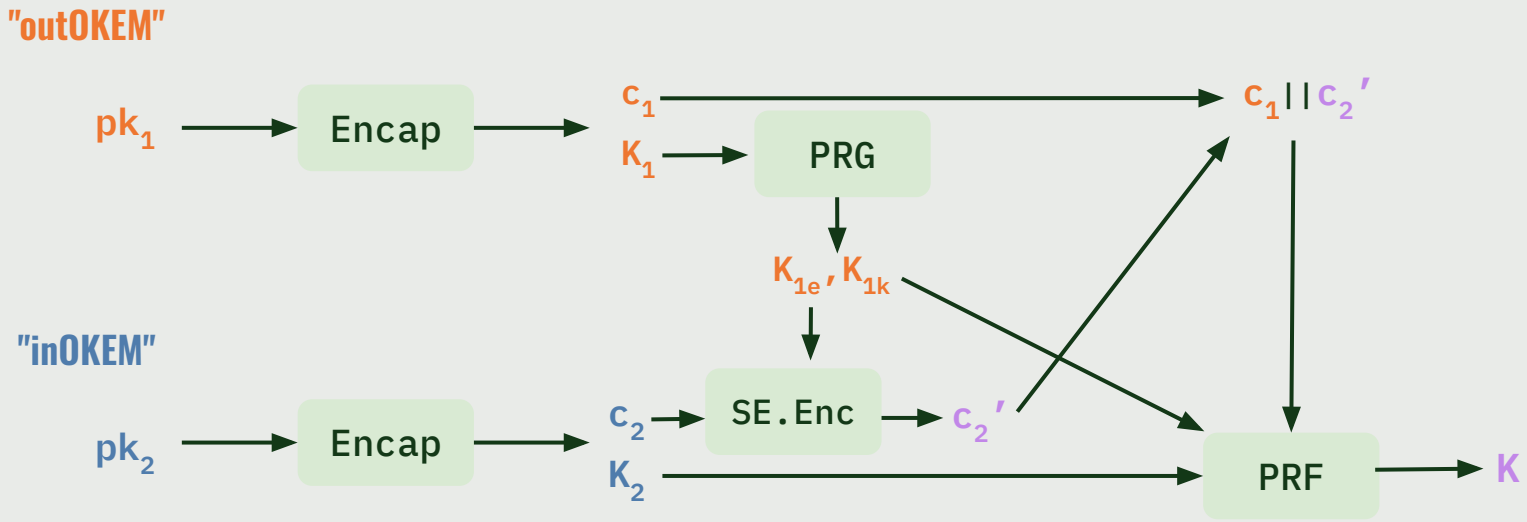
Outer-Encrypts-Inner Nested Combiner (OEINC)



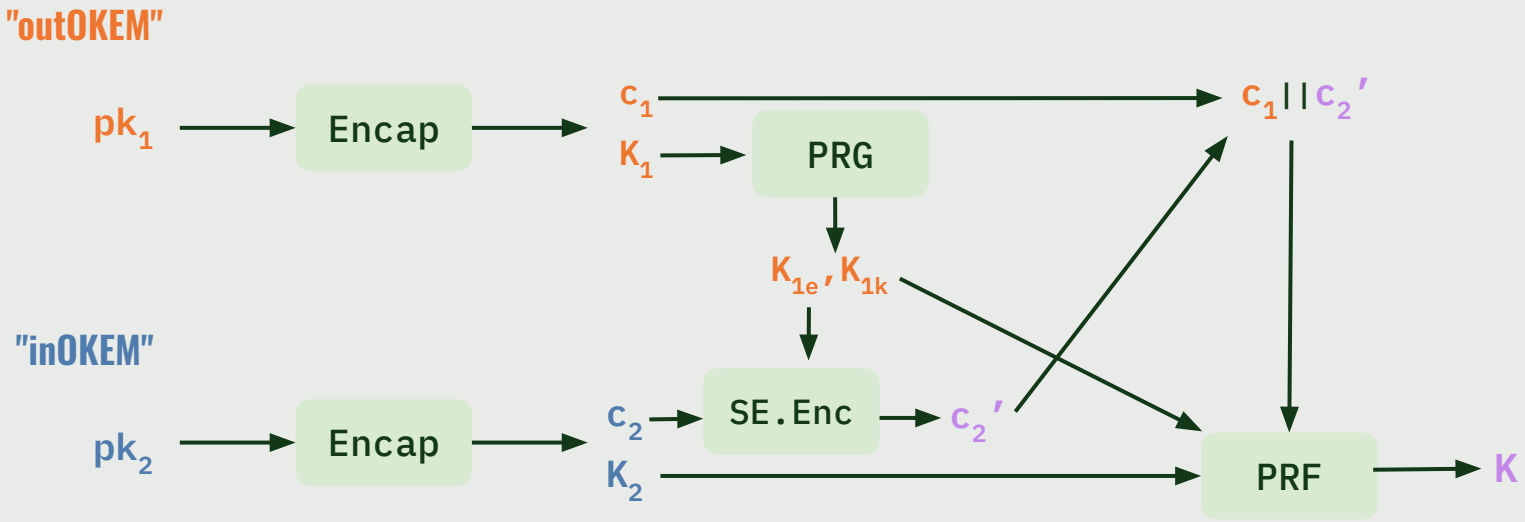
Outer-Encrypts-Inner Nested Combiner (OEINC)



Outer-Encrypts-Inner Nested Combiner (OEINC)



Outer-Encrypts-Inner Nested Combiner (OEINC)



- ☑ Hybrid IND-CCA
- ☑ Minimal overhead: 1 PRG + 1 XOR
- ☑ Hybrid Obfuscation (also, SPR-CCA, which implies anonymity)

Instantiating OEINC

Security Properties

Requires:

- outOKEM must have **statistical strong ciphertext uniformity** (ciphertexts must look uniform, even if you know sk, pk)

Achieves:

- **IND-CCA** outOKEM is IND-CCA or inOKEM is IND-CCA
- **SPR-CCA** outOKEM is SPR-CCA or inOKEM is SPR-CCA
- **Ciphertext uniformity** outOKEM is IND-CCA or inOKEM is ct-unif
- **Public key uniformity** outOKEM is pk-unif and inOKEM is pk-unif

We don't get hybrid public key uniformity! (Likely impossible)

We also **don't always need hybrid pk-unif**

outOKEM can be DHKEM

sk = x pk = xG

ct = Elligator2(r · pk)

inOKEM can basically be any ct-unif KEM

(and pk-unif if you want it)

Concrete Instantiation

outOKEM = DHKEM[Ristretto]+Elligator2

inOKEM = ML-Kemeleon/Saber/Frodo

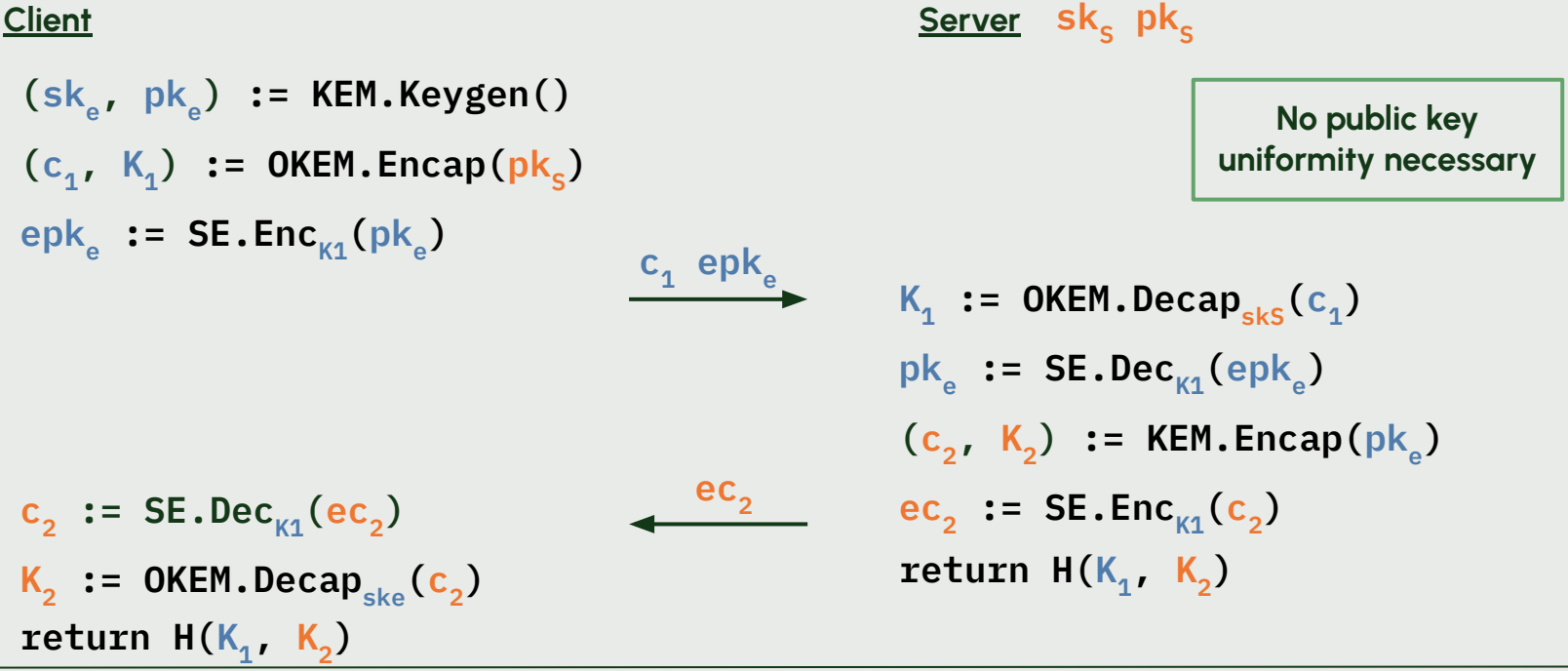
Applications of OEINC: Obfuscated Key Exchange

Drivel: A Hybrid Obfuscated Key Exchange Protocol

(O)KEM-based AKE

Pubkeys are encrypted with intermediate secrets

No public key uniformity necessary



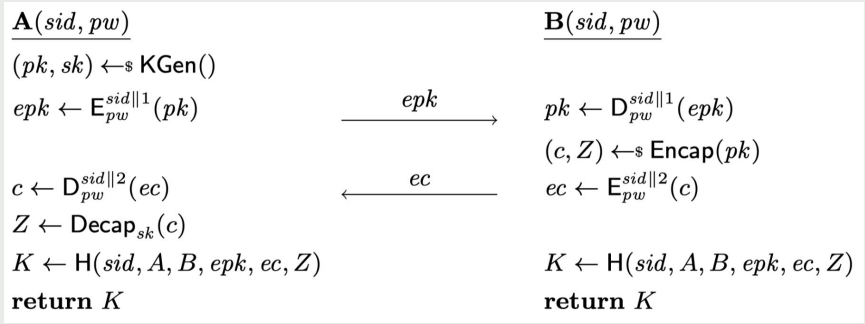
Applications of OEINC: Hybrid PAKE

Password authenticated key exchange (PAKE)

- Parties w/ **low-entropy password** want to establish a **high-entropy shared secret**:
  - **Active adversary has 1 pw guess** per protocol execution
  - **Passive adversary has no advantage** at all

KEM-based PAKEs (NoIC, CHIC, HIC, CAKE, OCAKE, ...)

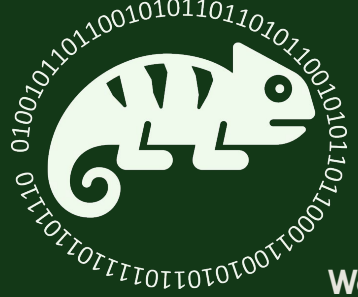
- CAKE proven secure in the UC model with **adaptive corruptions** (adversaries can corrupt any user at any time)
- **Needs ciphertext and public key uniformity**
- First hybrid PAKE with security against adaptive corruptions



We can instantiate CAKE with **OEINC[DHKEM+Elligator, StatFrodoKEM]**

This is **2 rounds.** Other PAKEs are 3 rounds or inefficient (350x slowdown).

7.5x comms overhead compared to 3-round PAKEs



Thanks! Questions?

We made:

- an OKEM from ML-KEM
- an OKEM combiner

We got:

- Hybrid obfuscated key exchange
- Hybrid PAKE

References:

- Günther, Stebila, Veitch. **Obfuscated Key Exchange.** CCS 2024. ia.cr/2024/1086
- Günther, Stebila, Veitch. **Kemeleon Encodings.** Internet-Draft. https://datatracker.ietf.org/doc/draft-veitch-kemeleon/
- Günther, Rosenberg, Stebila, Veitch. **Hybrid Obfuscated KEMs and Key Exchange.** IACR ePrint soon!