Misbinding KEMs

NIST Workshop on Guidance for KEMs

February 25/26, 2025

Speakers

Sophie Schmieg

Senior Staff Cryptography Engineer, Google

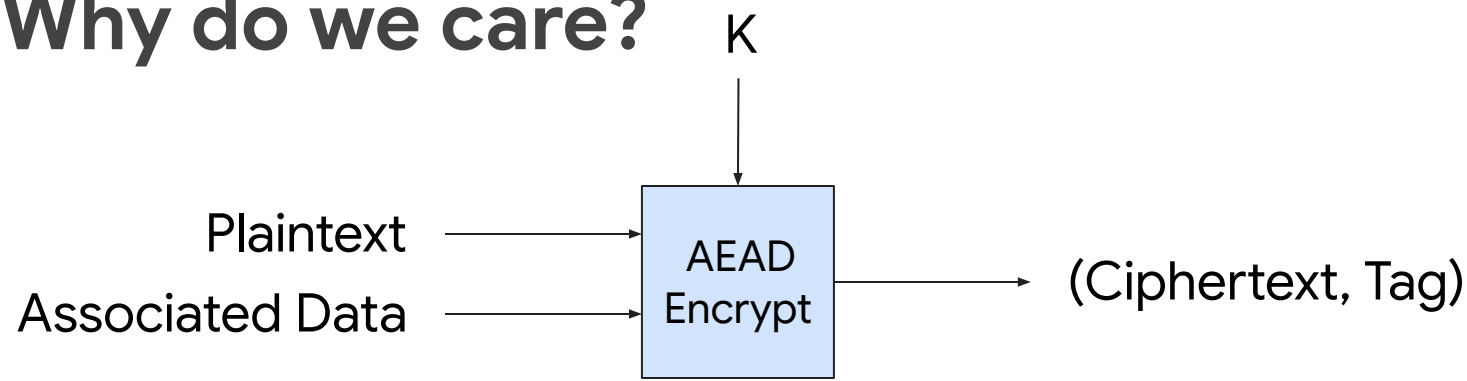
sschmieg@google.com

Deirdre Connolly

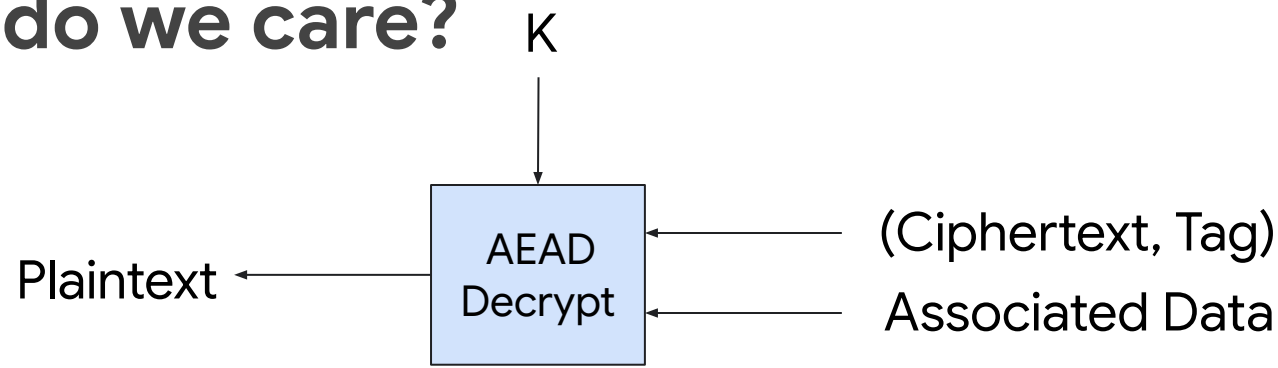
Senior Staff Standardization Research Engineer, SandboxAQ

durumcrustulum@gmail.com

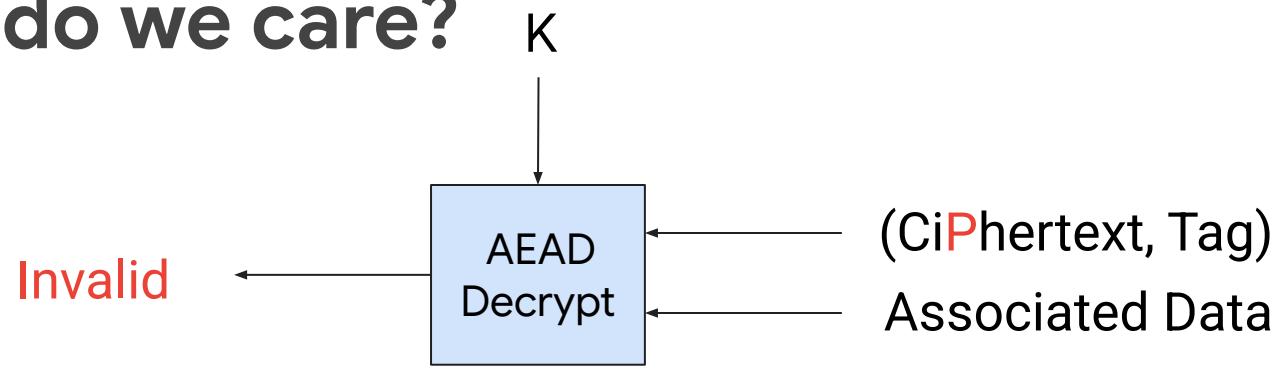
Why do we care?



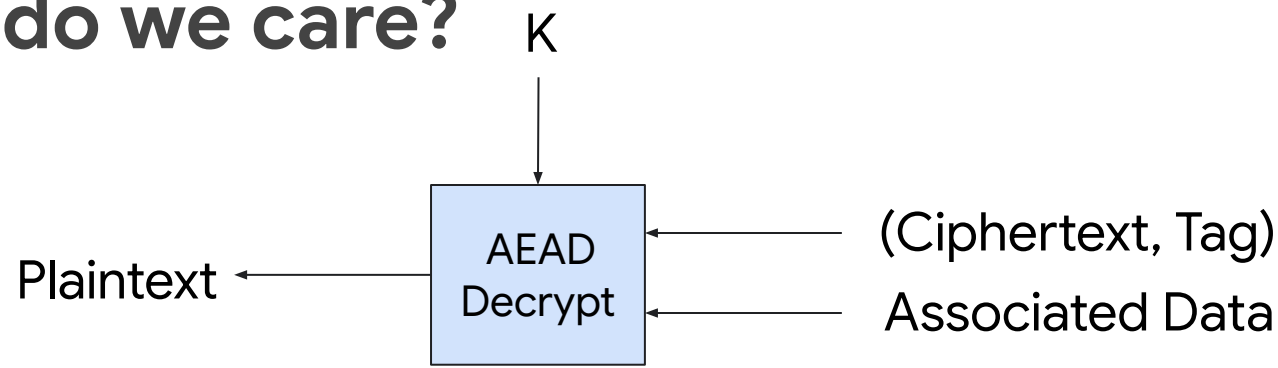
Why do we care?



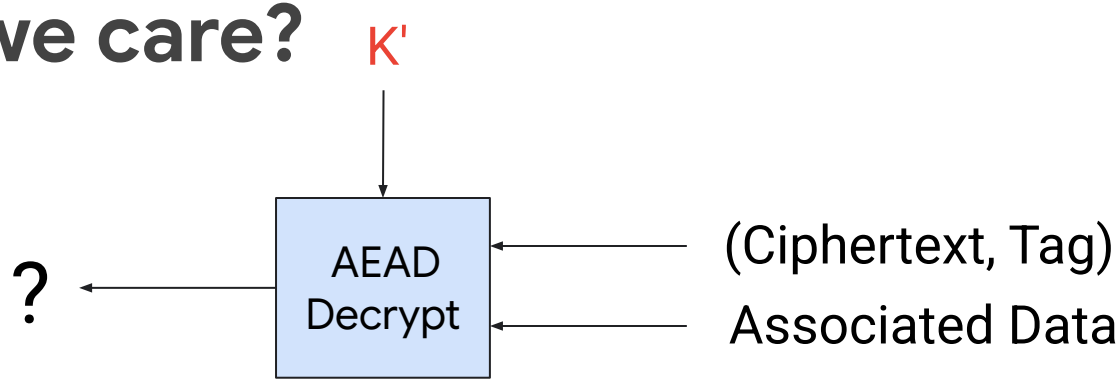
Why do we care?



Why do we care?



Why do we care?



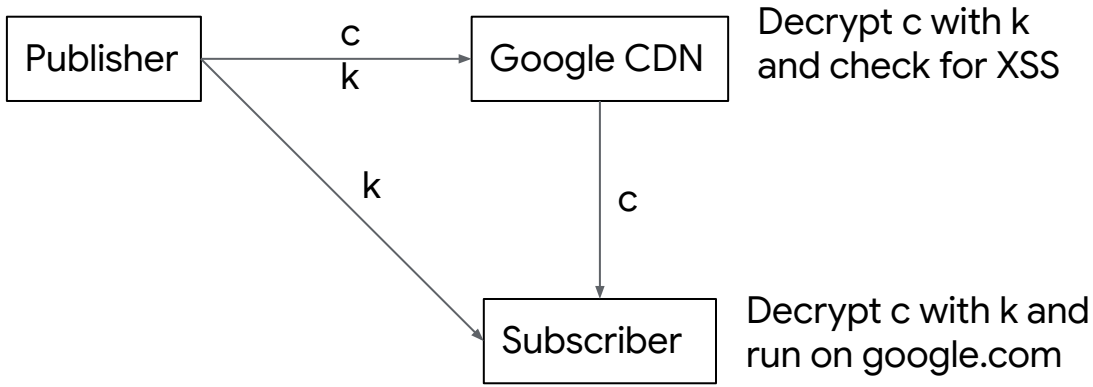
Why do we care?

Motivating example: AEAD key commitment

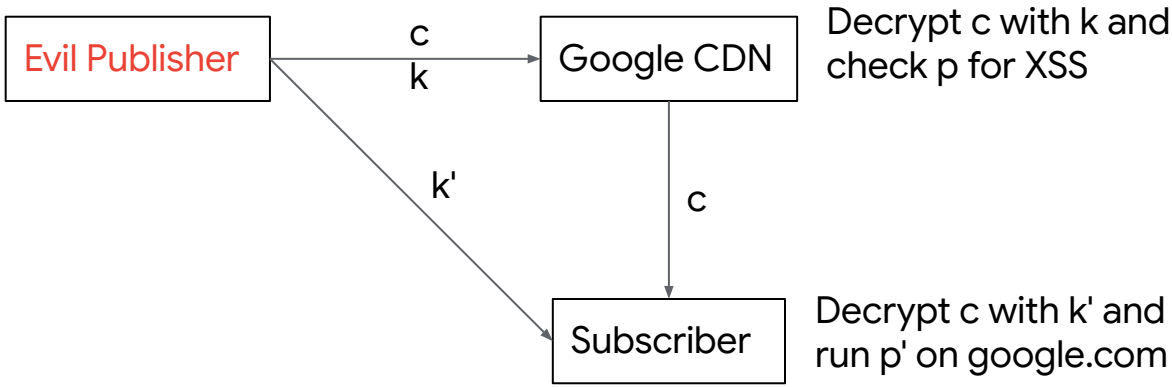
$k, k', c \leftarrow A()$
$p \leftarrow \text{Dec}(k, c)$
$p' \leftarrow \text{Dec}(k', c)$

A wins if $p \neq \perp$, $p' \neq \perp$ and $p \neq p'$

Why do we care?



Why do we care?



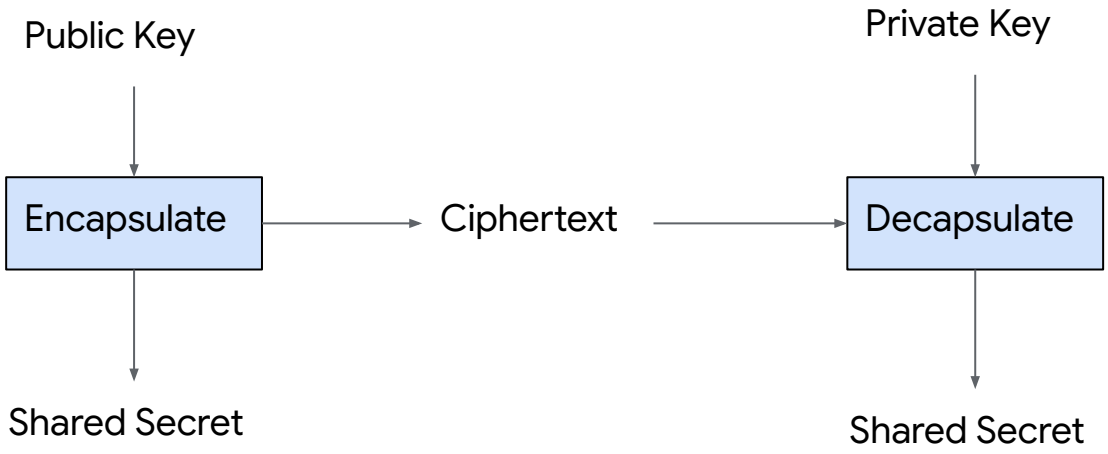
Why do we care?

AES-GCM is not key committing.

The attacker can create a winning ciphertext by solving a linear equation.

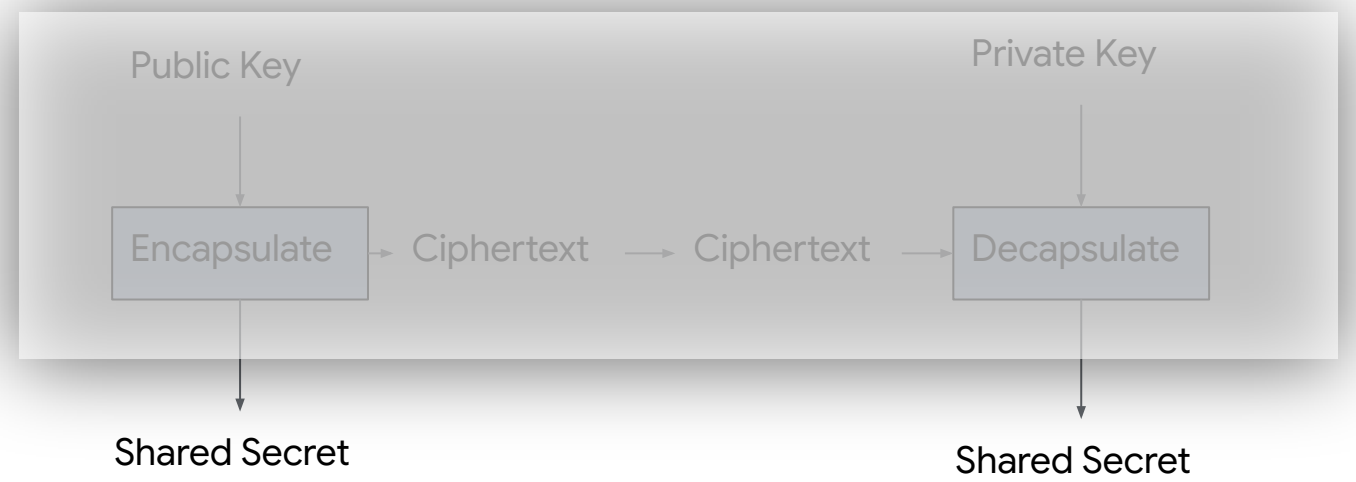
Attack has partially chosen plaintext.

Keeping Up with the KEMs

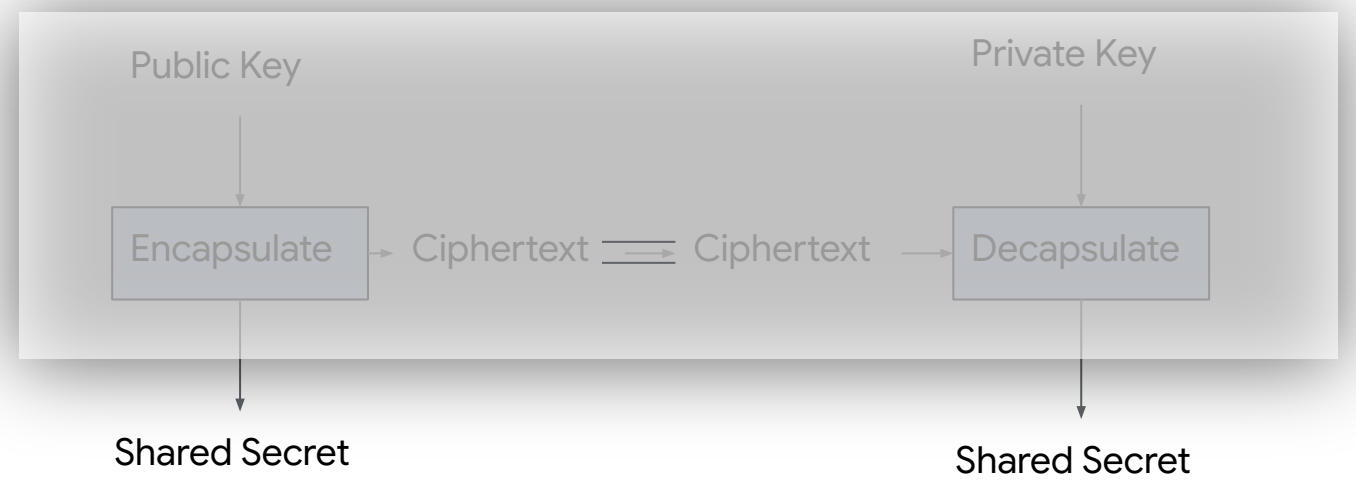


Keeping Up with the KEMs

Keeping Up with the KEMs

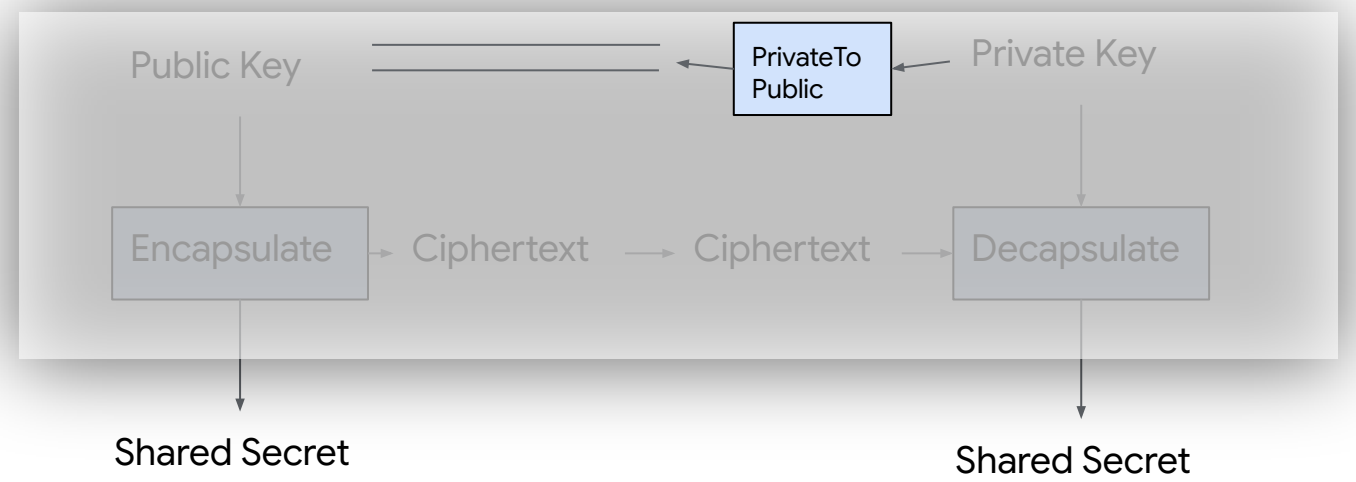


Keeping Up with the KEMs



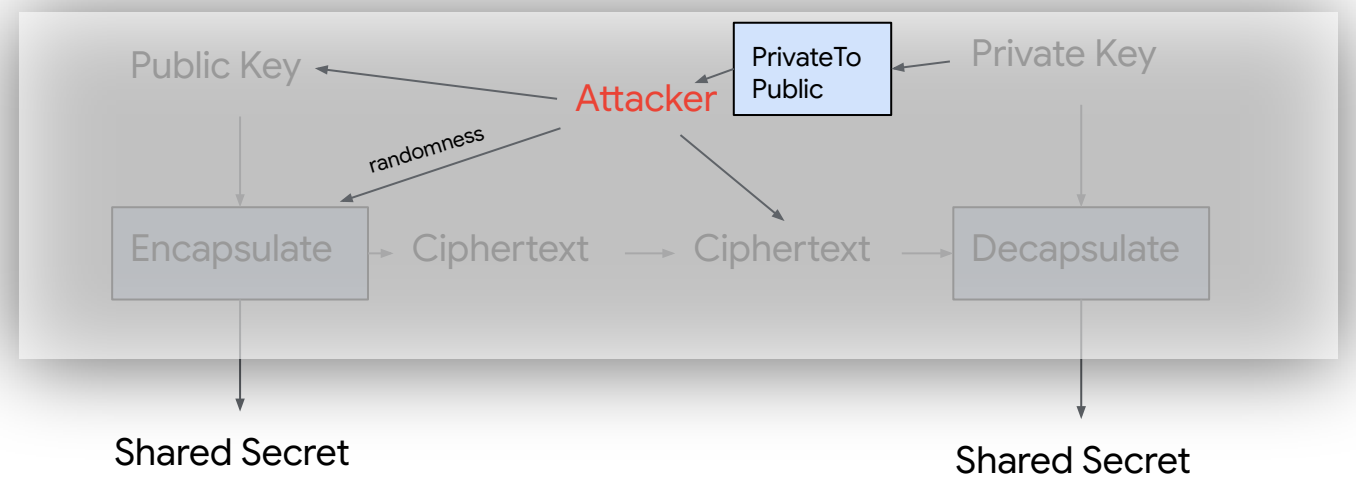
BIND-K-CT

Keeping Up with the KEMs



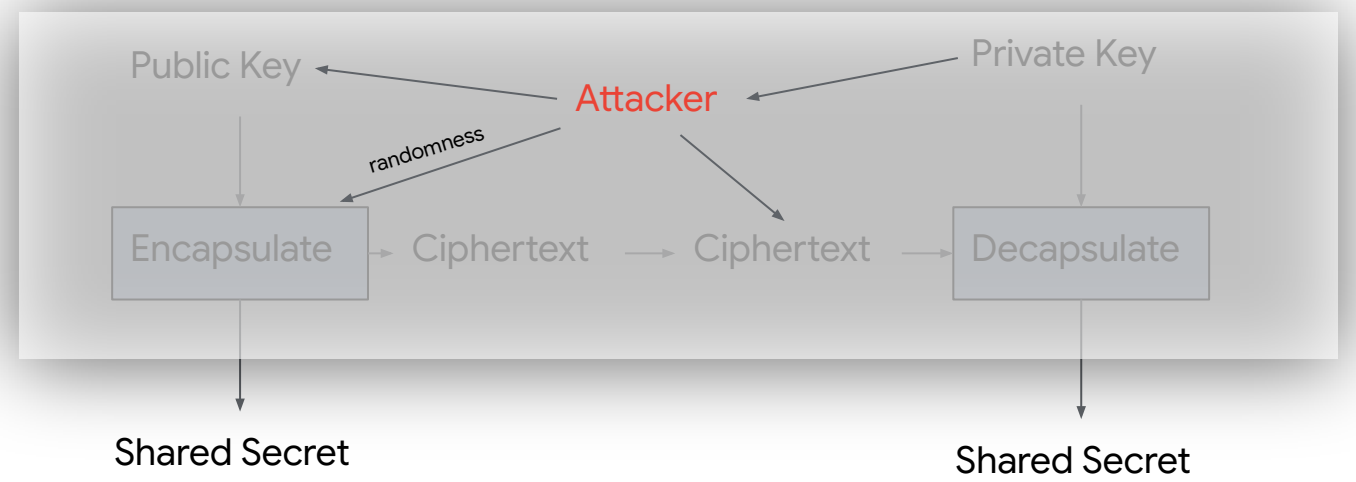
BIND-K-PK

Keeping Up with the KEMs



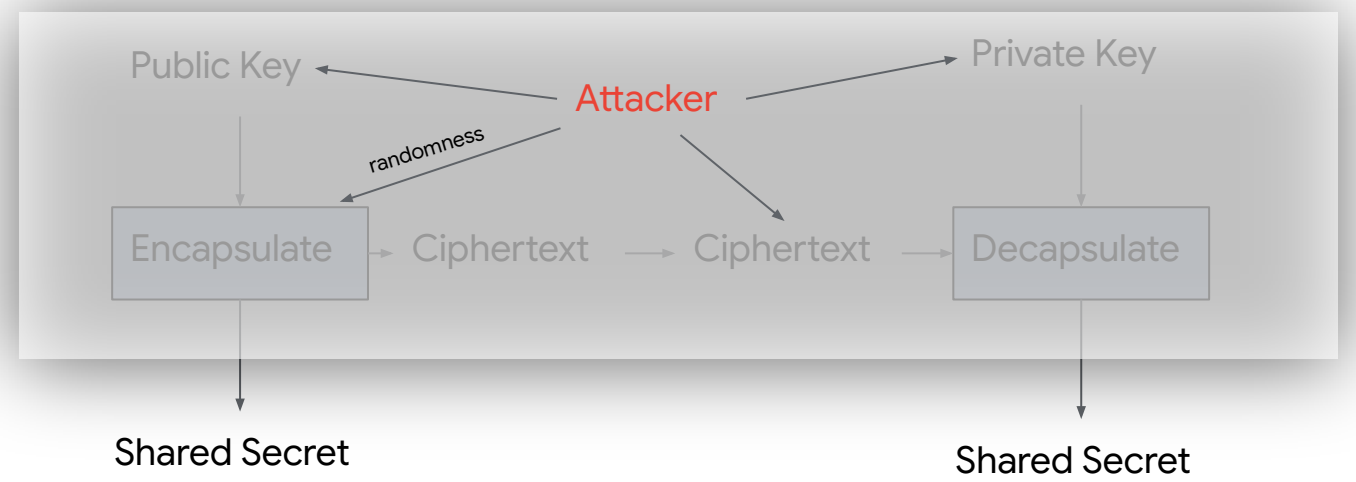
HON-BIND

Keeping Up with the KEMs



LEAK-BIND

Keeping Up with the KEMs



MAL-BIND

Unbindable Kemmy Schmidt

Does ML-KEM have these properties?

Yes for HON-BIND and LEAK-BIND

No for MAL-BIND-K-PK and MAL-BIND-K-CT

How: Malformed private key.

Wrong cached public key hash for MAL-BIND-K-PK violation

Non-unique rejection secret for MAL-BIND-K-CT violation

Unbindable Kemmy Schmidt

Does it matter?

Probably not.

Unbindable Kemmy Schmidt

Does it matter?

Probably not.

ECIES-KEM in practice fails to be HON-BIND-K-PK.
RSA-KEM fails to be HON-BIND-K-PK and HON-BIND-K-CT.

Unbindable Kemmy Schmidt

Does it matter?

Probably not.

ECIES-KEM in practice often fails to be HON-BIND-K-PK.
RSA-KEM fails to be HON-BIND-K-PK and HON-BIND-K-CT.

One can prove that ML-KEM's misbinding properties only depend on the *serialization of the private key*.

ML-KEM with private key serialized as a seed is both MAL-BIND-K-CT and MAL-BIND-K-PK.

Seed formats are a win/win

Seed formats are a win/win

Small sizes

Seed formats are a win/win

Small sizes

Performant expansion (at least for ML-KEM)

Seed formats are a win/win

Small sizes

Performant expansion (at least for ML-KEM)

Can get the strongest binding properties for free!

Seed formats are a win/win

Small sizes

Performant expansion (at least for ML-KEM)

Can get the strongest binding properties for free!

Works nicely for building hybrid KEMs

Hybrid KEMs

Hybrid KEMs are more than ‘combiners’

As seen, KEM security properties rely on keygen, encaps, and decaps

Not just the ‘combiner’

Different designs for keygen, encaps, decaps affect IND-CCA, binding properties

Can make the difference between ‘swiss army knife’ and ‘beware juggling that chainsaw’

X-Wing style

5.2. Key generation

An X-Wing keypair (decapsulation key, encapsulation key) is generated as follows.

```
def expandDecapsulationKey(sk):
  expanded = SHAKE256(sk, 96)
  (pk_M, sk_M) = ML-KEM-768.KeyGen_internal(expanded[0:32], expanded[32:64])
  sk_X = expanded[64:96]
  pk_X = X25519(sk_X, X25519_BASE)
  return (sk_M, sk_X, pk_M, pk_X)

def GenerateKeyPair():
  sk = random(32)
  (sk_M, sk_X, pk_M, pk_X) = expandDecapsulationKey(sk)
  return sk, concat(pk_M, pk_X)
```

`GenerateKeyPair()` returns the 32 byte secret decapsulation key `sk` and the 1216 byte encapsulation key `pk`.

Here and in the balance of the document for clarity we use the `M` and `X`subscripts for ML-KEM-768 and X25519 components respectively.

5.2.1. Key derivation

For testing, it is convenient to have a deterministic version of key generation. An X-Wing implementation MAY provide the following derandomized variant of key generation.

```
def GenerateKeyPairDerand(sk):
  sk_M, sk_X, pk_M, pk_X = expandDecapsulationKey(sk)
  return sk, concat(pk_M, pk_X)
```

CFRG Hybrid KEMs

6.1.1. Key generation

`QSF-SHA3-256-ML-KEM-768-P-256` KeyGen works as follows.

```
def expandDecapsulationKey(sk):
  expanded = SHAKE256(sk, 112)
  (pq_PK, pq_SK) = ML-KEM-768.KeyGen_internal(expanded[0:32], expanded[32:64])
  trad_SK = P-256.ScalarFromBytes(expanded[64:112])
  trad_PK = P-256.SerializeElement(P-256.ScalarMultBase(trad_SK))
  return (pq_SK, trad_SK, pq_PK, trad_PK)

def KeyGen():
  sk = random(32)
  (pq_SK, trad_SK, pq_PK, trad_PK) = expandDecapsulationKey(sk)
  return sk, concat(pq_PK, trad_PK)
```

Similarly, `QSF-SHA3-256-ML-KEM-768-P-256` DeriveKey works as follows:

```
def DeriveKey(seed):
  (pq_SK, trad_SK, pq_PK, trad_PK) = expandDecapsulationKey(seed)
  return sk, concat(pq_PK, trad_PK)
```

LAMPS Composite KEMs

4.1. Key Generation

To generate a new keypair for Composite schemes, the `KeyGen() -> (pk, sk)` function is used. The KeyGen() function calls the two key generation functions of the component algorithms for the Composite keypair in no particular order. Multi-process or multi-threaded applications might choose to execute the key generation functions in parallel for better key generation performance.

The following process is used to generate composite keypair values:

```
KeyGen() -> (pk, sk)

Explicit Inputs:
     None

Implicit Input:
  ML-KEM      A placeholder for the specific ML-KEM algorithm and
              parameter set to use, for example, could be "ML-KEM-65".

  Trad        A placeholder for the specific traditional algorithm and
              parameter set to use, for example "RSA-OAEP"
              or "X25519".

Output:
  (pk, sk)  The composite keypair.

Key Generation Process:

  1. Generate componint keys

    (mlkemPK, mlkemSK) = ML-KEM.KeyGen()
    (tradPK, tradSK)   = Trad.KeyGen()

  2. Check for component key gen failure
    if NOT (mlkemPK, mlkemSK) or NOT (tradPK, tradSK):
      output "Key generation error"

  3. Encode the component keys into composite structures

    pk = CompositeKEMPublicKey(mlkemPK, tradPK)
    sk = CompositeKEMPrivateKey(mlkemSK, tradSK)

  4. Output the composite keys

    return (pk, sk)
```

Figure 2: Composite KeyGen(pk, sk)

SP 800-227 ipd construction

3. **Construct a composite key-generation algorithm.** When a parameter set $p = (p_1, p_2)$ is input, the algorithm $\mathcal{C}[\Pi_1, \Pi_2]$.KeyGen will perform:

    1. $(\mathsf{ek}_1, \mathsf{dk}_1) \leftarrow \Pi_1.\mathsf{KeyGen}(p_1)$.
    2. $(\mathsf{ek}_2, \mathsf{dk}_2) \leftarrow \Pi_2.\mathsf{KeyGen}(p_2)$.
    3. Output composite encapsulation key $\mathsf{ek}_1 \| \mathsf{ek}_2$.
    4. Output composite decapsulation key $\mathsf{dk}_1 \| \mathsf{dk}_2$.

Hybrid KEMs Keygen

Hybrid KEMs Keygen

Root seed is smallest and supports strongest binding properties for hybrid KEMs as well as component KEMs

Hybrid KEMs Keygen

Root seed is smallest and supports strongest binding properties for hybrid KEMs as well as component KEMs

Eliminates combining component keys to produce hybrid key

Hybrid KEMs Keygen

Root seed is smallest and supports strongest binding properties for hybrid KEMs as well as component KEMs

Eliminates combining component keys to produce hybrid key

Guarantees well-formed and trusted hybrid KEM keys

Hybrid KEMs Keygen

Root seed is smallest and supports strongest binding properties for hybrid KEMs as well as component KEMs

Eliminates combining component keys to produce hybrid key

Guarantees well-formed and trusted hybrid KEM keys

For FIPS, requires the promised SP 800-133 update (🤞)

Hybrid KEMs Keygen

Root seed is smallest and supports strongest binding properties for hybrid KEMs as well as component KEMs

Eliminates combining component keys to produce hybrid key

Guarantees well-formed and trusted hybrid KEM keys

For FIPS, requires the promised SP 800-133 update (🤞)

Guidance on expanding from seed in/out of module is needed

Subject
[lamps] Request to Extend IETF WGLC for ML-KEM and ML-DSA private key format Specifications
[lamps] Re: Request to Extend IETF WGLC for ML-KEM and ML-DSA private key format Specifications
[lamps] Re: [EXTERNAL] Re: Request to Extend IETF WGLC for ML-KEM and ML-DSA private key format Specifications
[lamps] Re: [EXTERNAL] Re: Request to Extend IETF WGLC for ML-KEM and ML-DSA private key format Specifications
[lamps] Re: [EXTERNAL] Re: Request to Extend IETF WGLC for ML-KEM and ML-DSA private key format Specifications
[lamps] Re: [EXTERNAL] Re: Request to Extend IETF WGLC for ML-KEM and ML-DSA private key format Specifications
[lamps] Re: [EXTERNAL] Re: Request to Extend IETF WGLC for ML-KEM and ML-DSA private key format Specifications

Subject

[lamps] Request to Extend IETF WGLC for ML-KEM and ML-DSA private key format Specifications

[lamps] Re: Request to Extend IETF WGLC for ML-KEM and ML-DSA private key format Specifications

[lamps] Re: [EXTERNAL] Re: Request to Extend IETF WGLC for ML-KEM and ML-DSA private key format Specifications

[lamps] Re: [EXTERNAL] Re: Request to Extend IETF WGLC for ML-KEM and ML-DSA private key format Specifications

[lamps] Re: [EXTERNAL] Re: Request to Extend IETF WGLC for ML-KEM and ML-DSA private key format Specifications

[lamps] Re: [EXTERNAL] Re: Request to Extend IETF WGLC for ML-KEM and ML-DSA private key format Specifications

[lamps] Re: [EXTERNAL] Re: Request to Extend IETF WGLC for ML-KEM and ML-DSA private key format Specifications

137 Messages

Key Takeaways

More than IND-CCA2

The properties of a cryptographic primitive can sometimes be subtly more complex than just IND-CCA.

Seed formats win

Smallest, best security properties, efficient.

Document updates and guidance needed

Allow FIPS KDF to product seed bytes; guidance on seed format as equal, not ‘alternate’ to expanded, and allowed processing in modules, needed.

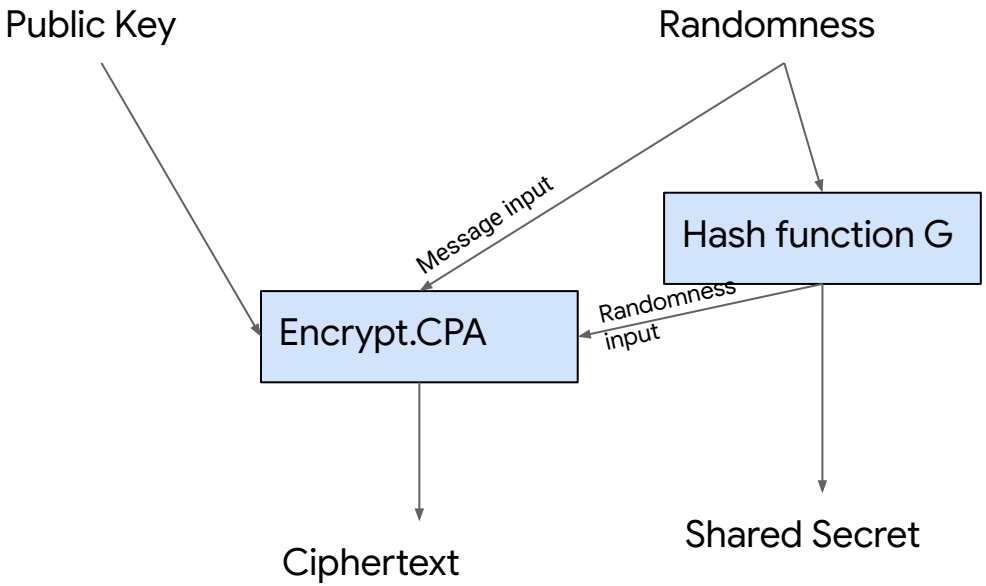
Thank you

Sophie Schmieg
Senior Staff Cryptography Engineer, Google
sschmieg@google.com

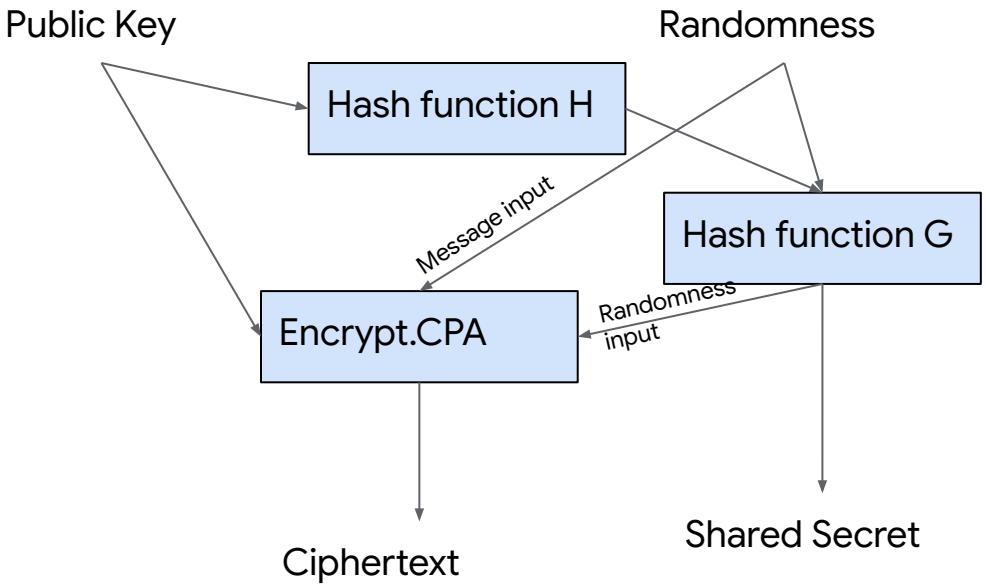
Deirdre Connolly
Senior Staff Standardization Research Engineer, SandboxAQ
durumcrustulum@gmail.com

Appendix

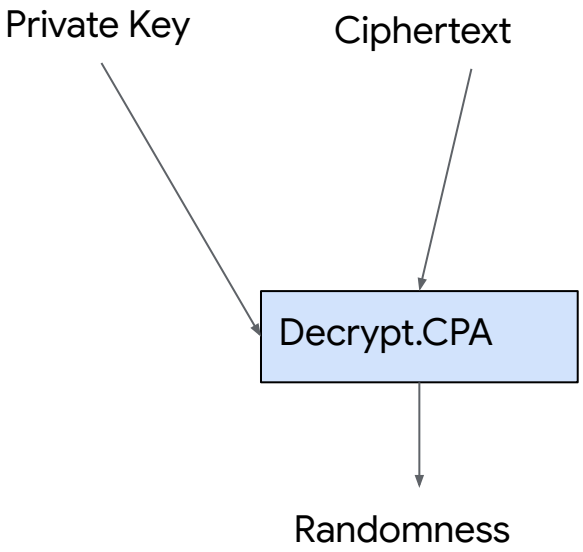
Unbindable Kemmy Schmidt



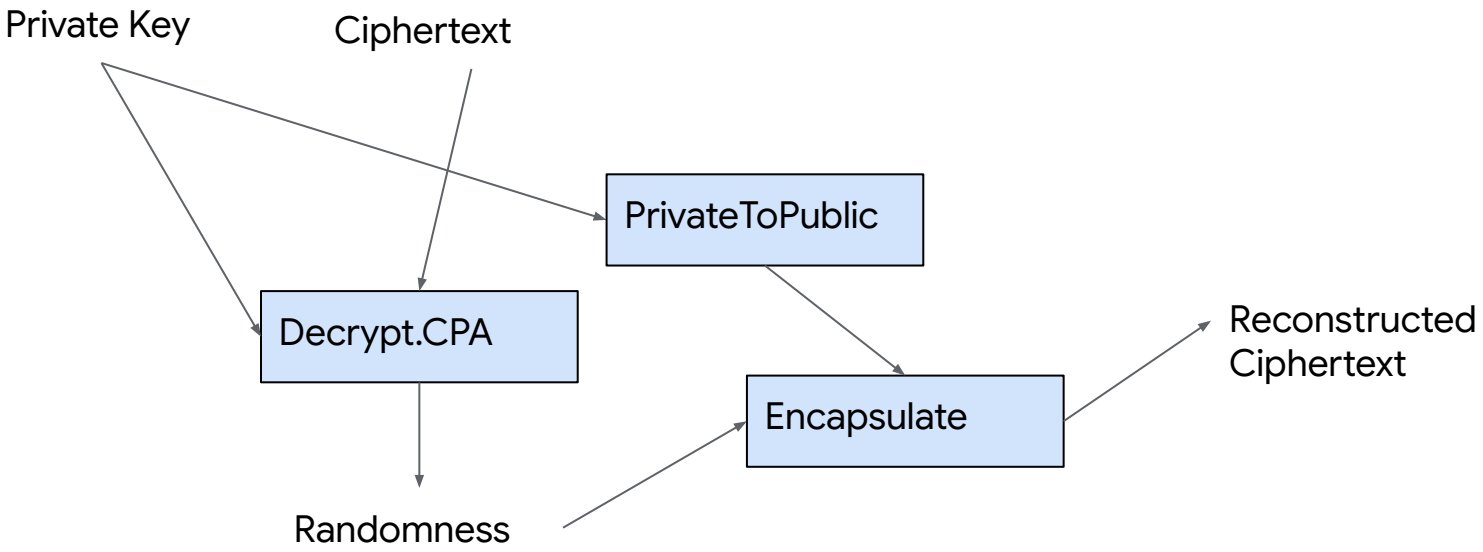
Unbindable Kemmy Schmidt



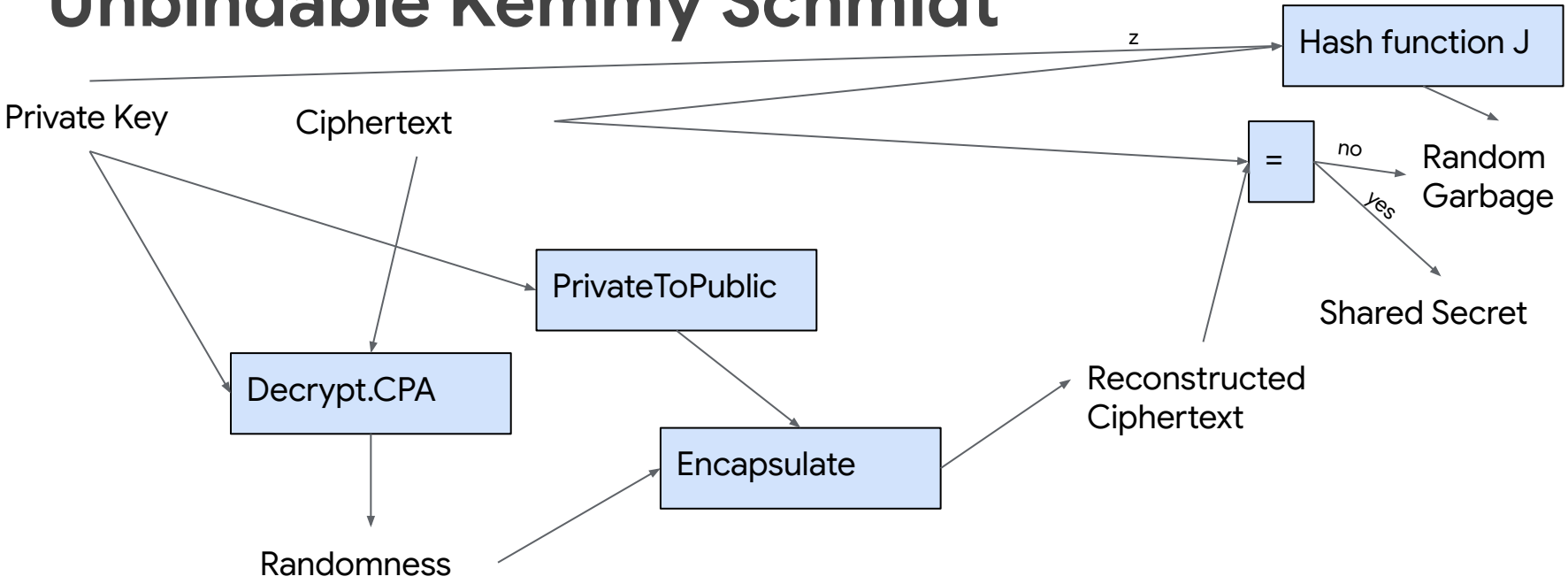
Unbindable Kemmy Schmidt



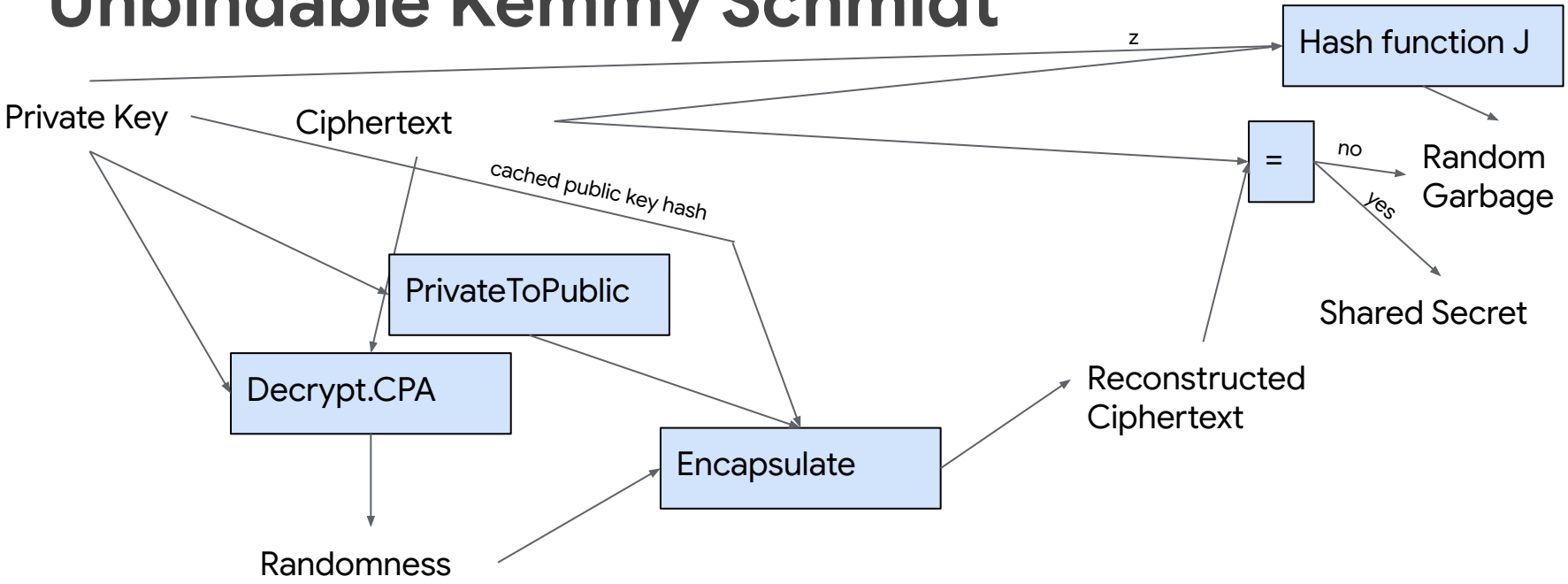
Unbindable Kemmy Schmidt



Unbindable Kemmy Schmidt



Unbindable Kemmy Schmidt

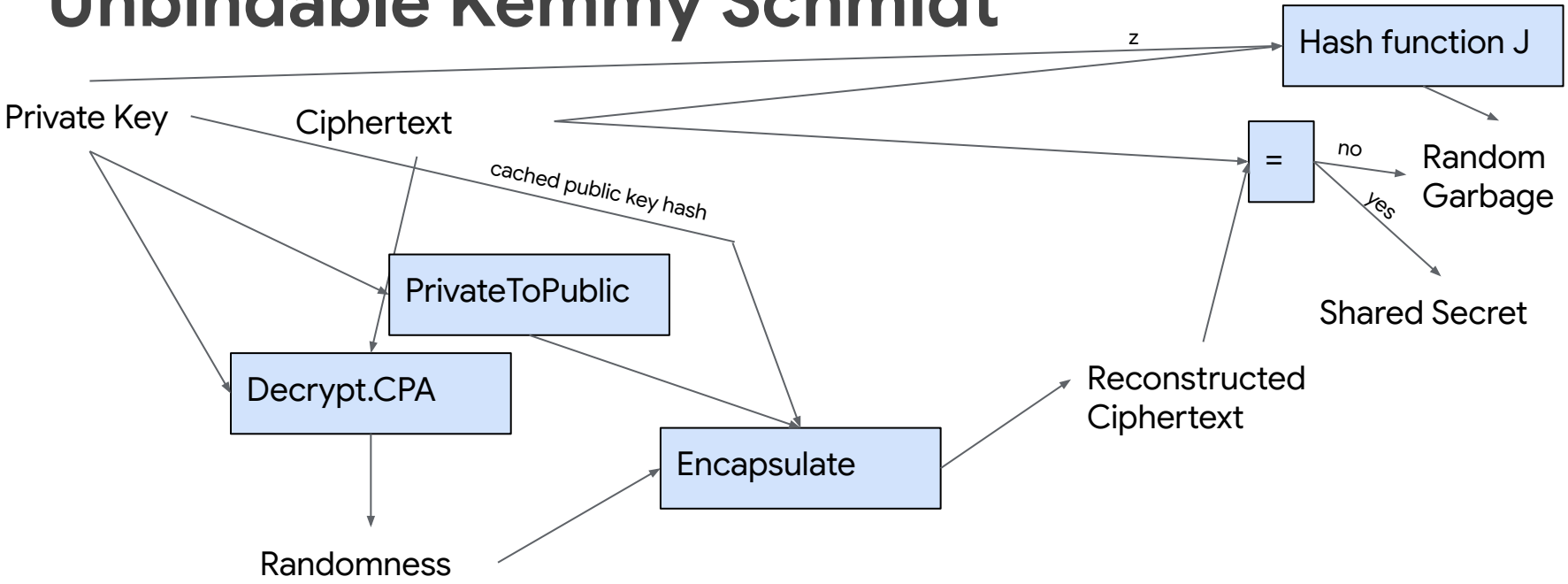


Unbindable Kemmy Schmidt

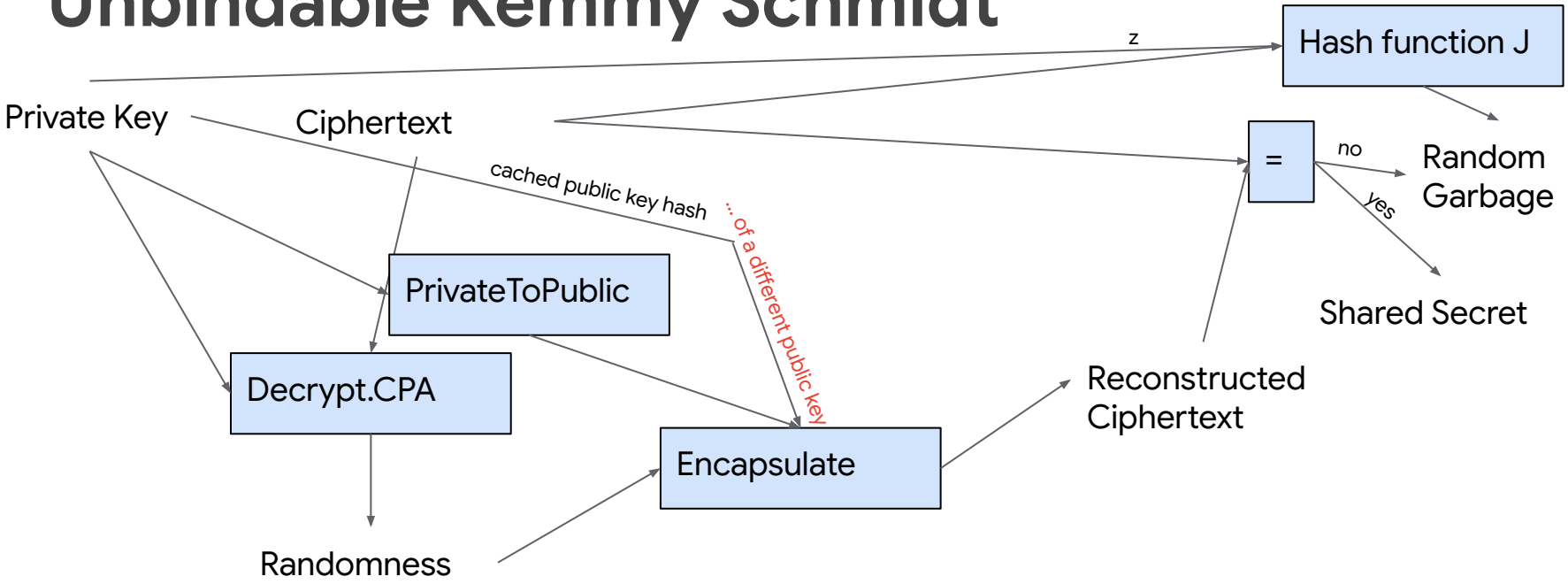
Private Key Format for Kyber

- CPA Private Key
- Public Key
- Cache of public key hash (h)
- FO rejection secret (z)

Unbindable Kemmy Schmidt



Unbindable Kemmy Schmidt



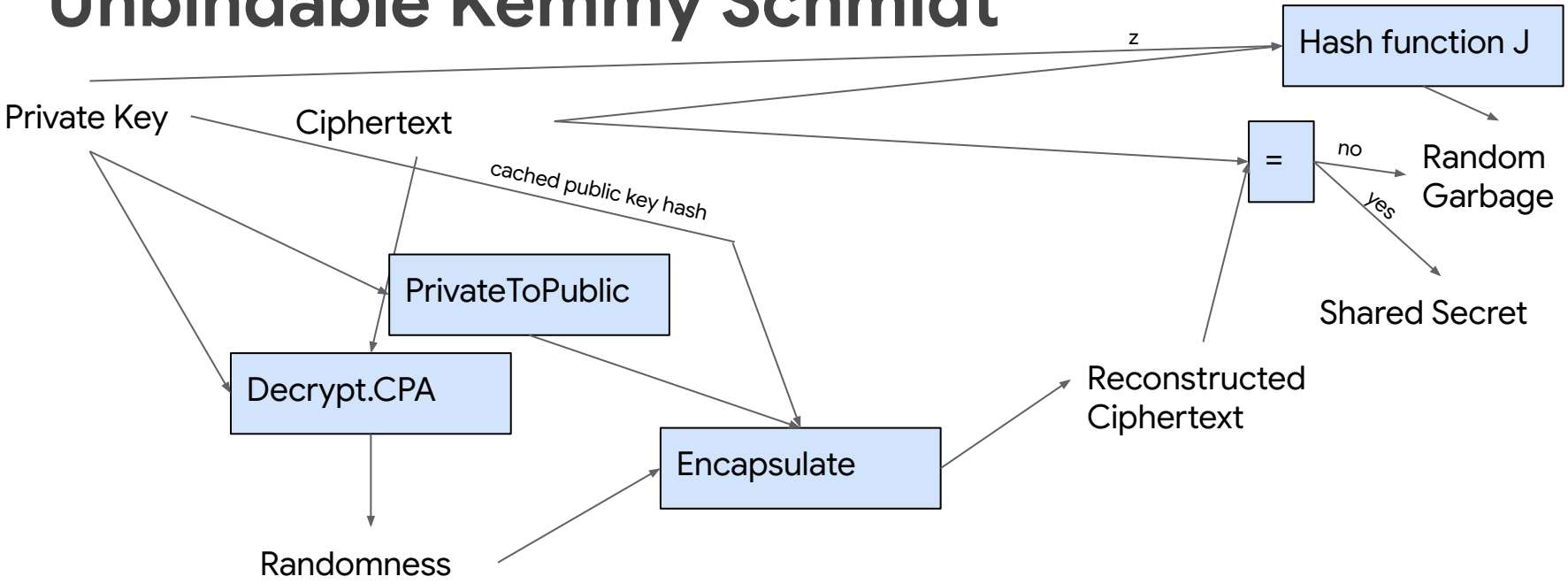
Unbindable Kemmy Schmidt

Fujisaki-Okamoto transform with the attacker prepared data:

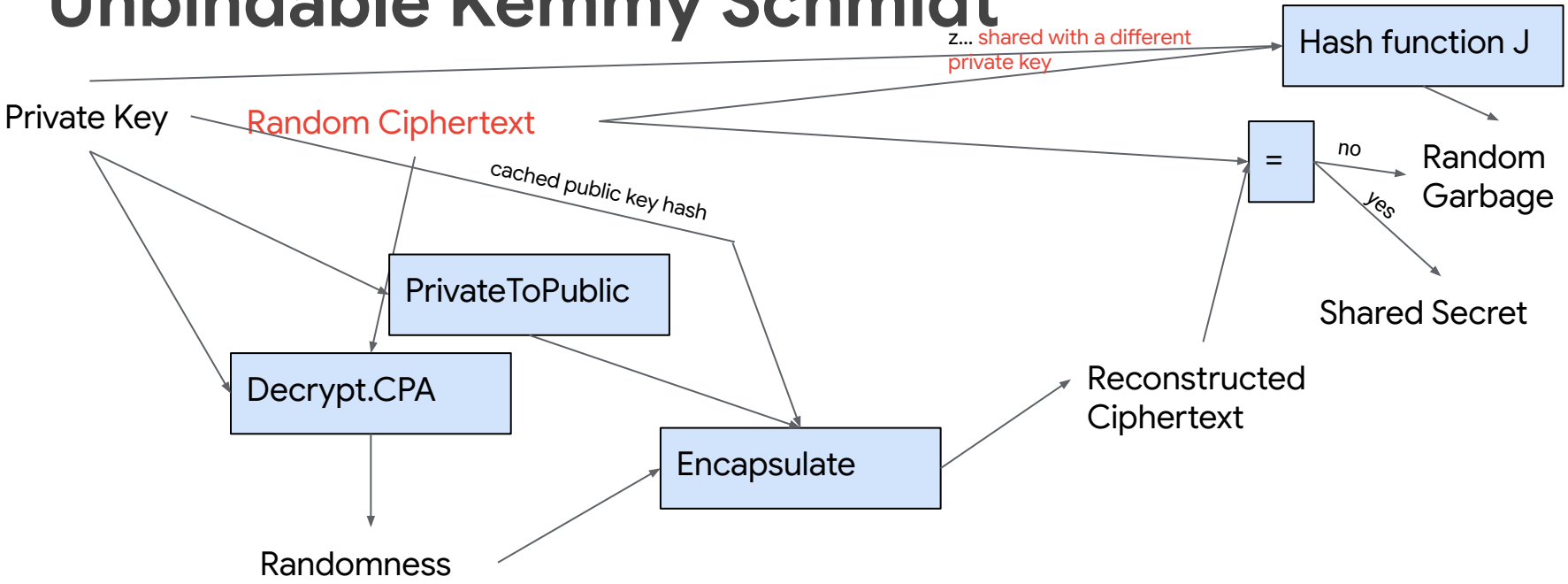
Enc(pk_0, m_0):
$K, r \leftarrow G(m_0 \,||\, H(pk_0))$
return K, Encrypt.CPA(pk_0, m_0; r_0)

Dec(sk_1, c_1):
$m' \leftarrow$ Decrypt.CPA(sk, c)
$K, r \leftarrow G(m' \,||\, h_0)$ // Note that $sk_1.h = h_0 = H(pk_0)$
$c' \leftarrow$ Encrypt.CPA($sk_1.pk$, m'; r)
$c = c'$ so return K

Unbindable Kemmy Schmidt



Unbindable Kemmy Schmidt



Unbindable Kemmy Schmidt

Fujisaki-Okamoto transform with the attacker prepared data:

Dec(sk_0, c):
m_0' ← Decrypt.CPA(sk_0, c)
K_0, r_0 ← G(m_0' || $sk_0.h$)
c_0' ← Encrypt.CPA($sk_0.pk$, m'; r)
$c \neq c_0'$ so return J(z || c) // Note that $sk_0.z = z$

Dec(sk_1, c):
m_1' ← Decrypt.CPA(sk_1, c)
K_1, r_1 ← G(m_1' || $sk_1.h$)
c_1' ← Encrypt.CPA($sk_1.pk$, m_1'; r_1)
$c \neq c_1'$ so return J(z || c) // Note that $sk_1.z = z$

Unbindable Kemmy Schmidt

Fujisaki-Okamoto transform for KEMs as used in ML-KEM Draft:

Enc(*pk*):
m ⟵ Rng()
K, *r* ⟵ G(*m* || H(*pk*))
return *K*, Encrypt.CPA(*pk*, *m*; *r*)

Dec(*sk*, *c*):
m' ⟵ Decrypt.CPA(*sk*, *c*)
K, *r* ⟵ G(*m*' || *sk.h*)
c' ⟵ Encrypt.CPA(*sk.pk*, *m*'; *r*)
if *c* = *c*' return *K*
else return J(*sk.z* || *c*)

Unbindable Kemmy Schmidt

MAL-BIND-K-PK and MAL-BIND-K-CT counterexample

$A()$:
$sk_0 \leftarrow (s_0, t_0, \rho_0, h_0, z_0) \leftarrow \text{KeyGen}()$
$pk_0 \leftarrow (t_0, \rho_0)$
$(s_1, t_1, \rho_1, h_1, z_1) \leftarrow \text{KeyGen}()$
$sk_1 \leftarrow (s_1, t_1, \rho_1, h_0, z_1)$
$m_0 \leftarrow \text{Rng}(32)$
$r \leftarrow G(m_0 \,||\, h_0)$
$c_1 \leftarrow \text{Enc}(pk_1, m_0; r)$
return (pk_0, sk_1, m_0, c_0)

Unbindable Kemmy Schmidt

MAL-BIND-K-PK counterexample

$A()$:
z = Rng(32)
$(s_0, t_0, \rho_0, h_0, z_0) \leftarrow$ KeyGen()
$(s_1, t_1, \rho_1, h_1, z_1) \leftarrow$ KeyGen()
$sk_0 \leftarrow (s_0, t_0, \rho_0, h_0, z)$
$sk_1 \leftarrow (s_1, t_1, \rho_1, h_1, z)$
$c \leftarrow$ Rng(len_c)
return (sk_0, sk_1, c, c)

Unbindable Kemmy Schmidt

Attack game for MAL-BIND-K-CT/MAL-BIND-K-PK:

$pk_0, sk_1, r_0, ct_1 \leftarrow A()$
$pk_1 \leftarrow \text{PrivateToPublic}(sk_1)$
$k_0, ct_0 \leftarrow \text{Encaps}(pk_0, r_0)$
$k_1 \leftarrow \text{Decaps}(sk_1, ct_1)$

Attacker wins if:

- $k_0 = k_1$, but $ct_0 \neq ct_1$ (For MAL-BIND-K-CT)
- $k_0 = k_1$, but $pk_0 \neq pk_1$ (For MAL-BIND-K-PK)