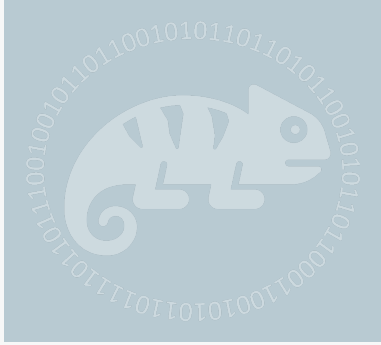


Hybrid Obfuscated Key Exchange and KEMs

Felix Günther
IBM Research – Zurich

Michael Rosenberg
Cloudflare

Douglas Stebila
University of Waterloo

Shannon Veitch
ETH Zurich

CRYPTO 2025. Santa Barbara, CA, United States.

Obfuscated key exchange

Obfuscated key exchange

Some Internet protocols **hide metadata**:

Obfuscated key exchange

Some Internet protocols **hide metadata**:

- TLS 1.3 Encrypted Client Hello, QUIC, obfs4, Shadowsocks, ...

Obfuscated key exchange

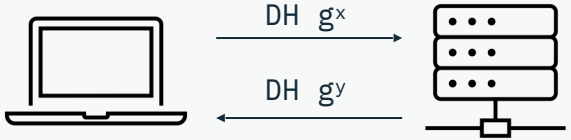
Some Internet protocols **hide metadata**:

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

Obfuscated key exchange

Some Internet protocols **hide metadata**:

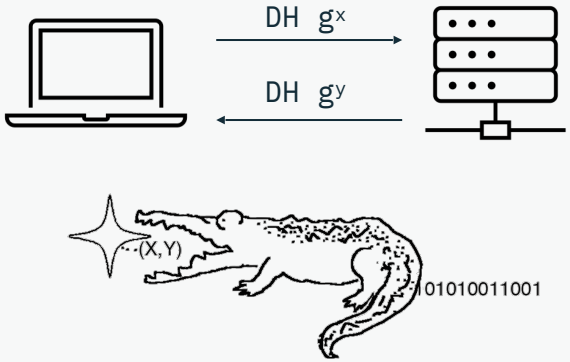
- TLS 1.3 Encrypted Client Hello, QUIC, obfs4, Shadowsocks, …
- "Fully encrypted" protocols, with **obfuscated key exchange**



Obfuscated key exchange

Some Internet protocols **hide metadata**:

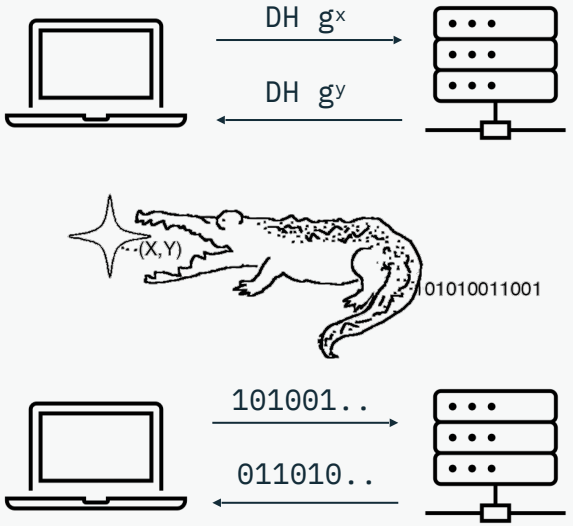
- TLS 1.3 Encrypted Client Hello, QUIC, obfs4, Shadowsocks, …
- "Fully encrypted" protocols, with **obfuscated key exchange**



Obfuscated key exchange

Some Internet protocols **hide metadata**:

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



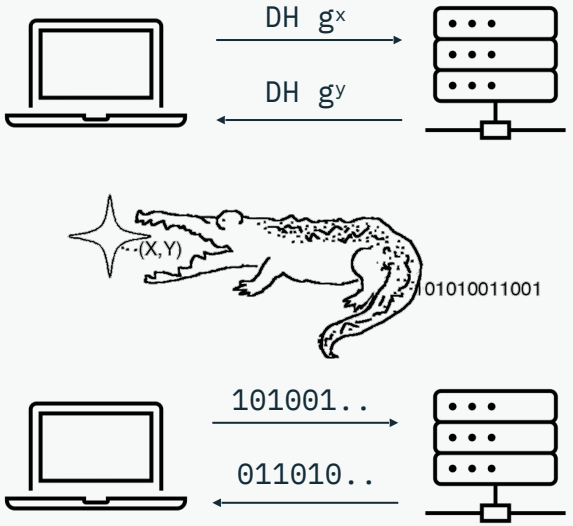
Obfuscated key exchange

Some Internet protocols **hide metadata**:

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

Classical: obfs4

PQ: pqobfs (via *obfuscated KEMs*)



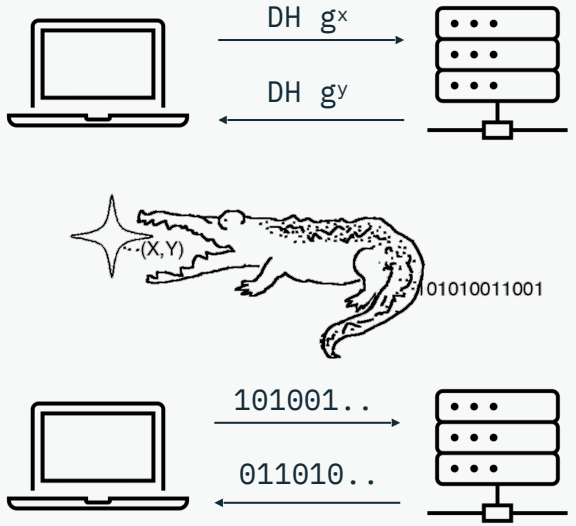
Obfuscated key exchange

Some Internet protocols **hide metadata**:

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

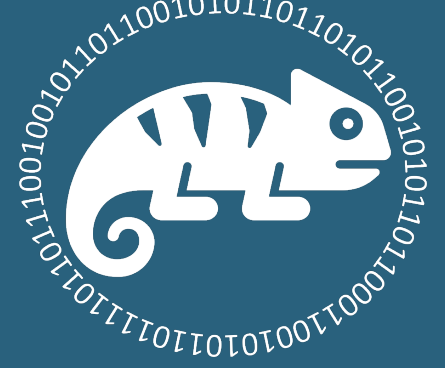
Classical: obfs4

PQ: pqobfs (via *obfuscated KEMs*)



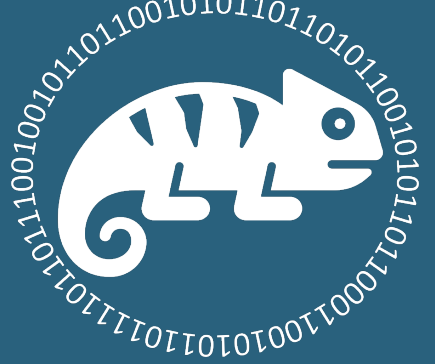
But what about hybrid?

Our contributions



Our contributions

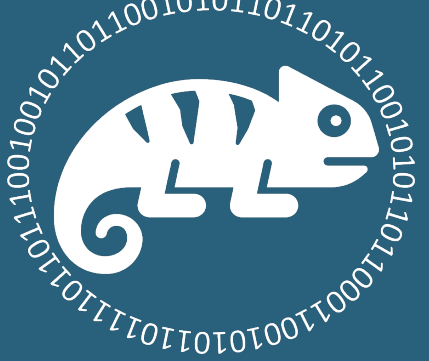
We build **hybrid obfuscated key exchange**



Our contributions

We build **hybrid obfuscated key exchange**

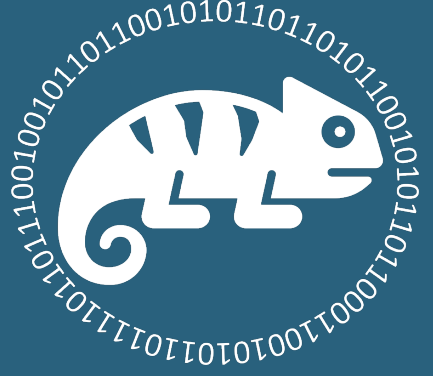
1. Define an obfuscated KEM (OKEM) combiner



Our contributions

We build **hybrid obfuscated key exchange**

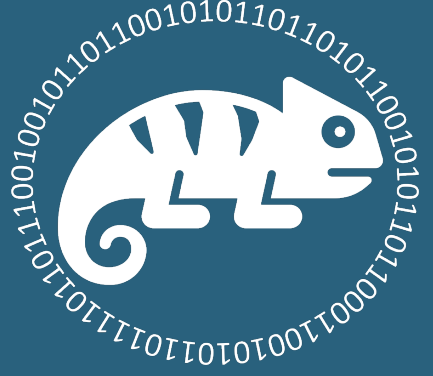
1. Define an obfuscated KEM (OKEM) combiner
2. Define Drivel, a new KEX protocol



Our contributions

We build **hybrid obfuscated key exchange**

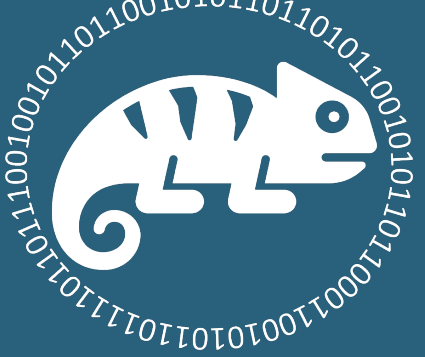
1. Define an obfuscated KEM (OKEM) combiner
2. Define Drivel, a new KEX protocol
3. Bonus: Hybrid PAKE from OKEM



Our contributions

We build **hybrid obfuscated key exchange**

1. Define an obfuscated KEM (OKEM) combiner
2. Define Drivel, a new KEX protocol
3. Bonus: Hybrid PAKE from OKEM



Obfuscated key encaps. mechanism (OKEM)

Obfuscated key encaps. mechanism (OKEM)

$\mathsf{KeyGen}() \mapsto (\mathsf{sk}, \mathsf{pk})$

Obfuscated key encaps. mechanism (OKEM)

KeyGen() $\mapsto$ (sk, pk)

Encap(pk) $\mapsto$ (c, K)

Obfuscated key encaps. mechanism (OKEM)

$\mathsf{KeyGen}() \mapsto (\mathsf{sk}, \mathsf{pk})$

$\mathsf{Encap}(\mathsf{pk}) \mapsto (c, K)$

$\mathsf{Decap}(\mathsf{sk}, c) \mapsto K$

Obfuscated key encaps. mechanism (OKEM)

KeyGen() $\mapsto$ (sk, pk, $\hat{\text{pk}}$)

Encap(pk) $\mapsto (c, K)$

Decap(sk, c) $\mapsto K$

Obfuscated key encaps. mechanism (OKEM)

$\mathsf{KeyGen}() \mapsto (\mathsf{sk}, \mathsf{pk}, \hat{\mathsf{pk}})$

$\mathsf{Encap}(\mathsf{pk}) \mapsto (c, K)$

$\mathsf{Decap}(\mathsf{sk}, c) \mapsto K$

Obfuscated key encaps. mechanism (OKEM)

KeyGen() $\mapsto$ (sk, pk, $\hat{\text{pk}}$)

Encap(pk) $\mapsto (c, K)$

Decap(sk, c) $\mapsto K$

DecodePk($\hat{\text{pk}}$) $\mapsto$ pk

Obfuscated key encaps. mechanism (OKEM)

KeyGen() $\mapsto$ (sk, pk, $\hat{\text{pk}}$)

Encap(pk) $\mapsto (c, K)$

Decap(sk, c) $\mapsto K$

DecodePk($\hat{\text{pk}}$) $\mapsto$ pk

Obfuscated key encaps. mechanism (OKEM)

$\mathsf{KeyGen}() \mapsto (\mathsf{sk}, \mathsf{pk}, \hat{\mathsf{pk}})$

$\mathsf{Encap}(\mathsf{pk}) \mapsto (c, K)$

$\mathsf{Decap}(\mathsf{sk}, c) \mapsto K$

$\mathsf{DecodePk}(\hat{\mathsf{pk}}) \mapsto \mathsf{pk}$

Ciphertext uniformity

Cannot distinguish c from uniform, given pk

Strong variant: given sk

Obfuscated key encaps. mechanism (OKEM)

$\mathsf{KeyGen}() \mapsto (\mathsf{sk}, \mathsf{pk}, \hat{\mathsf{pk}})$

$\mathsf{Encap}(\mathsf{pk}) \mapsto (c, K)$

$\mathsf{Decap}(\mathsf{sk}, c) \mapsto K$

$\mathsf{DecodePk}(\hat{\mathsf{pk}}) \mapsto \mathsf{pk}$

Ciphertext uniformity

Cannot distinguish c from uniform, given pk

Strong variant: given sk

Public key uniformity

Cannot distinguish pk from uniform

Obfuscated key encaps. mechanism (OKEM)

$\mathsf{KeyGen}() \mapsto (\mathsf{sk}, \mathsf{pk}, \hat{\mathsf{pk}})$

$\mathsf{Encap}(\mathsf{pk}) \mapsto (c, K)$

$\mathsf{Decap}(\mathsf{sk}, c) \mapsto K$

$\mathsf{DecodePk}(\hat{\mathsf{pk}}) \mapsto \mathsf{pk}$

Ciphertext uniformity

Cannot distinguish c from uniform, given pk

Strong variant: given sk

Public key uniformity

Cannot distinguish pk from uniform

SPR-CCA, IND-CCA as normal

Obfuscated key encaps. mechanism (OKEM)

$\mathsf{KeyGen}() \mapsto (\mathsf{sk}, \mathsf{pk}, \hat{\mathsf{pk}})$

$\mathsf{Encap}(\mathsf{pk}) \mapsto (c, K)$

$\mathsf{Decap}(\mathsf{sk}, c) \mapsto K$

$\mathsf{DecodePk}(\hat{\mathsf{pk}}) \mapsto \mathsf{pk}$

Ciphertext uniformity

Cannot distinguish c from uniform, given pk

Strong variant: given sk

Public key uniformity

Cannot distinguish pk from uniform

SPR-CCA, IND-CCA as normal

Classical OKEM: **ECDH** over Ristretto w/ Elligator2 encoding

Obfuscated key encaps. mechanism (OKEM)

$\mathsf{KeyGen}() \mapsto (\mathsf{sk}, \mathsf{pk}, \hat{\mathsf{pk}})$

$\mathsf{Encap}(\mathsf{pk}) \mapsto (c, K)$

$\mathsf{Decap}(\mathsf{sk}, c) \mapsto K$

$\mathsf{DecodePk}(\hat{\mathsf{pk}}) \mapsto \mathsf{pk}$

Ciphertext uniformity

Cannot distinguish c from uniform, given pk

Strong variant: given sk

Public key uniformity

Cannot distinguish pk from uniform

SPR-CCA, IND-CCA as normal

Classical OKEM: **ECDH** over Ristretto w/ Elligator2 encoding

unconditional strong unif.

Obfuscated key encaps. mechanism (OKEM)

$\mathsf{KeyGen}() \mapsto (\mathsf{sk}, \mathsf{pk}, \hat{\mathsf{pk}})$

$\mathsf{Encap}(\mathsf{pk}) \mapsto (c, K)$

$\mathsf{Decap}(\mathsf{sk}, c) \mapsto K$

$\mathsf{DecodePk}(\hat{\mathsf{pk}}) \mapsto \mathsf{pk}$

Ciphertext uniformity

Cannot distinguish c from uniform, given pk

Strong variant: given sk

Public key uniformity

Cannot distinguish pk from uniform

SPR-CCA, IND-CCA as normal

unconditional strong unif.

Classical OKEM: **ECDH** over Ristretto w/ Elligator2 encoding

PQ OKEM: **FrodoKEM** **Saber** **ML-Kemeleon**

Obfuscated key encaps. mechanism (OKEM)

KeyGen() $\mapsto$ (sk, pk, $\hat{\text{pk}}$)

Encap(pk) $\mapsto (c, K)$

Decap(sk, c) $\mapsto K$

DecodePk($\hat{\text{pk}}$) $\mapsto$ pk

Ciphertext uniformity

Cannot distinguish c from uniform, given pk

Strong variant: given sk

Public key uniformity

Cannot distinguish pk from uniform

SPR-CCA, IND-CCA as normal

unconditional strong unif.

Classical OKEM: **ECDH** over Ristretto w/ Elligator2 encoding

PQ OKEM: **FrodoKEM**^{LWE} **Saber**^{MLWR} **ML-Kemeleon**^{MLWE}

Hybrid KEMs: The Parallel Approach

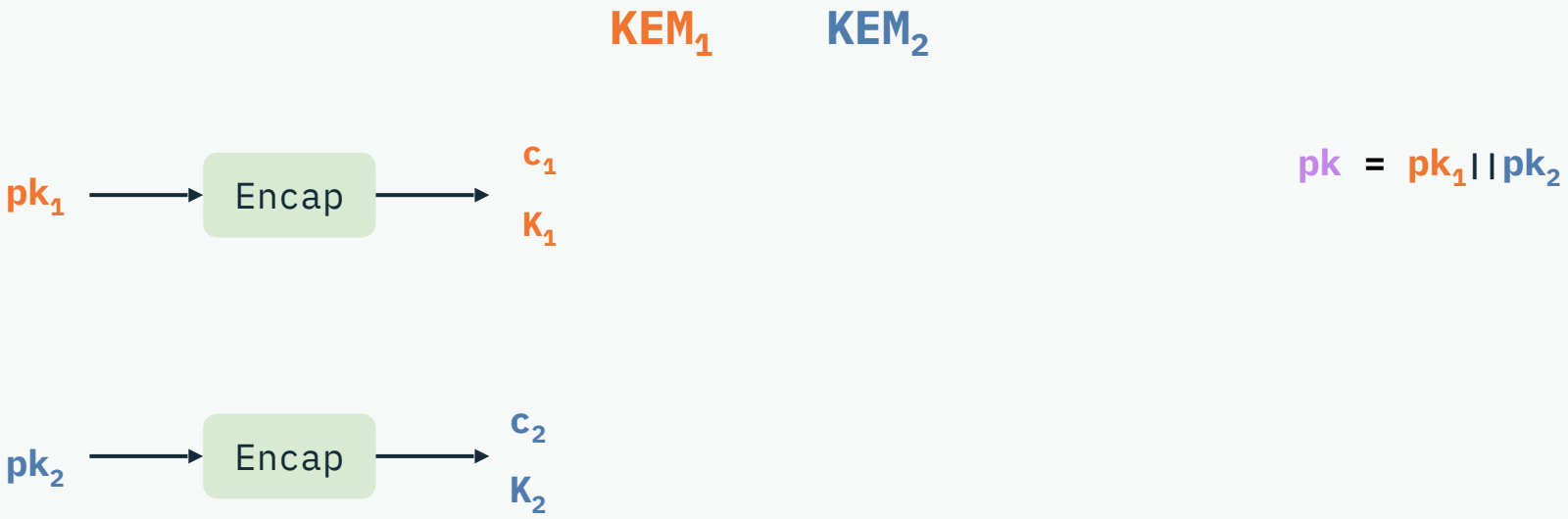
KEM_1 KEM_2

Hybrid KEMs: The Parallel Approach

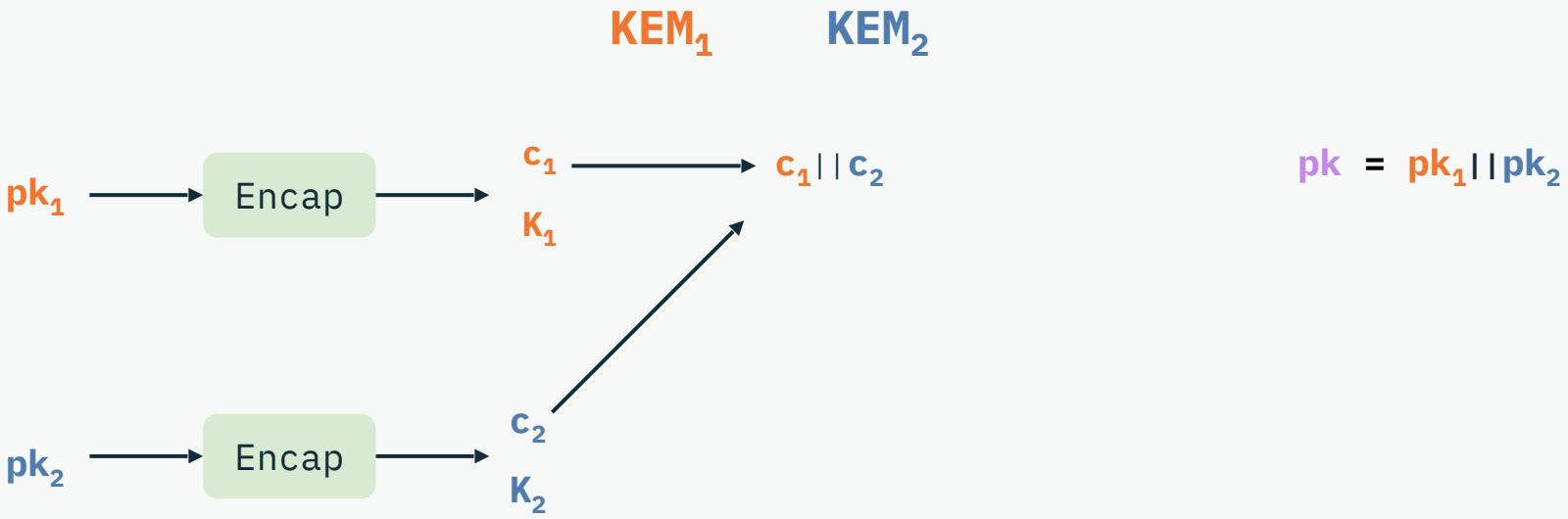
KEM_1 KEM_2

$pk = pk_1 || pk_2$

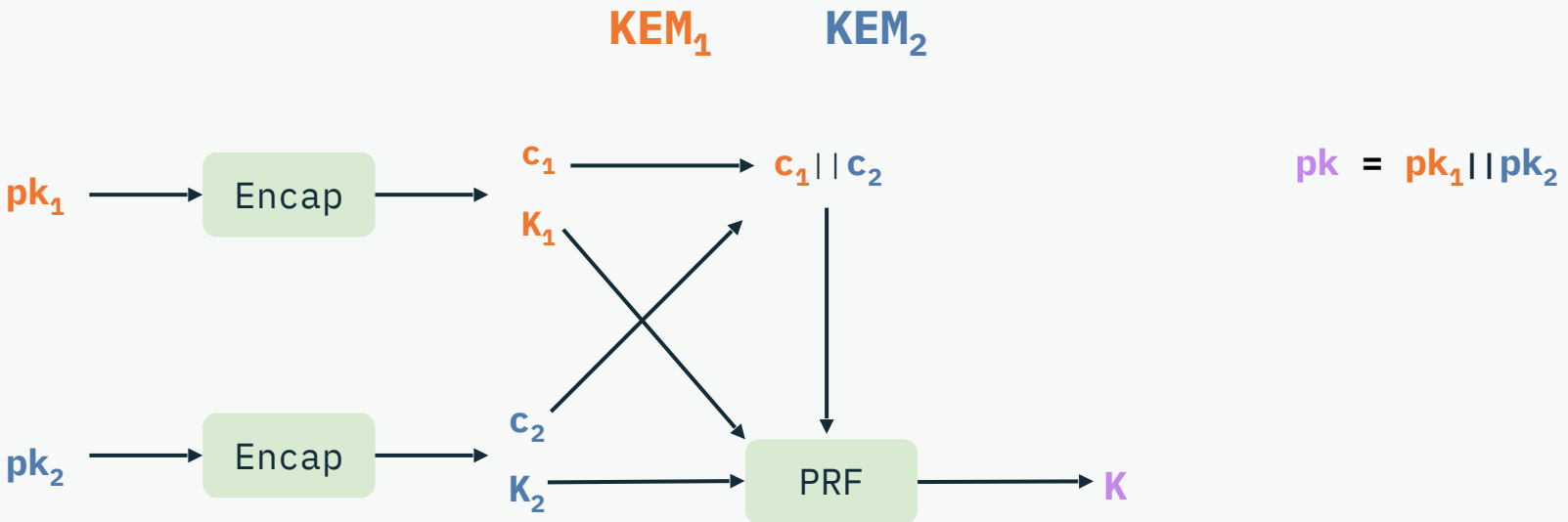
Hybrid KEMs: The Parallel Approach



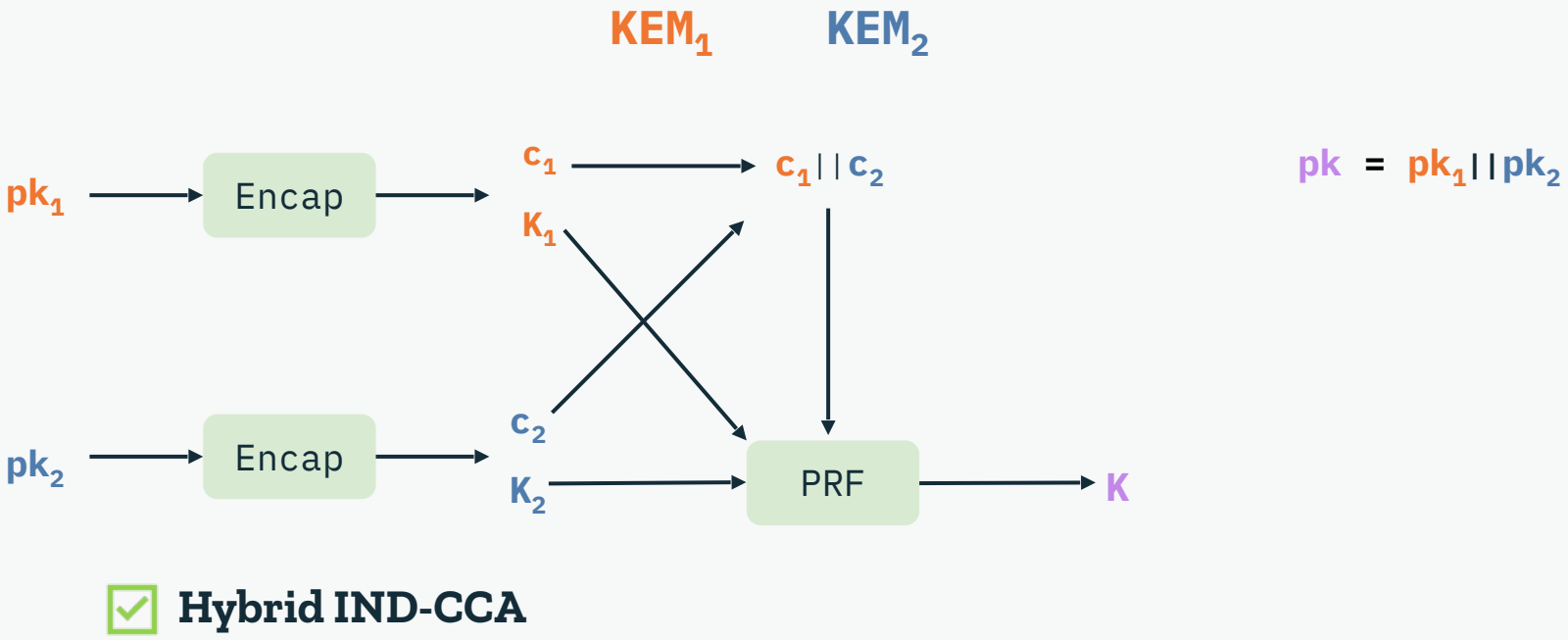
Hybrid KEMs: The Parallel Approach



Hybrid KEMs: The Parallel Approach

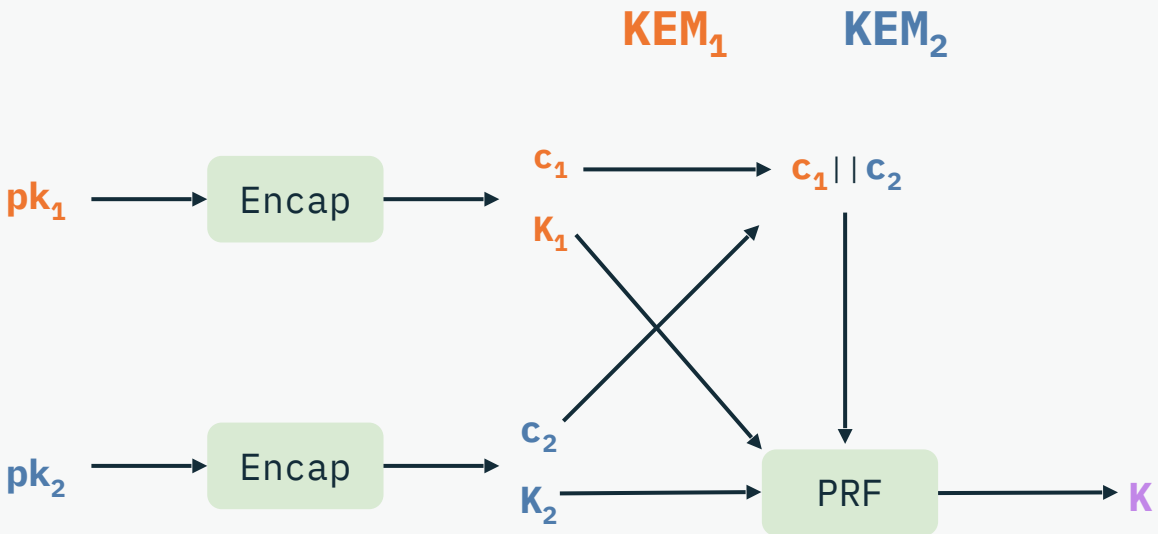


Hybrid KEMs: The Parallel Approach



☑ **Hybrid IND-CCA**

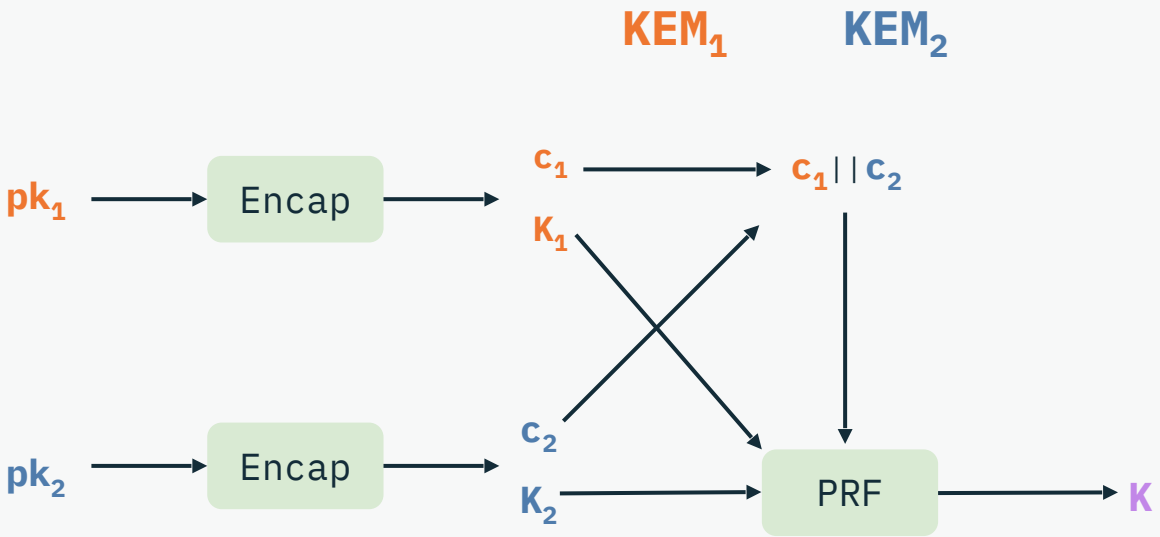
Hybrid KEMs: The Parallel Approach



☑ **Hybrid IND-CCA**

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

Hybrid KEMs: The Parallel Approach



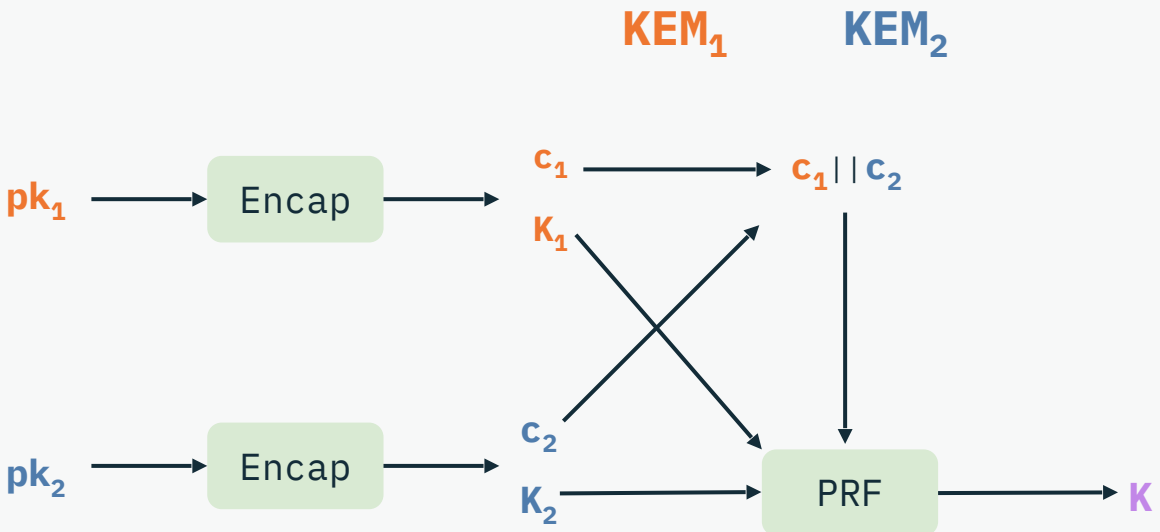
$pk = pk_1 || pk_2$

Suffices to distinguish **either** ciphertext

☑ **Hybrid IND-CCA**

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

Hybrid KEMs: The Parallel Approach



$pk = pk_1 || pk_2$

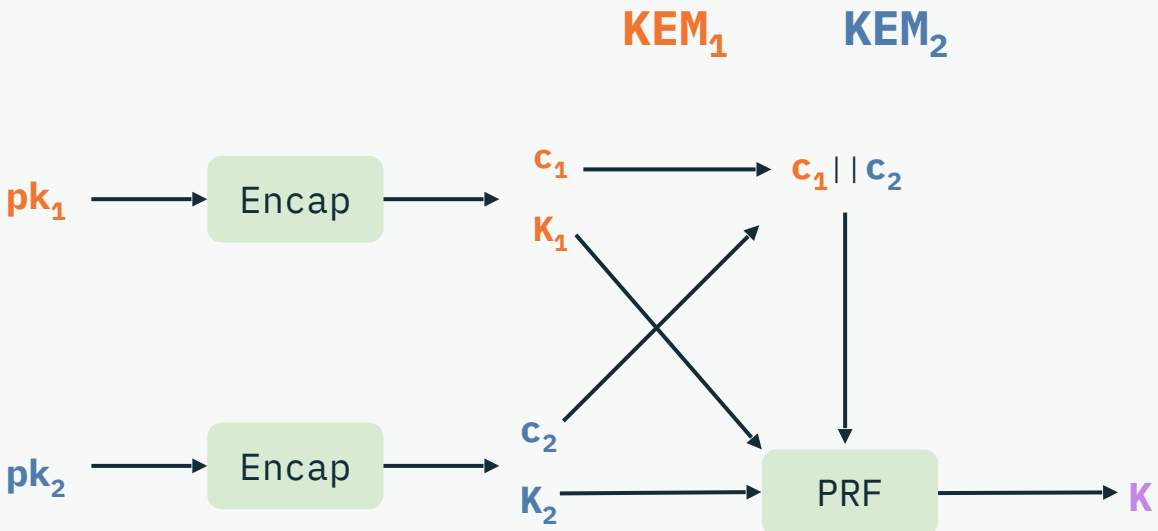
Suffices to distinguish **either** ciphertext

Do both ciphertexts have to be revealed?

☑ **Hybrid IND-CCA**

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

Hybrid KEMs: The Parallel Approach



$pk = pk_1 || pk_2$

Suffices to distinguish **either** ciphertext

Do both ciphertexts have to be revealed?

Sometimes c_1 is **unconditionally uniform**

☑ **Hybrid IND-CCA**

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

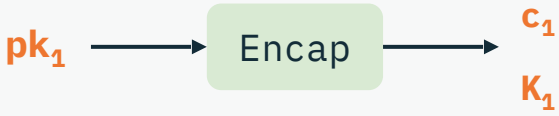
Outer-Encrypts-Inner Nested Combiner (OEINC)

$pk = pk_1 || pk_2$

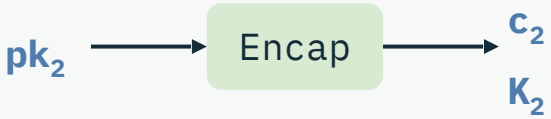
Outer-Encrypts-Inner Nested Combiner (OEINC)

$pk = pk_1 || pk_2$

"outOKEM"

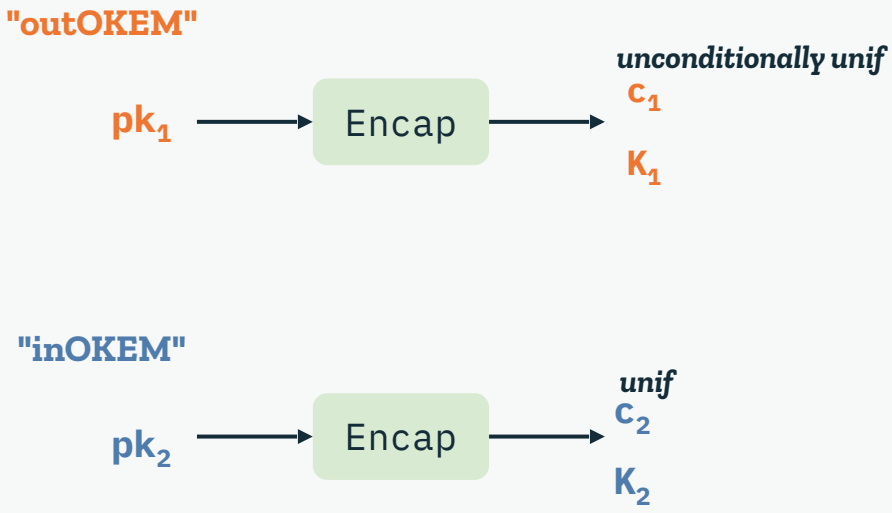


"inOKEM"



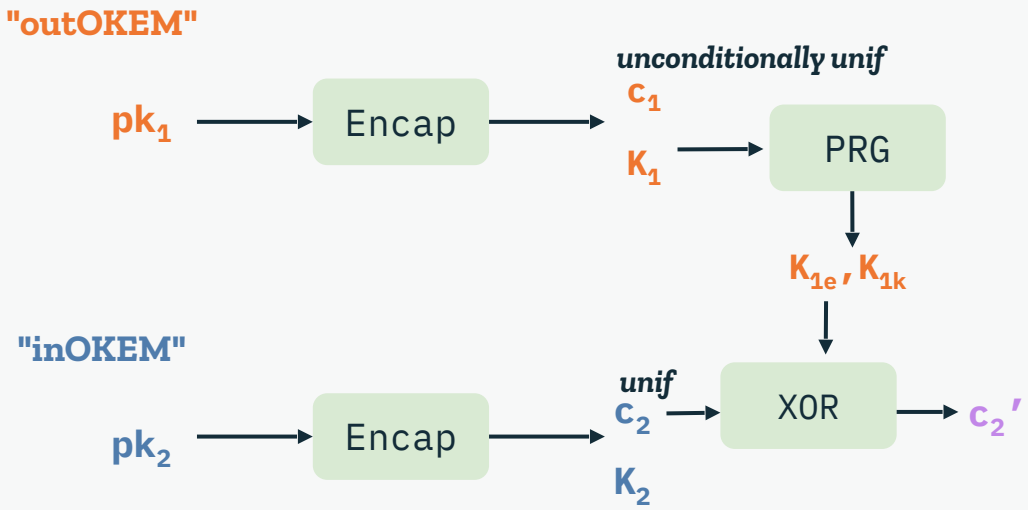
Outer-Encrypts-Inner Nested Combiner (OEINC)

$pk = pk_1 || pk_2$

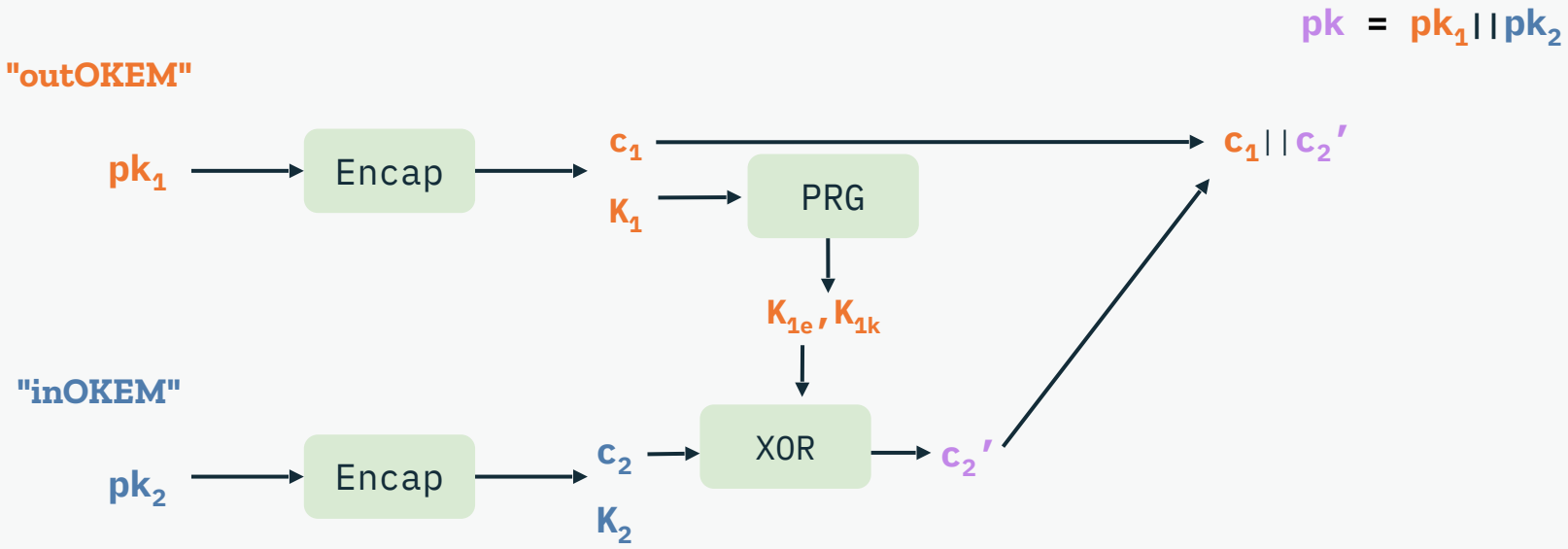


Outer-Encrypts-Inner Nested Combiner (OEINC)

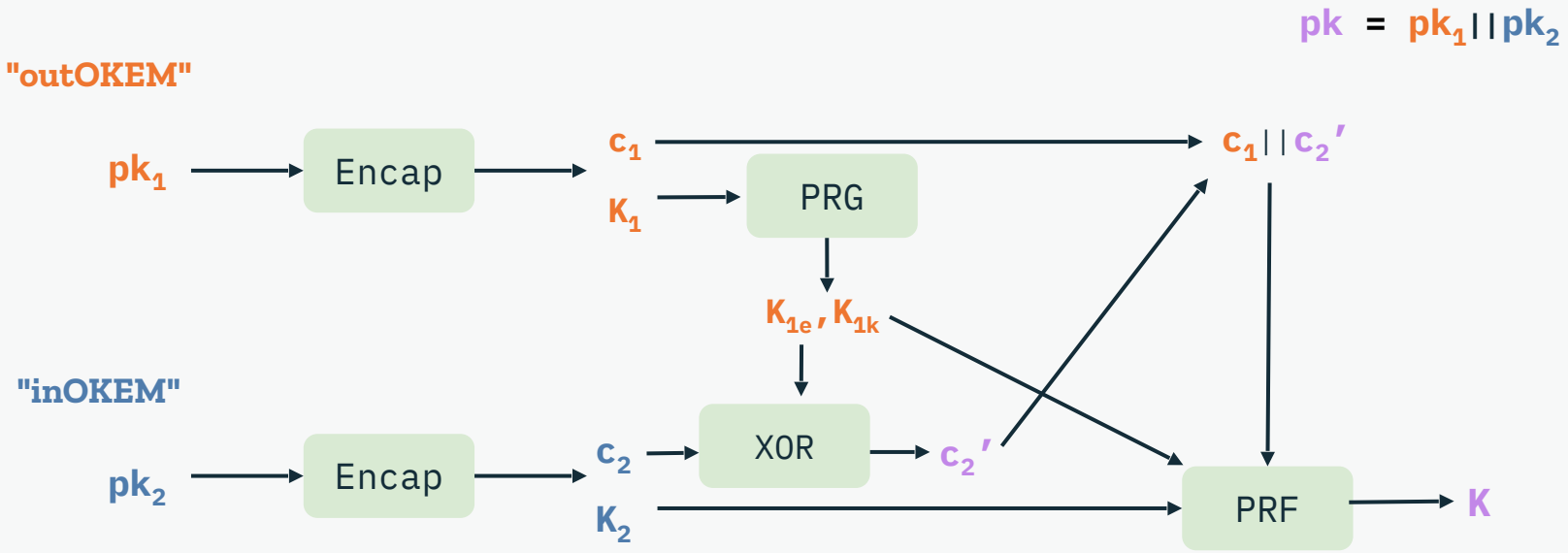
$pk = pk_1 || pk_2$



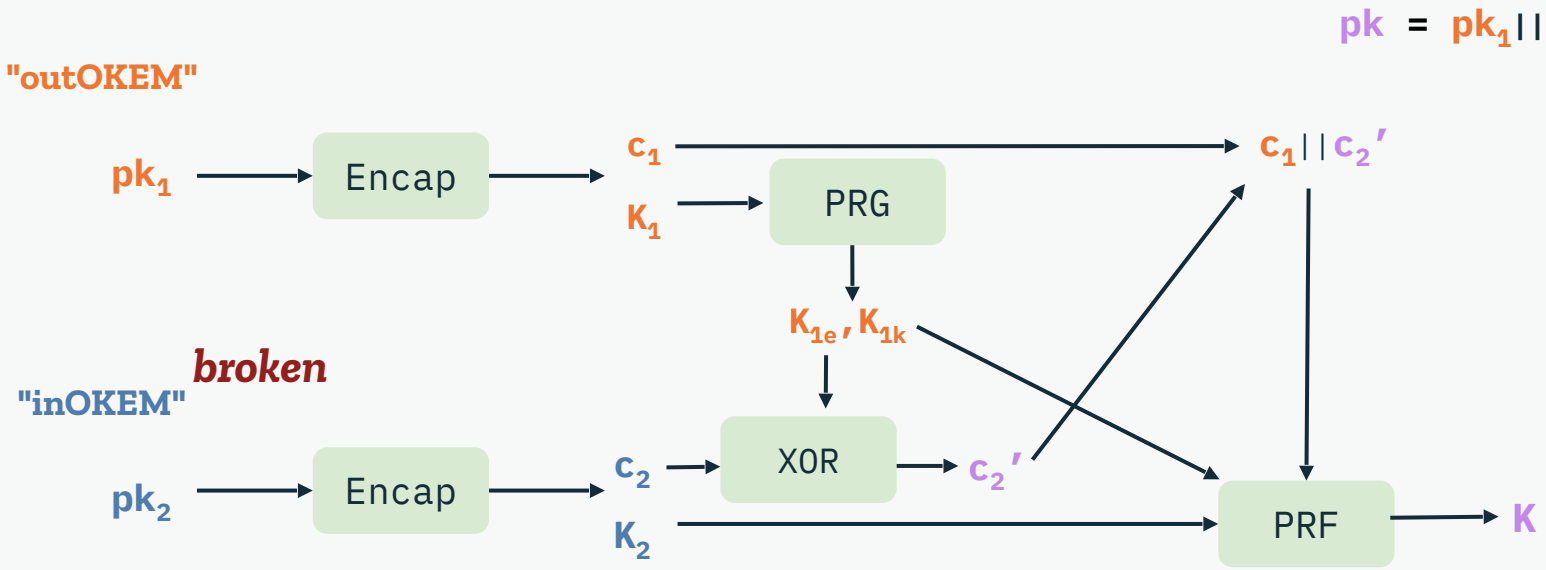
Outer-Encrypts-Inner Nested Combiner (OEINC)



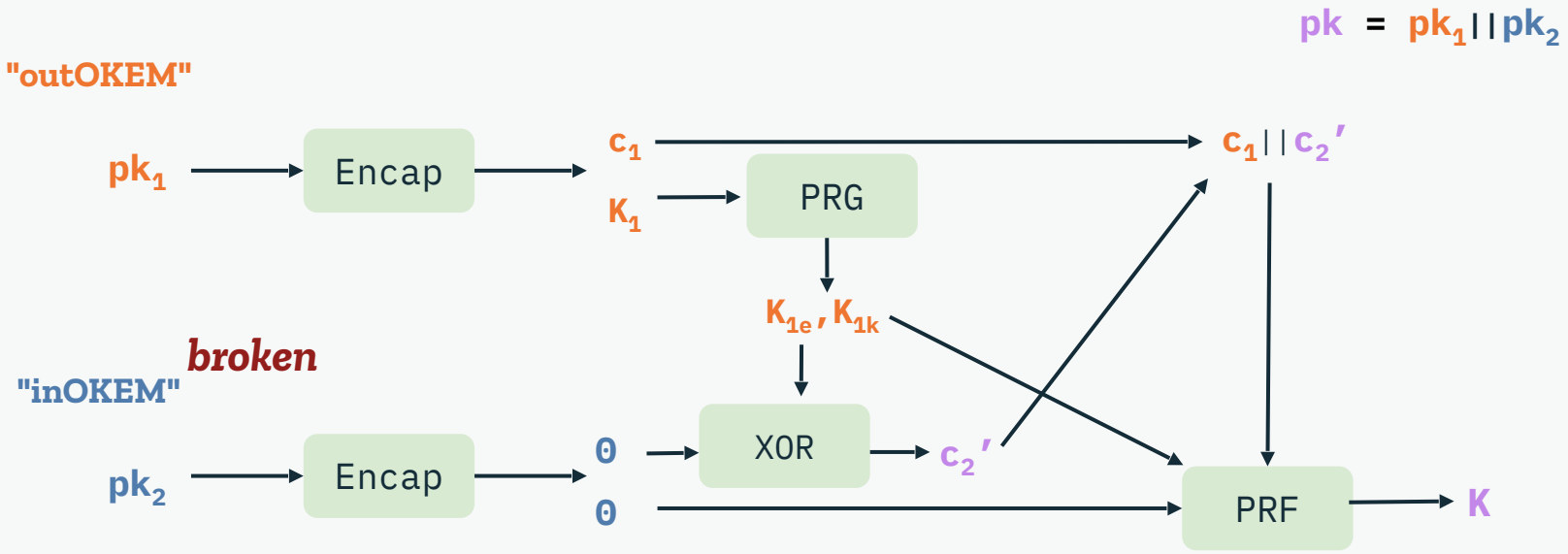
Outer-Encrypts-Inner Nested Combiner (OEINC)



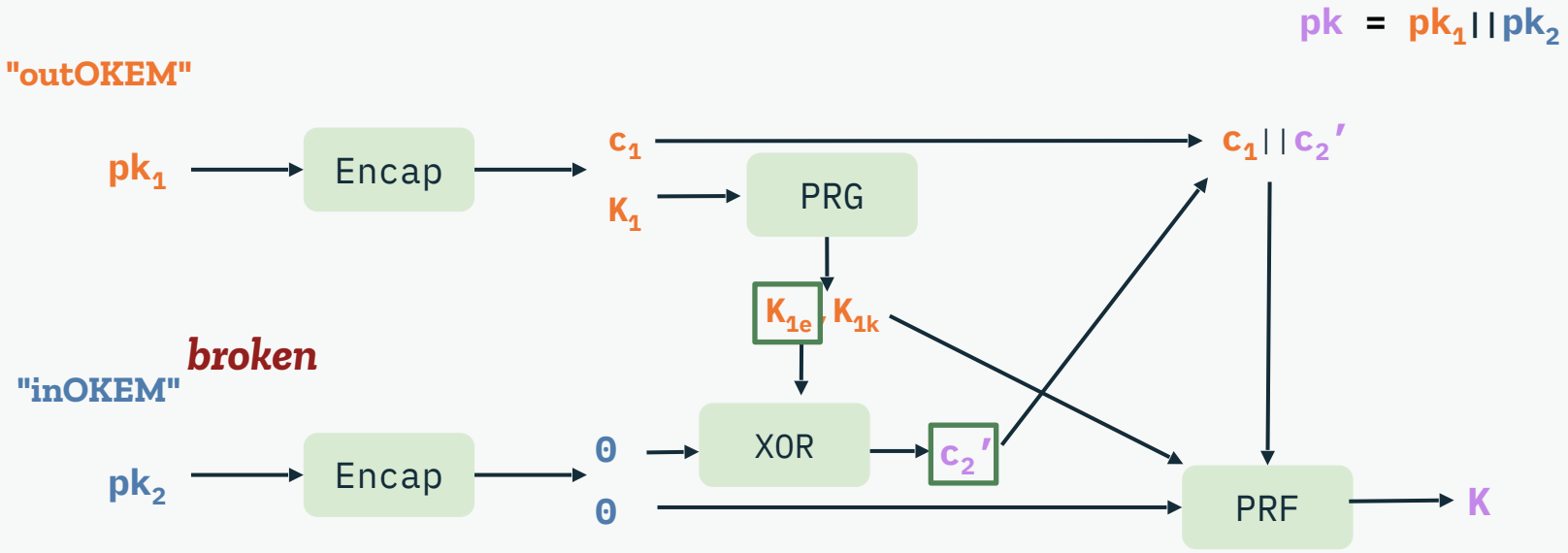
Outer-Encrypts-Inner Nested Combiner (OEINC)



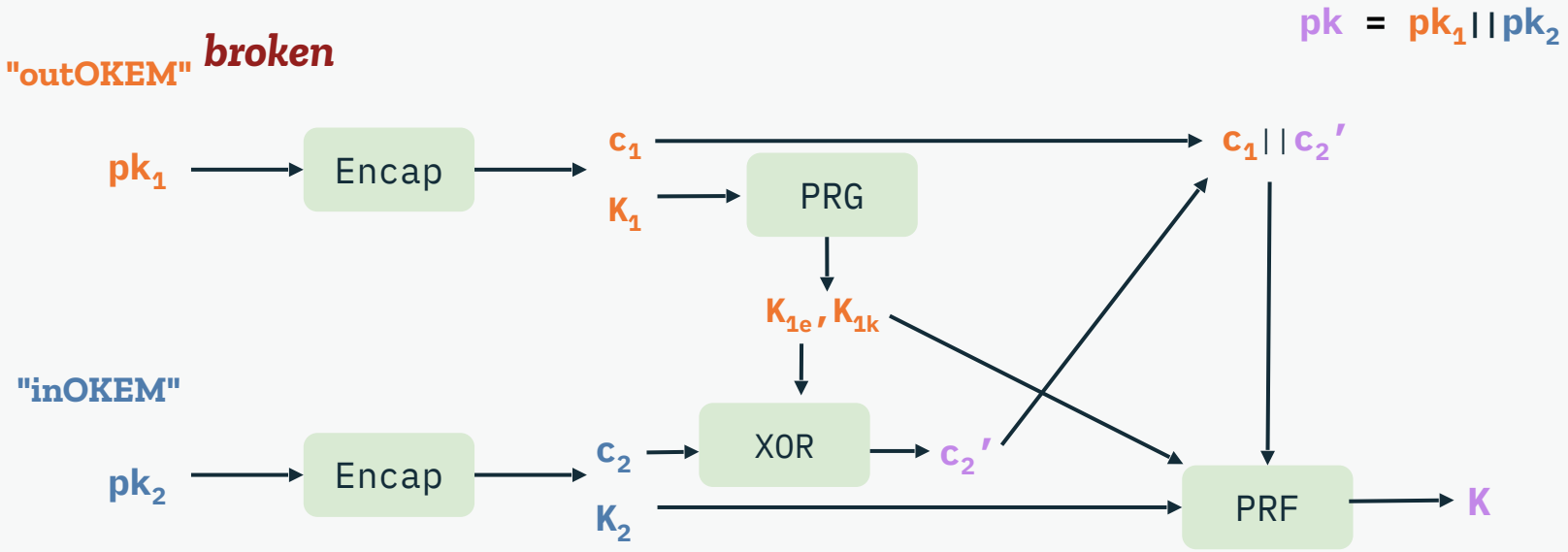
Outer-Encrypts-Inner Nested Combiner (OEINC)



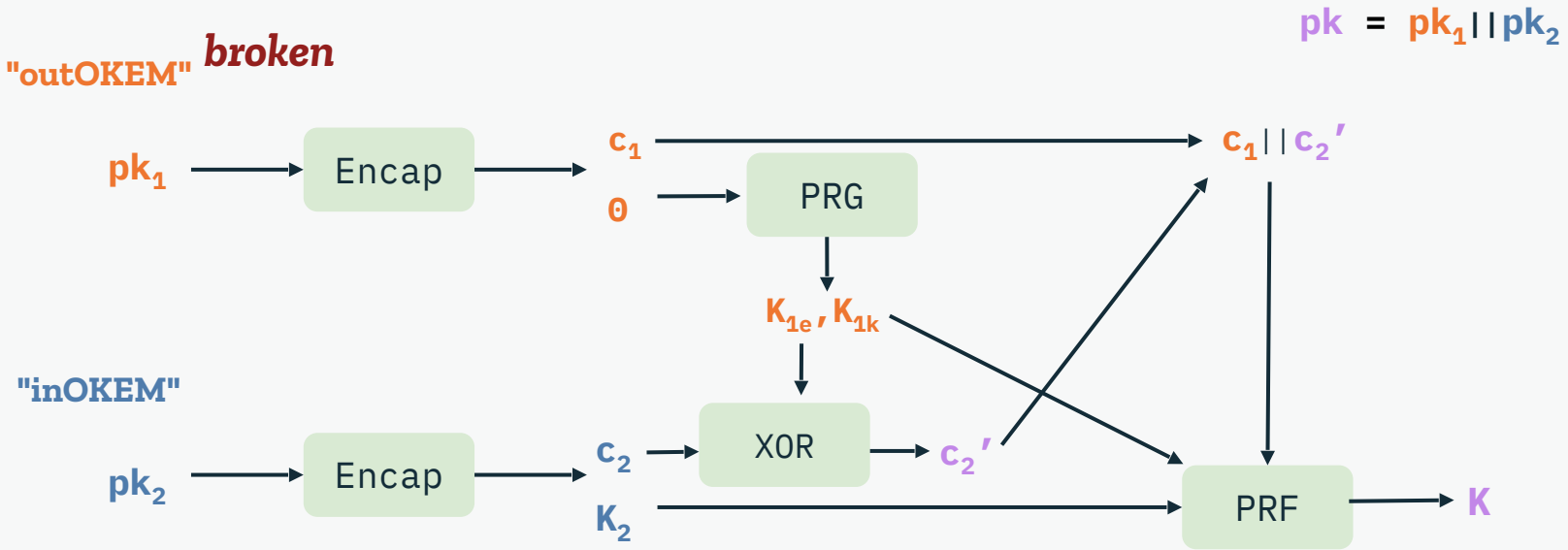
Outer-Encrypts-Inner Nested Combiner (OEINC)



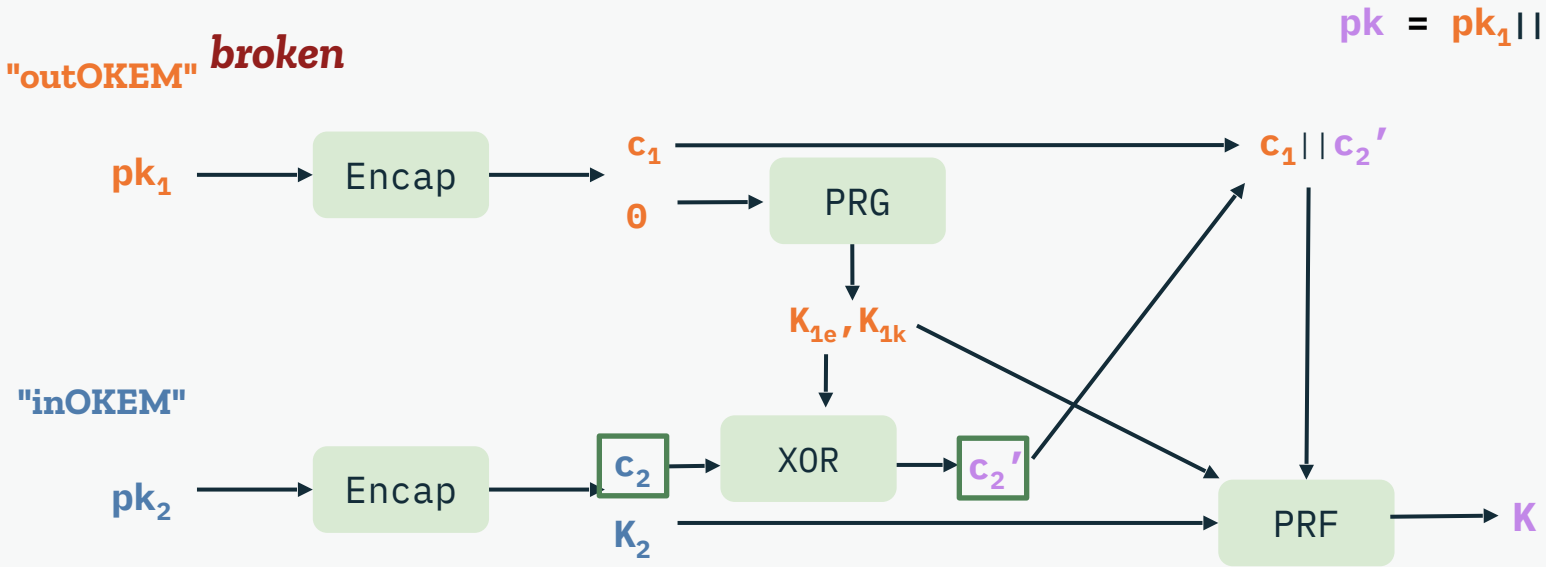
Outer-Encrypts-Inner Nested Combiner (OEINC)



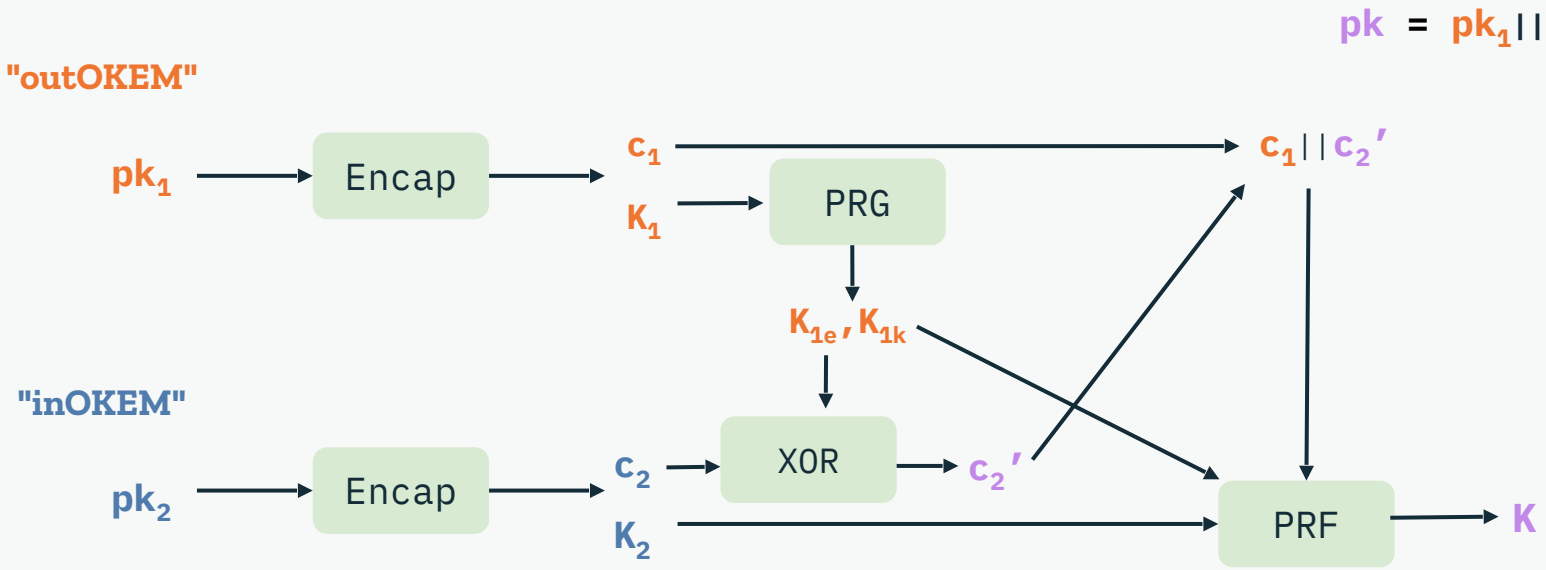
Outer-Encrypts-Inner Nested Combiner (OEINC)



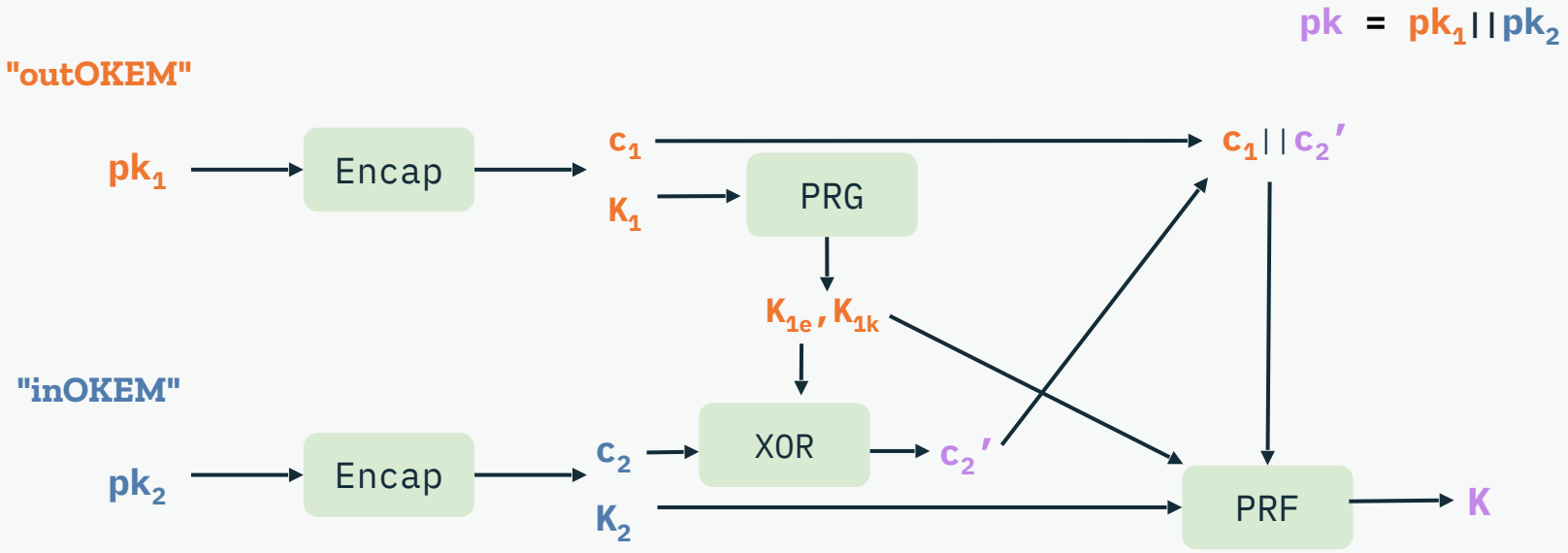
Outer-Encrypts-Inner Nested Combiner (OEINC)



Outer-Encrypts-Inner Nested Combiner (OEINC)



Outer-Encrypts-Inner Nested Combiner (OEINC)



Similar trick in: **PAKE combiner** [HR24,LL24], **deniable AKEM combiner** [GHJ25], **OPRF combiner** [FH25]

Security of OEINC

Security of OEINC

outOKEM must have **unconditional strong ciphertext uniformity**

Security of OEINC

outOKEM must have **unconditional strong ciphertext uniformity**

Achieves:

Security of OEINC

outOKEM must have **unconditional strong ciphertext uniformity**

Achieves:

IND-CCA outOKEM is IND-CCA or inOKEM is IND-CCA

Security of OEINC

outOKEM must have **unconditional strong ciphertext uniformity**

Achieves:

IND-CCA	outOKEM is IND-CCA	or	inOKEM is IND-CCA
Ciphertext uniformity	outOKEM is SPR-CCA	or	inOKEM is ct-unif

Security of OEINC

outOKEM must have **unconditional strong ciphertext uniformity**

Achieves:

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

Security of OEINC

outOKEM must have **unconditional strong ciphertext uniformity**

Achieves:

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

Hybrid pk-unif ⇒ both must be unconditionally pk-unif

Security of OEINC

outOKEM must have **unconditional strong ciphertext uniformity**

Achieves:

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

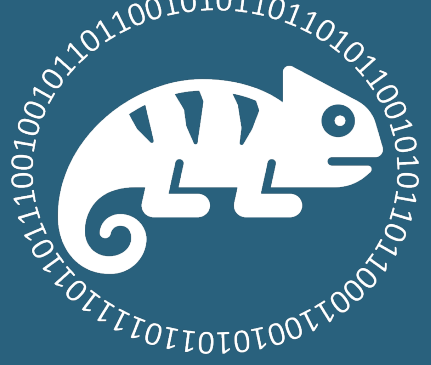
Hybrid pk-unif ⇒ both must be unconditionally pk-unif

We don't always need pk-unif

Our contributions

We build **hybrid obfuscated key exchange**

1. Define an obfuscated KEM (OKEM) combiner
2. Define Drivel, a new KEX protocol
3. Bonus: Hybrid PAKE from OKEM



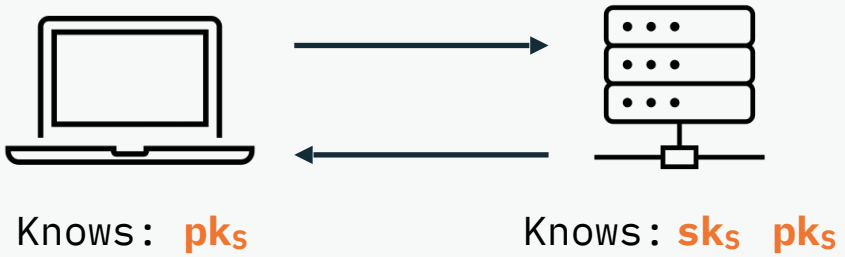
Obfuscated Key Exchange

Obfuscated Key Exchange

Client does key exchange with a **bridge**

Obfuscated Key Exchange

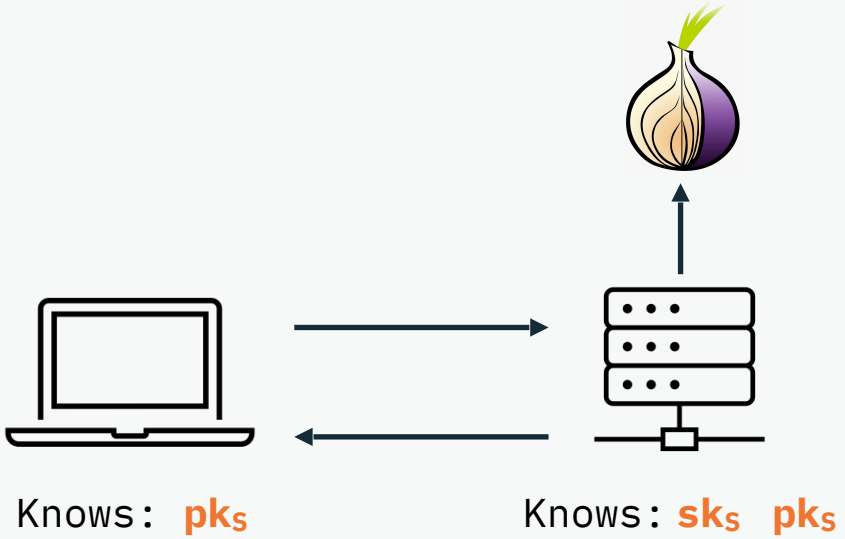
Client does key exchange with a **bridge**



Obfuscated Key Exchange

Client does key exchange with a **bridge**

The bridge relays client traffic to **Tor**

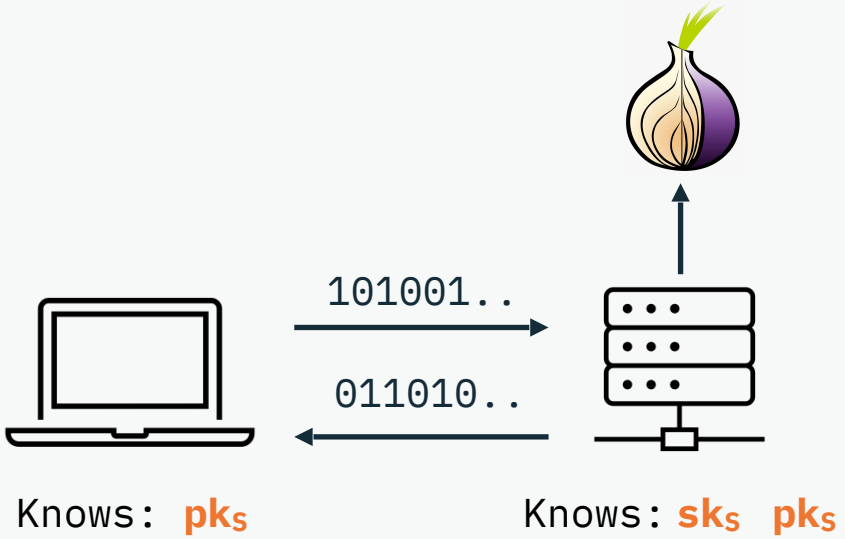


Obfuscated Key Exchange

Client does key exchange with a **bridge**

The bridge relays client traffic to **Tor**

Key exchange has to **appear uniform**



Drivel: Hybrid Obfuscated Key Exchange from OKEM

Drivel: Hybrid Obfuscated Key Exchange from OKEM

Prior work: pqobfs

Drivel: Hybrid Obfuscated Key Exchange from OKEM

pqobfs (simplified)

Client **Server** sk_S pk_S

Drivel: Hybrid Obfuscated Key Exchange from OKEM

pqobfs (simplified)

Client **Server** sk_S pk_S

$(sk_e, pk_e) := \text{OKEM.Keygen}()$

$(c_2, K_2) := \text{OKEM.Encap}(pk_e)$

$K_2 := \text{KEM.Decap}(sk_e, c_2)$

Drivel: Hybrid Obfuscated Key Exchange from OKEM

pqobfs (simplified)

Client

(sk_e, pk_e) := OKEM.Keygen()

(c_1, K_1) := OKEM.Encap(pk_s)

Server sk_s pk_s

K_1 := OKEM.Decap(sk_s, c_1)

(c_2, K_2) := OKEM.Encap(pk_e)

K_2 := KEM.Decap(sk_e, c_2)

Drivel: Hybrid Obfuscated Key Exchange from OKEM

pqobfs (simplified)

Client

(sk_e, pk_e) := OKEM.Keygen()

(c_1, K_1) := OKEM.Encap(pk_S)

Server sk_S pk_S

K_1 := OKEM.Decap(sk_S, c_1)

(c_2, K_2) := OKEM.Encap(pk_e)

K_2 := KEM.Decap(sk_e, c_2)

return H(K_1, K_2)

return H(K_1, K_2)

Drivel: Hybrid Obfuscated Key Exchange from OKEM

pqobfs (simplified)

Client

$(sk_e, pk_e) := \text{OKEM.Keygen}()$

$(c_1, K_1) := \text{OKEM.Encap}(pk_s)$

Requires pk-unif

c_1 pk_e

Server sk_s pk_s

$K_1 := \text{OKEM.Decap}(sk_s, c_1)$

$(c_2, K_2) := \text{OKEM.Encap}(pk_e)$

c_2

$K_2 := \text{KEM.Decap}(sk_e, c_2)$

return $H(K_1, K_2)$

return $H(K_1, K_2)$

Drivel: Hybrid Obfuscated Key Exchange from OKEM

pqobfs (simplified)

Client

(sk_e, pk_e) := OKEM.Keygen()

(c_1, K_1) := OKEM.Encap(pk_S)

Server sk_S pk_S

K_1 := OKEM.Decap(sk_S, c_1)

(c_2, K_2) := OKEM.Encap(pk_e)

K_2 := KEM.Decap(sk_e, c_2)

return H(K_1, K_2)

return H(K_1, K_2)

Drivel: Hybrid Obfuscated Key Exchange from OKEM

Client

(sk_e, pk_e) := KEM.Keygen()

(c_1, K_1) := OKEM.Encap(pk_s)

Server sk_s pk_s

K_1 := OKEM.Decap(sk_s, c_1)

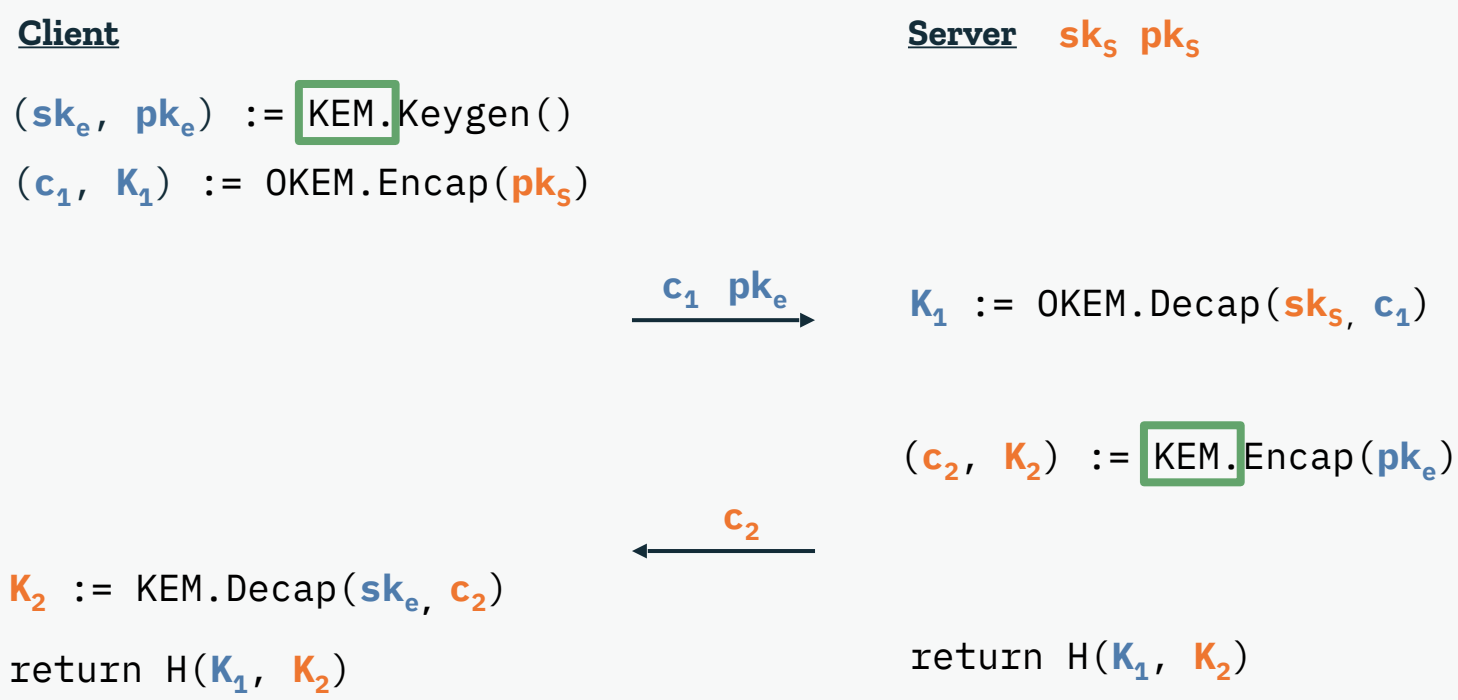
(c_2, K_2) := KEM.Encap(pk_e)

K_2 := KEM.Decap(sk_e, c_2)

return H(K_1, K_2)

return H(K_1, K_2)

Drivel: Hybrid Obfuscated Key Exchange from OKEM



Drivel: Hybrid Obfuscated Key Exchange from OKEM

Client

(sk_e, pk_e) := KEM.Keygen()

(c_1, K_1) := OKEM.Encap(pk_s)

Server sk_s pk_s

K_1 := OKEM.Decap(sk_s, c_1)

(c_2, K_2) := KEM.Encap(pk_e)

K_2 := KEM.Decap(sk_e, c_2)

return H(K_1, K_2)

return H(K_1, K_2)

Drivel: Hybrid Obfuscated Key Exchange from OKEM

Client

(sk_e, pk_e) := KEM.Keygen()

(c_1, K_1) := OKEM.Encap(pk_s)

K_2 := KEM.Decap(sk_e, c_2)

return H(K_1, K_2)

Server sk_s pk_s

K_1 := OKEM.Decap(sk_s, c_1)

(c_2, K_2) := KEM.Encap(pk_e)

return H(K_1, K_2)

Drivel: Hybrid Obfuscated Key Exchange from OKEM

Client

(sk_e, pk_e) := KEM.Keygen()

(c_1, K_1) := OKEM.Encap(pk_S)

epk_e := $SE.Enc_{K1}(pk_e)$

Server sk_S pk_S

K_1 := OKEM.Decap(sk_S, c_1)

pk_e := $SE.Dec_{K1}(epk_e)$

(c_2, K_2) := KEM.Encap(pk_e)

K_2 := KEM.Decap(sk_e, c_2)

return H(K_1, K_2)

return H(K_1, K_2)

Drivel: Hybrid Obfuscated Key Exchange from OKEM

Client

$(sk_e, pk_e) := KEM.Keygen()$

$(c_1, K_1) := OKEM.Encap(pk_S)$

$epk_e := SE.Enc_{K1}(pk_e)$

$K_2 := KEM.Decap(sk_e, c_2)$

return $H(K_1, K_2)$

Server sk_S pk_S

$K_1 := OKEM.Decap(sk_S, c_1)$

$pk_e := SE.Dec_{K1}(epk_e)$

$(c_2, K_2) := KEM.Encap(pk_e)$

return $H(K_1, K_2)$

Drivel: Hybrid Obfuscated Key Exchange from OKEM

Client

$(sk_e, pk_e) := KEM.Keygen()$

$(c_1, K_1) := OKEM.Encap(pk_S)$

$epk_e := SE.Enc_{K1}(pk_e)$

$c_1\ epk_e$ →

$c_2 := SE.Dec_{K1}(ec_2)$

$K_2 := KEM.Decap(sk_e, c_2)$

return $H(K_1, K_2)$

Server sk_S pk_S

$K_1 := OKEM.Decap(sk_S, c_1)$

$pk_e := SE.Dec_{K1}(epk_e)$

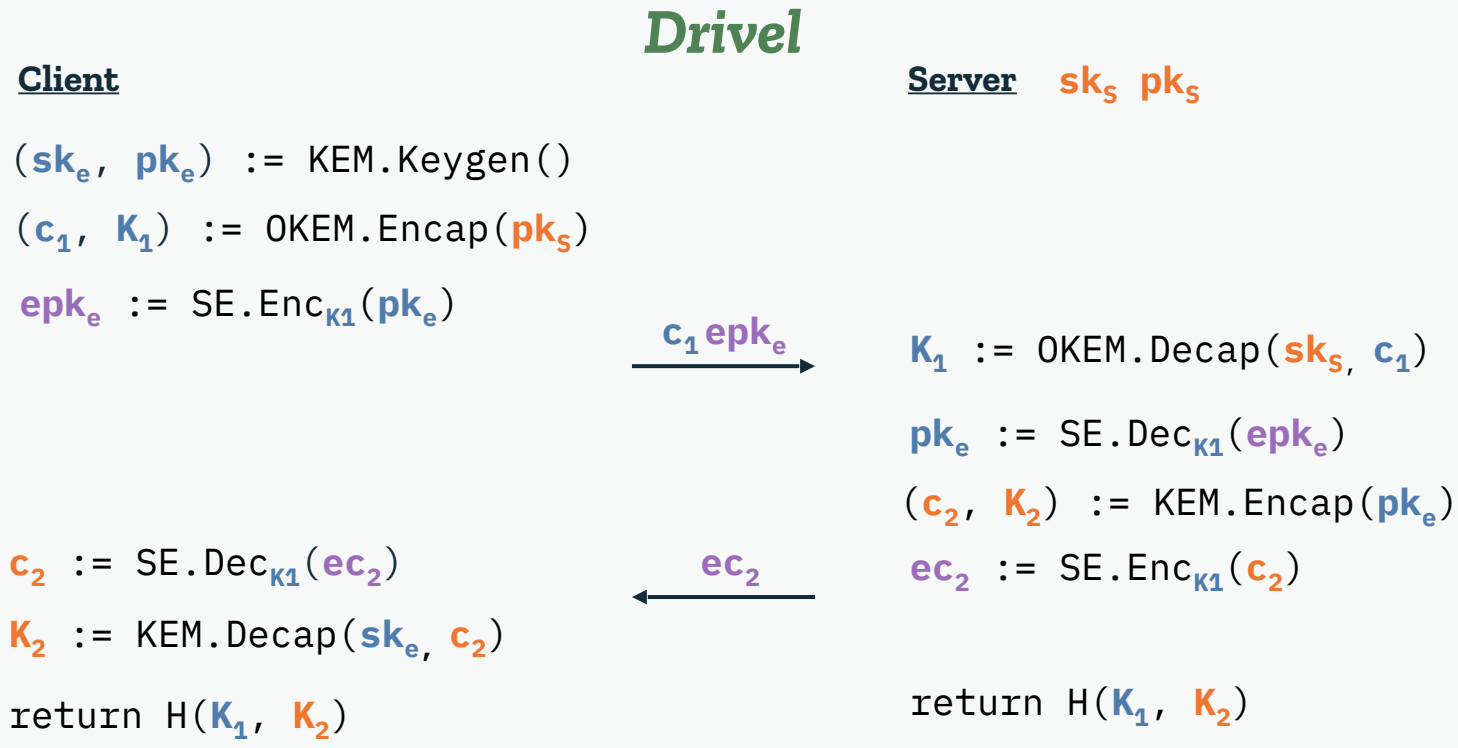
$(c_2, K_2) := KEM.Encap(pk_e)$

← ec_2

$ec_2 := SE.Enc_{K1}(c_2)$

return $H(K_1, K_2)$

Drivel: Hybrid Obfuscated Key Exchange from OKEM



Drivel: Hybrid Obfuscated Key Exchange from OKEM

Drivel

No pk-unif requirement anymore

Client

$(sk_e, pk_e) := KEM.Keygen()$

$(c_1, K_1) := OKEM.Encap(pk_S)$

$epk_e := SE.Enc_{K1}(pk_e)$

$c_1\ epk_e$ →

$K_1 := OKEM.Decap(sk_S, c_1)$

$pk_e := SE.Dec_{K1}(epk_e)$

$(c_2, K_2) := KEM.Encap(pk_e)$

$ec_2 := SE.Enc_{K1}(c_2)$

← ec_2

$c_2 := SE.Dec_{K1}(ec_2)$

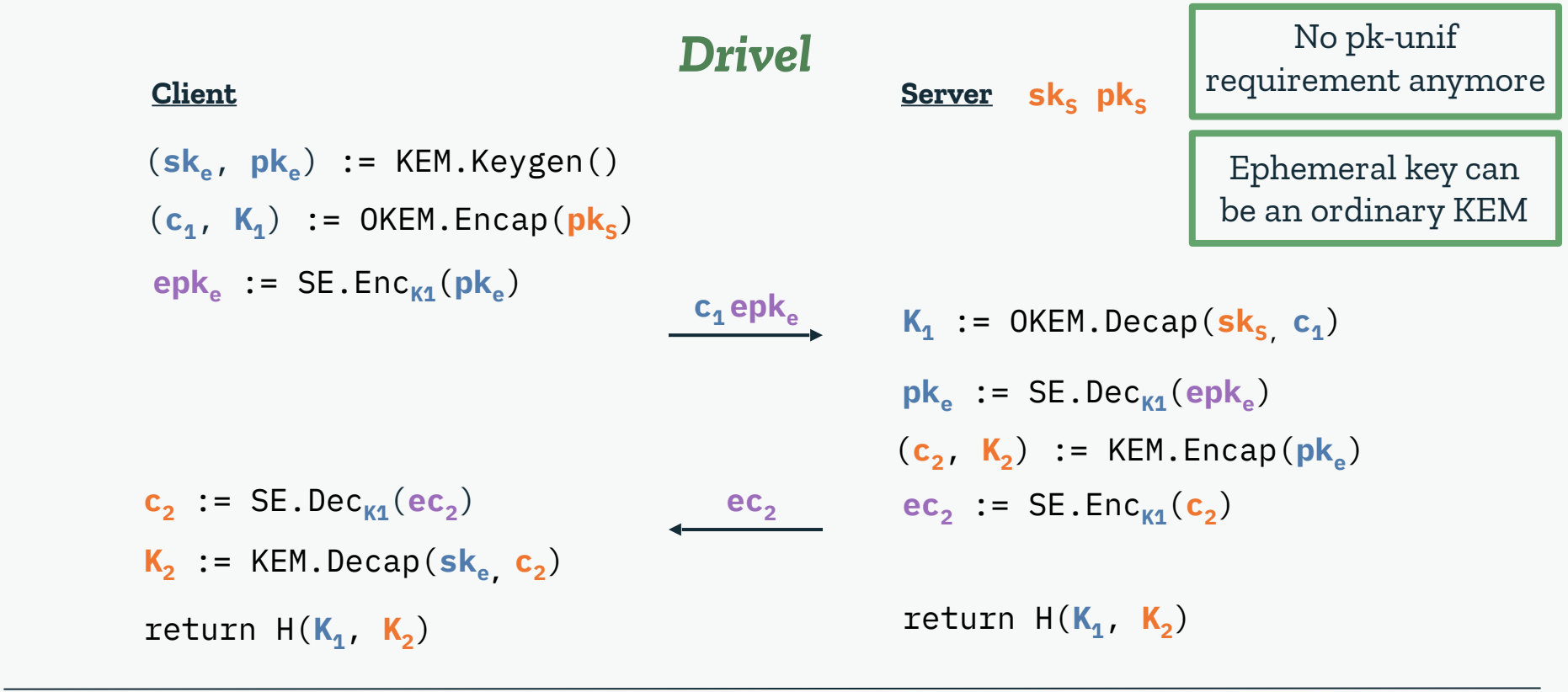
$K_2 := KEM.Decap(sk_e, c_2)$

return $H(K_1, K_2)$

Server sk_S pk_S

return $H(K_1, K_2)$

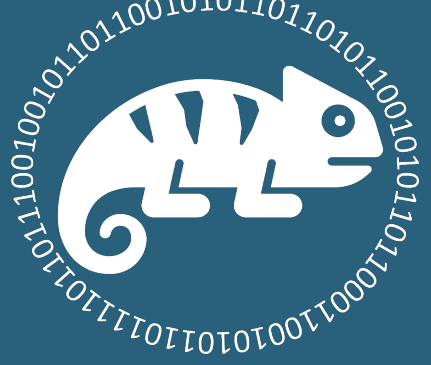
Drivel: Hybrid Obfuscated Key Exchange from OKEM



Our contributions

We build **hybrid obfuscated key exchange**

1. Define an obfuscated KEM (OKEM) combiner
2. Define Drivel, a new KEX protocol
3. Bonus: Hybrid PAKE from OKEM



Bonus OEINC Application: Hybrid PAKE

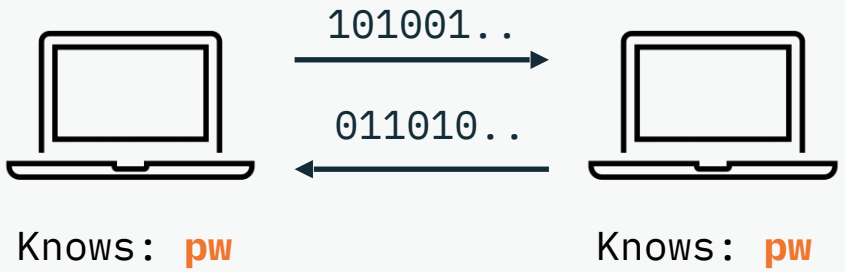
Bonus OEINC Application: Hybrid PAKE

Password authenticated key exchange (PAKE)

Bonus OEINC Application: Hybrid PAKE

Password authenticated key exchange (PAKE)

Parties w/ **low-entropy password** want to establish a **high-entropy shared secret**:

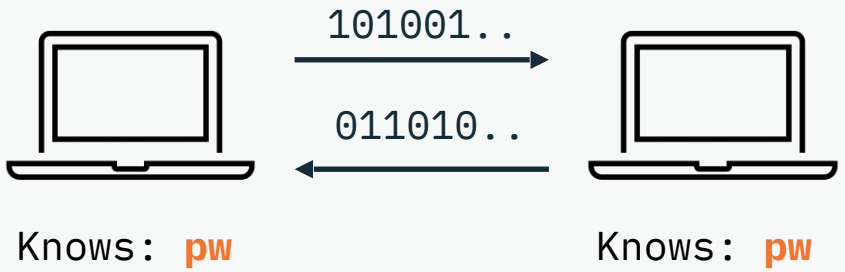


Bonus OEINC Application: Hybrid PAKE

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

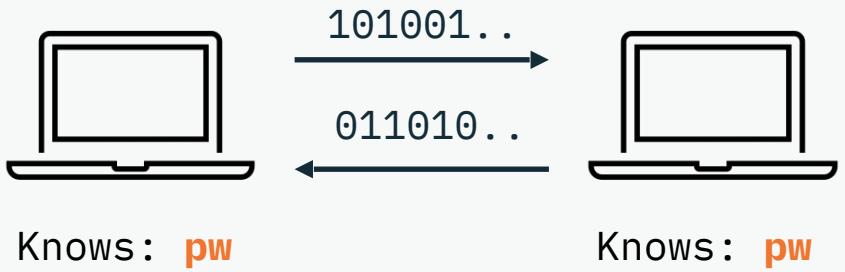


Bonus OEINC Application: Hybrid PAKE

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 0 pw guesses**

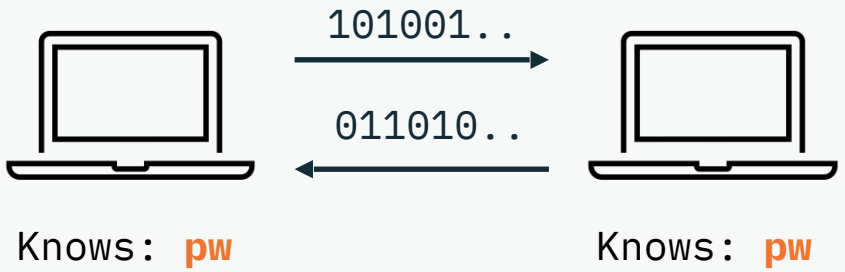


Bonus OEINC Application: Hybrid PAKE

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 0 pw guesses**



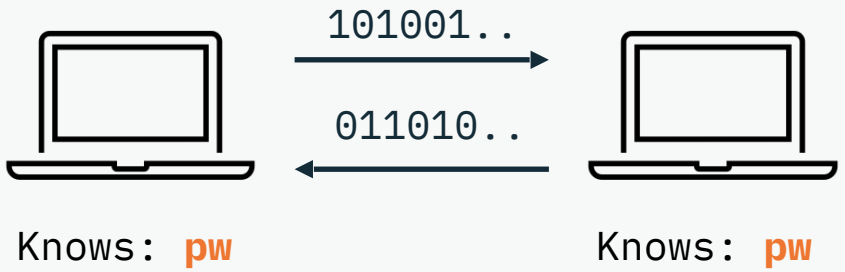
Tons of **KEM-based PAKEs:** OQUAKE, Tempo, NoIC, CHIC, EKE-PRF, CAKE, OCAKE, ...

Bonus OEINC Application: 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 0 pw guesses**



Tons of **KEM-based PAKEs:** OQUAKE, Tempo, NoIC, CHIC, EKE-PRF, CAKE, OCAKE, ...

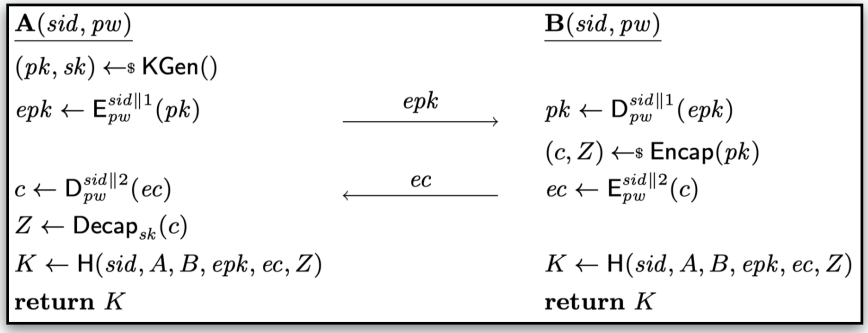
Bonus OEINC Application: Hybrid PAKE

Bonus OEINC Application: Hybrid PAKE

CAKE proven in the UC model with
adaptive corruptions

Bonus OEINC Application: Hybrid PAKE

CAKE proven in the UC model with **adaptive corruptions**

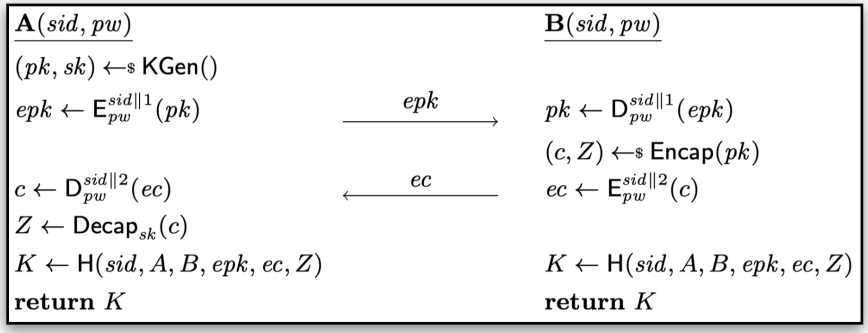


CAKE

Bonus OEINC Application: Hybrid PAKE

CAKE proven in the UC model with **adaptive corruptions**

Requires **ciphertext and public key uniformity***



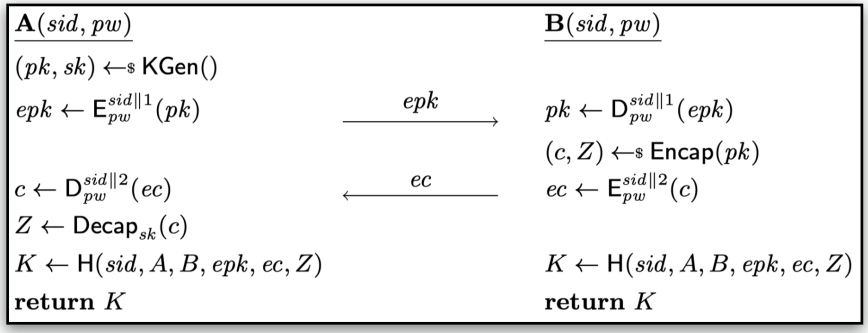
CAKE

Bonus OEINC Application: Hybrid PAKE

CAKE proven in the UC model with **adaptive corruptions**

Requires **ciphertext and public key uniformity***

- LWE schemes lose unconditional pk-unif bc of a well-known optimization



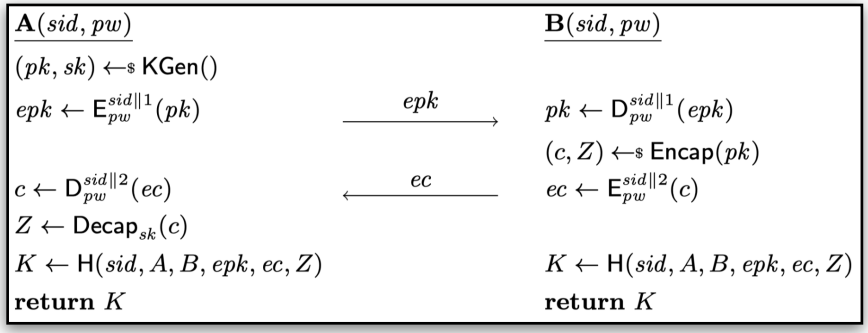
CAKE

Bonus OEINC Application: Hybrid PAKE

CAKE proven in the UC model with **adaptive corruptions**

Requires **ciphertext and public key uniformity***

- LWE schemes lose unconditional pk-unif bc of a well-known optimization
- Undo this optimization. Call it **StatFrodoKEM**



CAKE

Bonus OEINC Application: Hybrid PAKE

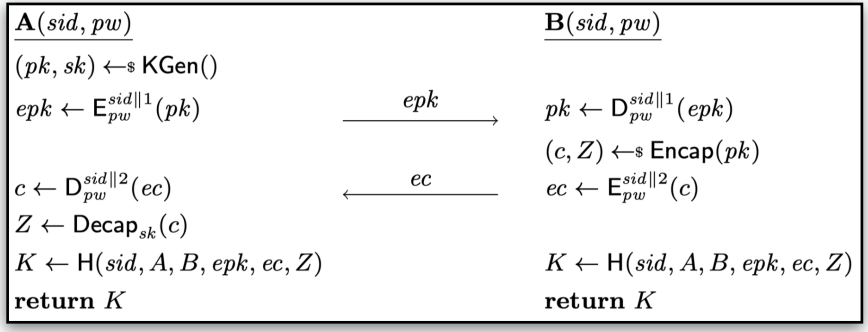
CAKE proven in the UC model with **adaptive corruptions**

Requires **ciphertext and public key uniformity***

- LWE schemes lose unconditional pk-unif bc of a well-known optimization
- Undo this optimization. Call it **StatFrodoKEM**

Hence

CAKE[OEINC[DHKEM+Elligator, StatFrodoKEM]]



CAKE

Bonus OEINC Application: Hybrid PAKE

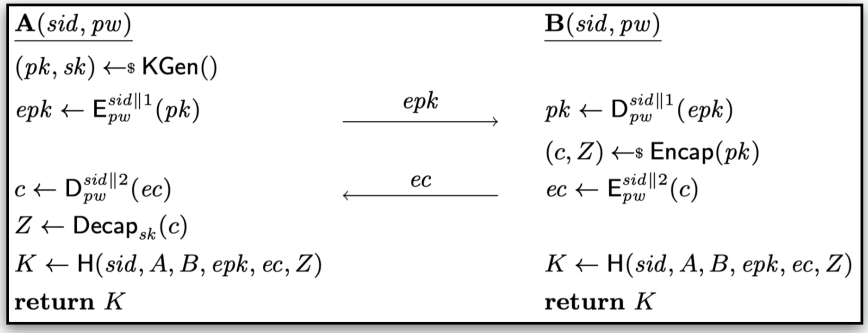
CAKE proven in the UC model with **adaptive corruptions**

Requires **ciphertext and public key uniformity***

- LWE schemes lose unconditional pk-unif bc of a well-known optimization
- Undo this optimization. Call it **StatFrodoKEM**

Hence

CAKE[OEINC[DHKEM+Elligator, StatFrodoKEM]]



CAKE

First hybrid PAKE with security against adaptive corruptions

Bonus OEINC Application: Hybrid PAKE

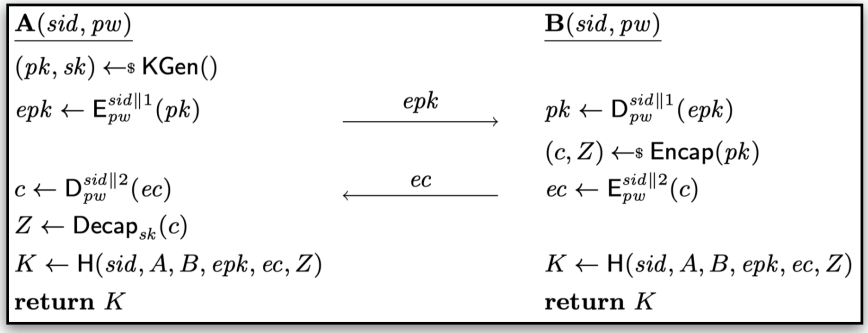
CAKE proven in the UC model with **adaptive corruptions**

Requires **ciphertext and public key uniformity***

- LWE schemes lose unconditional pk-unif bc of a well-known optimization
- Undo this optimization. Call it **StatFrodoKEM**

Hence

CAKE[OEINC[DHKEM+Elligator, StatFrodoKEM]]



CAKE

First hybrid PAKE with security against adaptive corruptions

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

Bonus OEINC Application: Hybrid PAKE

CAKE proven in the UC model with **adaptive corruptions**

Requires **ciphertext and public key uniformity***

- LWE schemes lose unconditional pk-unif bc of a well-known optimization
- Undo this optimization. Call it **StatFrodoKEM**

Hence

CAKE[OEINC[DHKEM+Elligator, StatFrodoKEM]]

$\mathbf{A}(sid, pw)$		$\mathbf{B}(sid, pw)$
$(pk, sk) \leftarrow\$ \mathsf{KGen}()$		
$epk \leftarrow \mathsf{E}_{pw}^{sid\|\|1}(pk)$	$\xrightarrow{epk}$	$pk \leftarrow \mathsf{D}_{pw}^{sid\|\|1}(epk)$
		$(c, Z) \leftarrow\$ \mathsf{Encap}(pk)$
$c \leftarrow \mathsf{D}_{pw}^{sid\|\|2}(ec)$	$\xleftarrow{ec}$	$ec \leftarrow \mathsf{E}_{pw}^{sid\|\|2}(c)$
$Z \leftarrow \mathsf{Decap}_{sk}(c)$		
$K \leftarrow \mathsf{H}(sid, A, B, epk, ec, Z)$		$K \leftarrow \mathsf{H}(sid, A, B, epk, ec, Z)$
return K		**return** K

CAKE

First hybrid PAKE with security against adaptive corruptions

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

7.5x comms overhead compared to 3-round PAKEs

Conclusion

Conclusion

Contributions

Conclusion

Contributions

1. **OEINC**, an obfuscated KEM (OKEM) combiner

Conclusion

Contributions

1. **OEINC**, an obfuscated KEM (OKEM) combiner
2. **Drivel**, a new hybrid obfuscated key exchange protocol

Conclusion

Contributions

1. **OEINC**, an obfuscated KEM (OKEM) combiner
2. **Drivel**, a new hybrid obfuscated key exchange protocol
3. First adaptively secure **hybrid PAKE**

Conclusion

Hybrid Obfuscated Key Exchange and KEMs

ia.cr/2025/408

Felix Günther
Michael Rosenberg
Douglas Stebila
Shannon Veitch

Contributions

1. **OEINC**, an obfuscated KEM (OKEM) combiner
2. **Drivel**, a new hybrid obfuscated key exchange protocol
3. First adaptively secure **hybrid PAKE**

Questions?

Ask me questions irl

References:

- Faller, Hesse. How to (not) combine Oblivious Pseudorandom Functions. ia.cr/2025/1084
- Gajland, Hwang, Janneck. Shadowfax: A Deniability-Preserving AKEM Combiner. ia.cr/2025/154
- 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-irtf-cfrg-kemeleon/ **<— send us your feedback!**
- Hesse, Rosenberg. PAKE Combiners and Efficient Post-Quantum Instantiations. ia.cr/2024/1621
- Lyu, Liu. Hybrid Password Authentication Key Exchange in the UC Framework. ia.cr/2024/1630.