Anonymous, Robust Post-Quantum Public Key Encryption

Varun Maram
Applied Cryptography Group
ETH Zurich

Joint work with Paul Grubbs and Kenneth G. Paterson

[Full version of paper: https://eprint.iacr.org/2021/708.pdf]

NIST PQC Finalists

PQC Standardization Process: Third Round Candidate Announcement

July 22, 2020

It has been almost a year and a half since the second round of the NIST PQC Standardization Process began. After careful consideration, NIST would like to announce the candidates that will be moving on to the third round. The seven third-round Finalists are:

Third Round Finalists

Public-Key Encryption/KEMs
Classic McEliece
CRYSTALS-KYBER
NTRU
SABER

(Image taken from *https://www.nist.gov/news-events/news/2020/07/pqc-standardization-process-third-round-candidate-announcement*)

NIST PQC Finalists

PQC Standardization Process: Third Round Candidate Announcement

July 22, 2020

It has been almost a year and a half since the second round of the NIST PQC Standardization Process began. After careful consideration, NIST would like to announce the candidates that will be moving on to the third round. The seven third-round Finalists are:

Third Round Finalists

Public-Key Encryption/KEMs
- Classic McEliece
- CRYSTALS-KYBER
- NTRU
- SABER

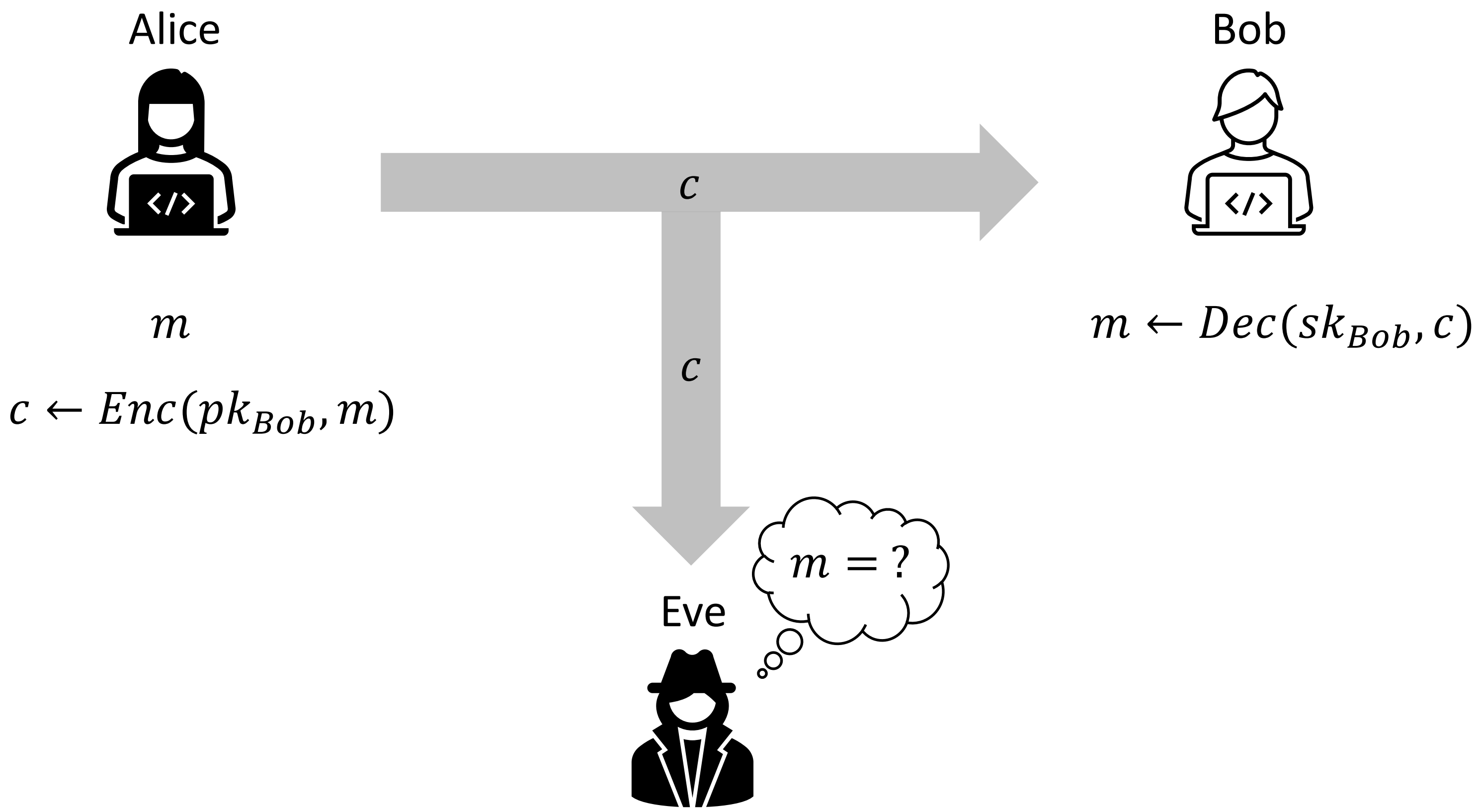
4.A.2 Security Definition for Encryption/Key-Establishment
NIST intends to standardize one or more schemes that enable "semantically secure" encryption or key encapsulation with respect to adaptive chosen ciphertext attack, for general use. This property is generally denoted *IND-CCA2 security* in academic literature.

The above security definition should be taken as a statement of what NIST will consider to be a relevant attack. Submitted KEM and encryption schemes will be evaluated based on how well they appear to provide this property, when used as specified by the

(Image taken from *https://www.nist.gov/news-events/news/2020/07/pqc-standardization-process-third-round-candidate-announcement*)
(Image taken from *https://csrc.nist.gov/CSRC/media/Projects/Post-Quantum-Cryptography/documents/call-for-proposals-final-dec-2016.pdf*)

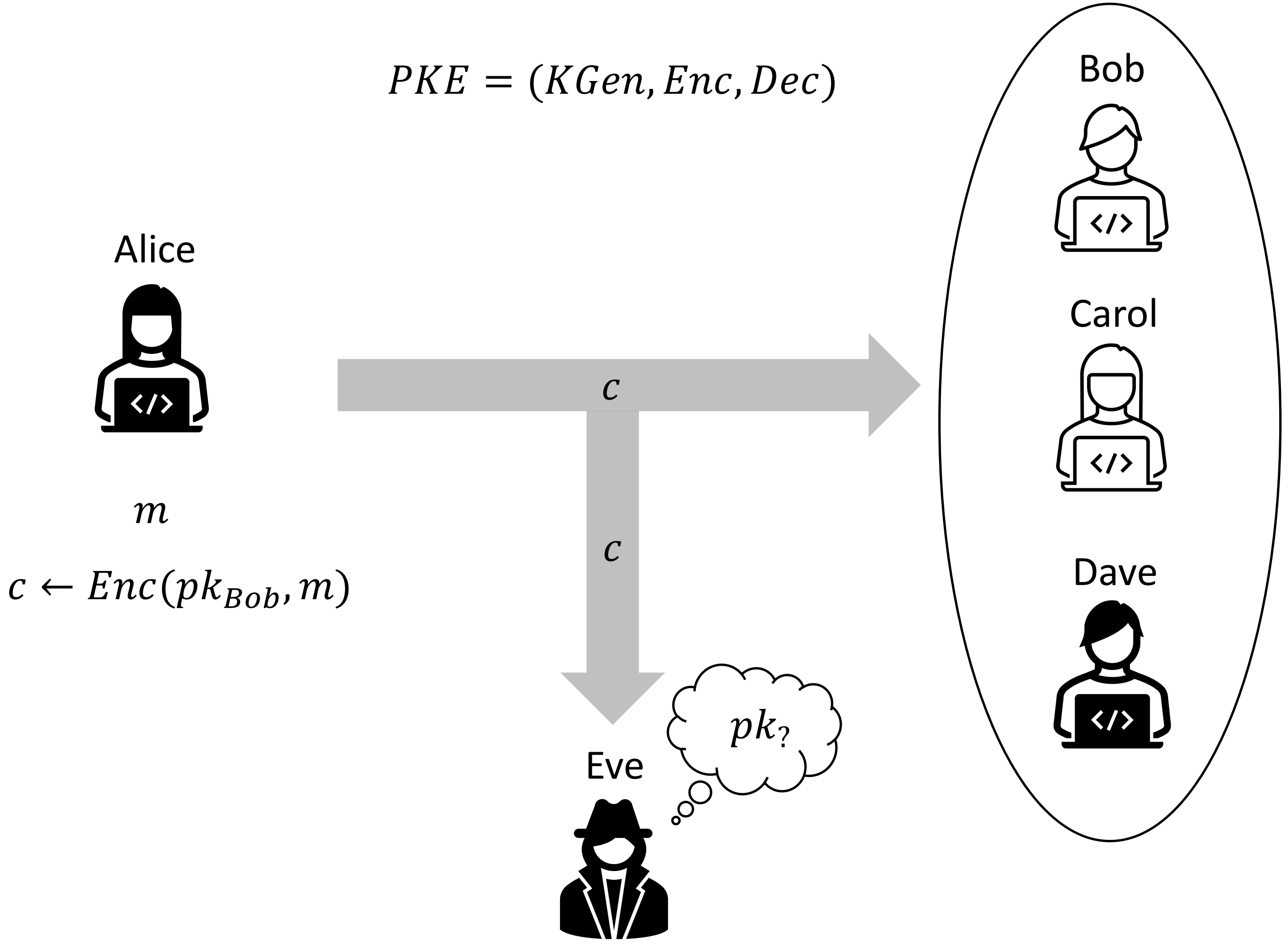
IND-CCA Security

$PKE = (KGen, Enc, Dec)$



Anonymity (ANO-CCA security)

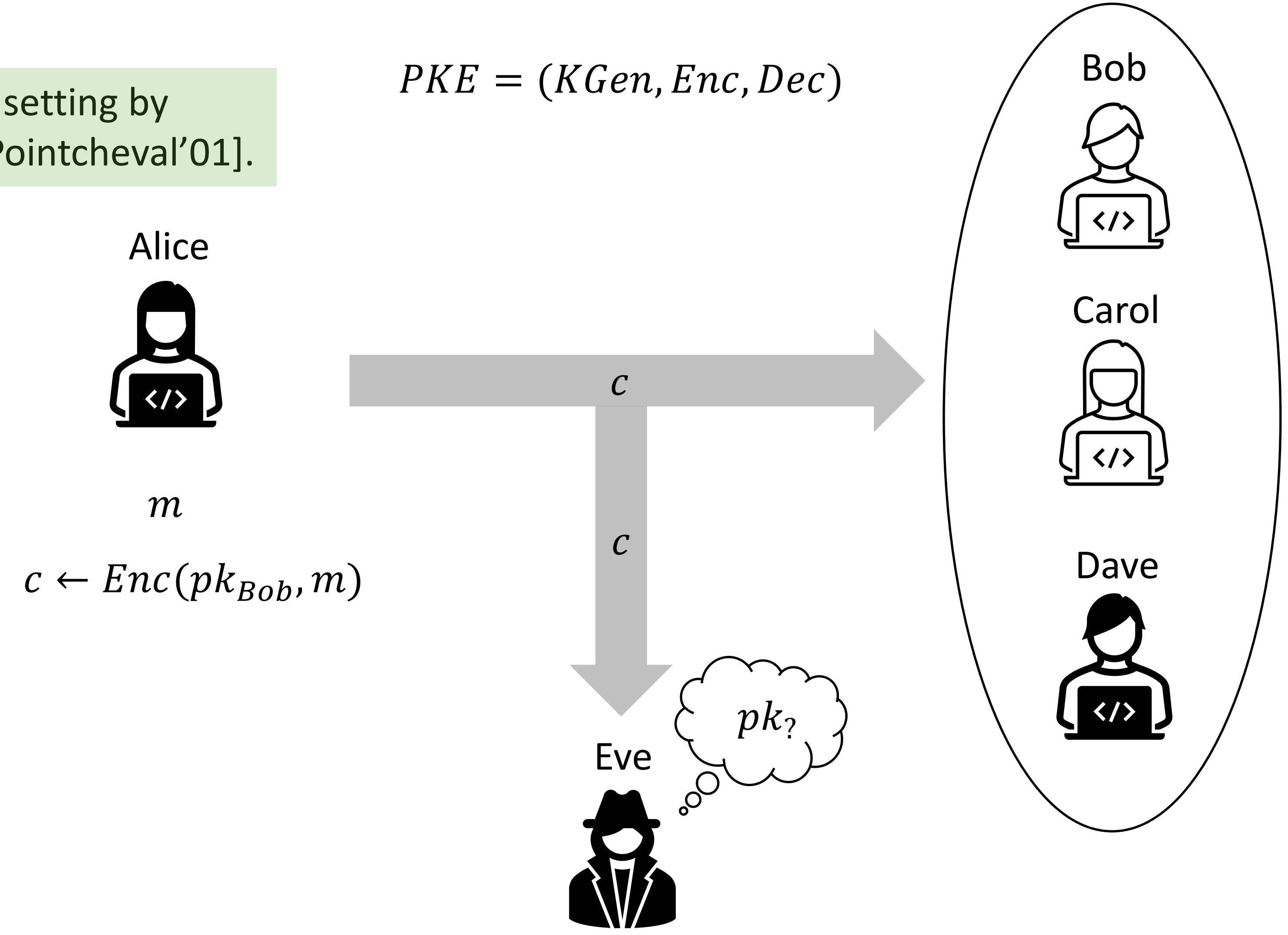
$PKE = (KGen, Enc, Dec)$



Anonymity (ANO-CCA security)

Formalized in a public-key setting by [Bellare-Boldyreva-Desai-Pointcheval'01].

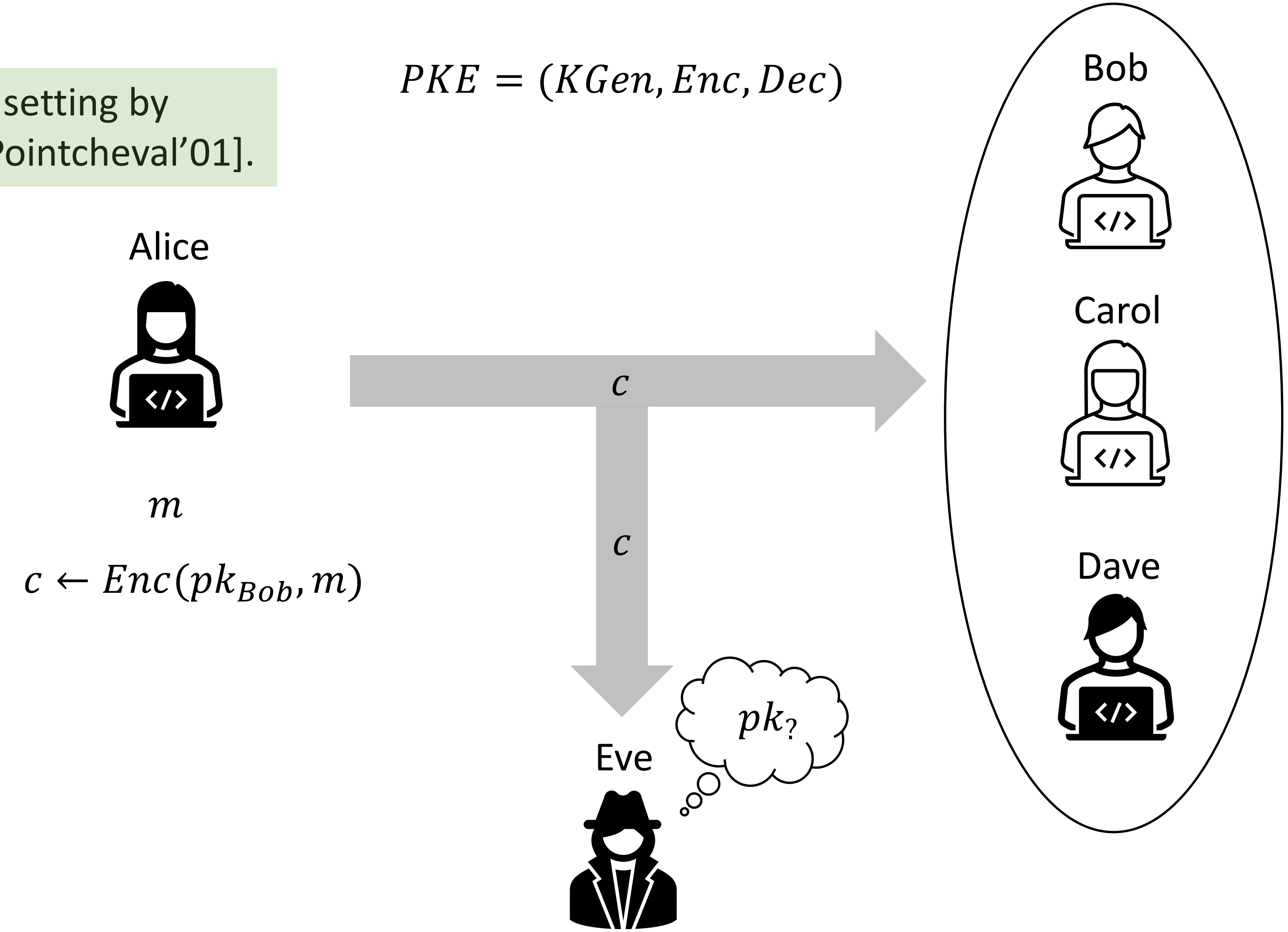
$PKE = (KGen, Enc, Dec)$



Anonymity (ANO-CCA security)

Formalized in a public-key setting by [Bellare-Boldyreva-Desai-Pointcheval'01].

$PKE = (KGen, Enc, Dec)$

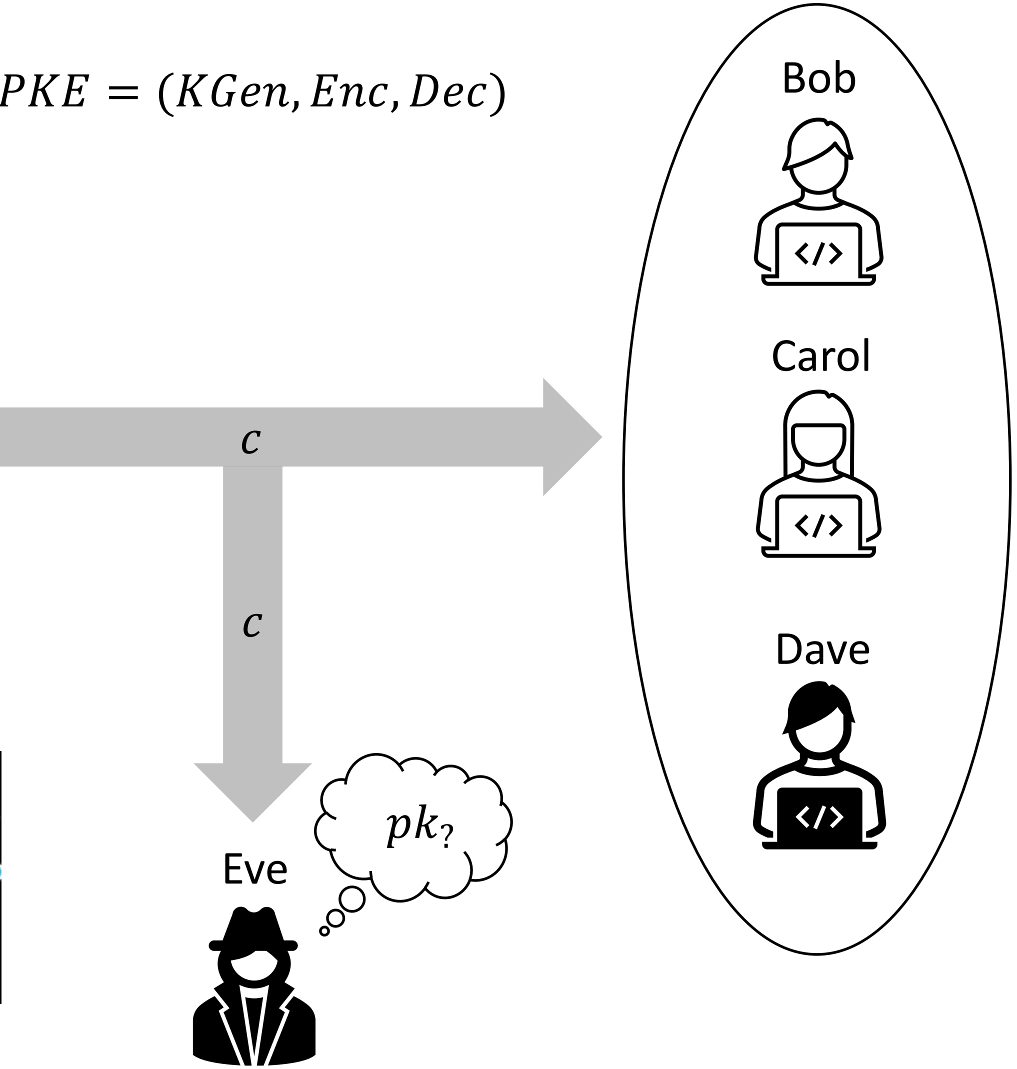


(Image taken from https://z.cash)

Anonymity (ANO-CCA security)

Formalized in a public-key setting by [Bellare-Boldyreva-Desai-Pointcheval'01].

$PKE = (KGen, Enc, Dec)$



m

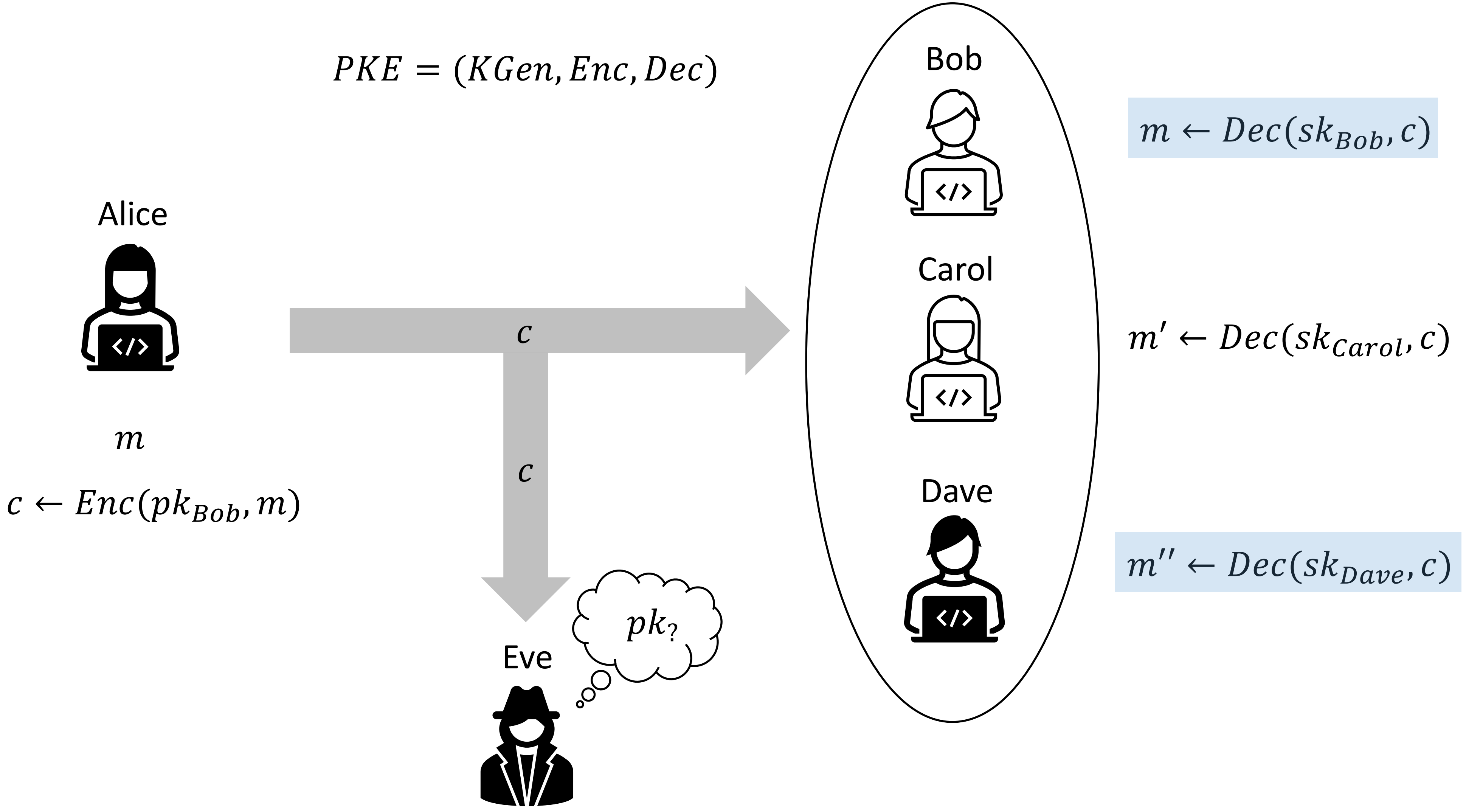
$c \leftarrow Enc(pk_{Bob}, m)$

(Image taken from https://z.cash)

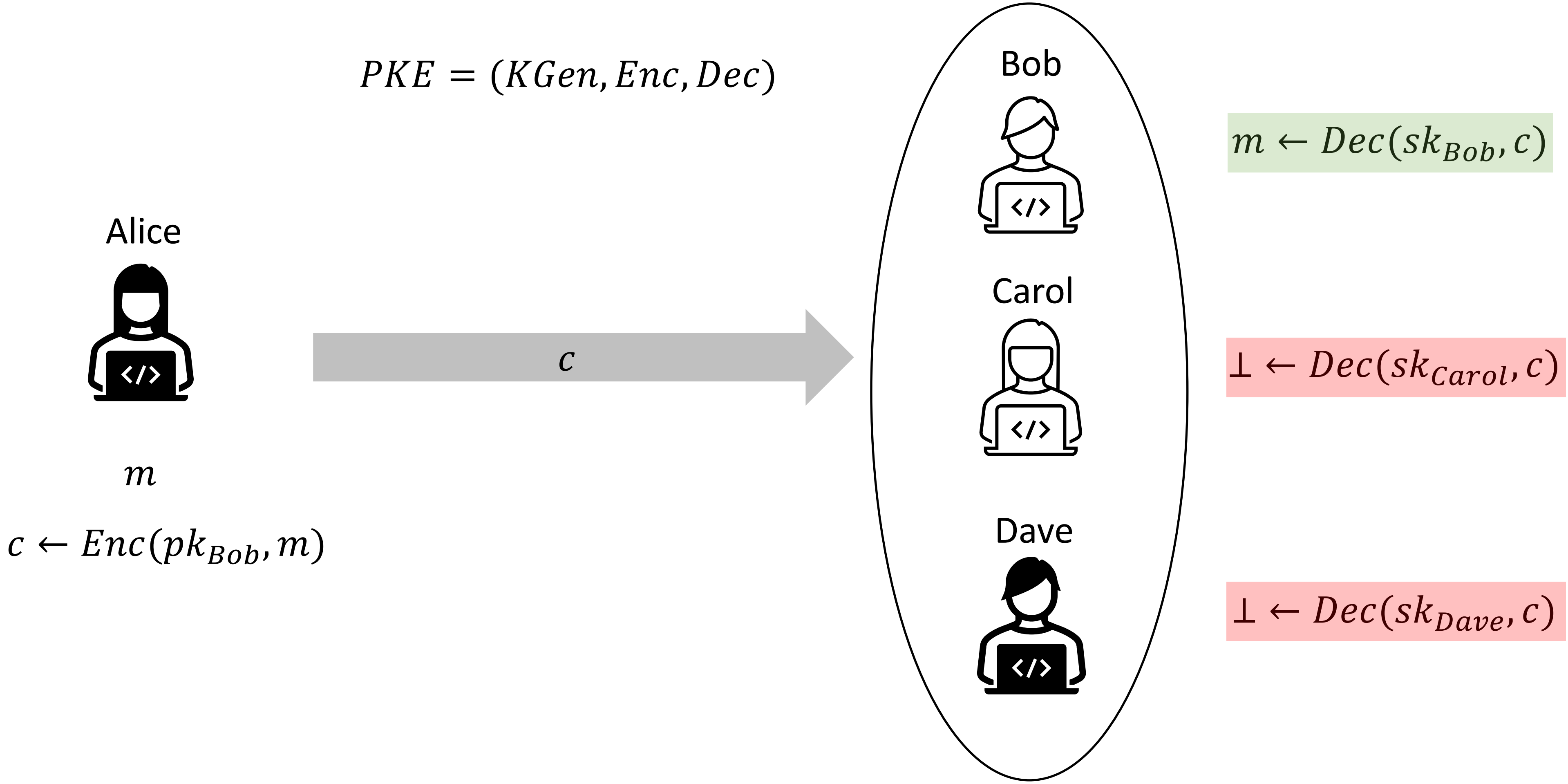
(Image taken from https://digiday.com/marketing/ad-buyers-programmatic-auction/)

Anonymity (ANO-CCA security)

$PKE = (KGen, Enc, Dec)$



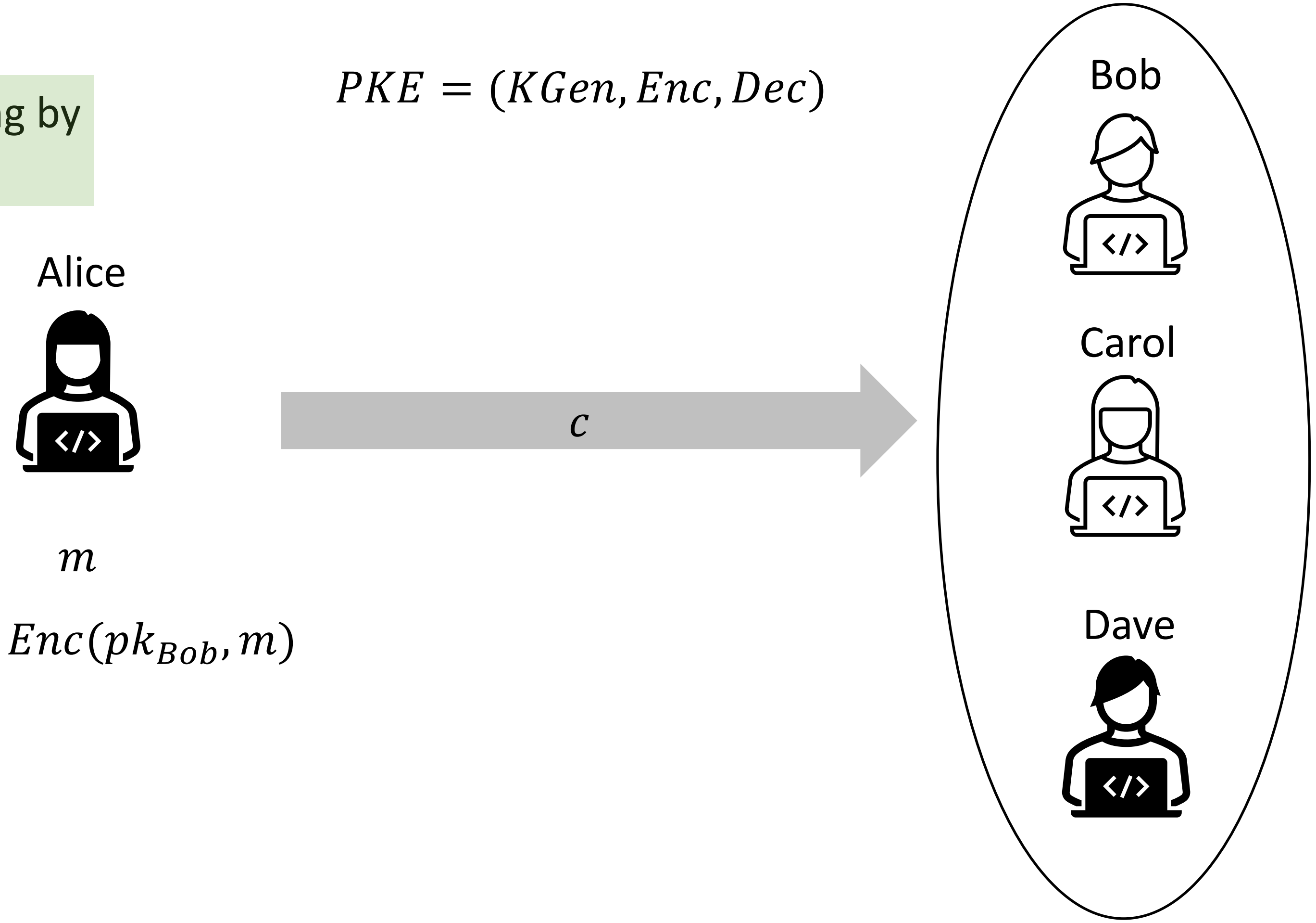
Robustness (SROB-CCA security)



Robustness (SROB-CCA security)

Formalized in a public-key setting by [Abdalla-Bellare-Neven'10].

$PKE = (KGen, Enc, Dec)$



m

$c \leftarrow Enc(pk_{Bob}, m)$

$m \leftarrow Dec(sk_{Bob}, c)$

$\perp \leftarrow Dec(sk_{Carol}, c)$

$\perp \leftarrow Dec(sk_{Dave}, c)$

KEM-DEM Paradigm

<u>Public-Key Encryption/KEMs</u>

Classic McEliece

CRYSTALS-KYBER

NTRU

SABER

KEM-DEM Paradigm

Public-Key Encryption/KEMs

Classic McEliece

CRYSTALS-KYBER

NTRU

SABER

$PKE = (KGen, Enc, Dec)$

IND-CCA secure

KEM-DEM Paradigm

Public-Key Encryption/KEMs

Classic McEliece

CRYSTALS-KYBER

NTRU

SABER

$KEM = (KGen, Encap, Decap)$

$PKE = (KGen, Enc, Dec)$

KEM

PKE

IND-CCA secure

IND-CCA secure

KEM-DEM Paradigm

Public-Key Encryption/KEMs

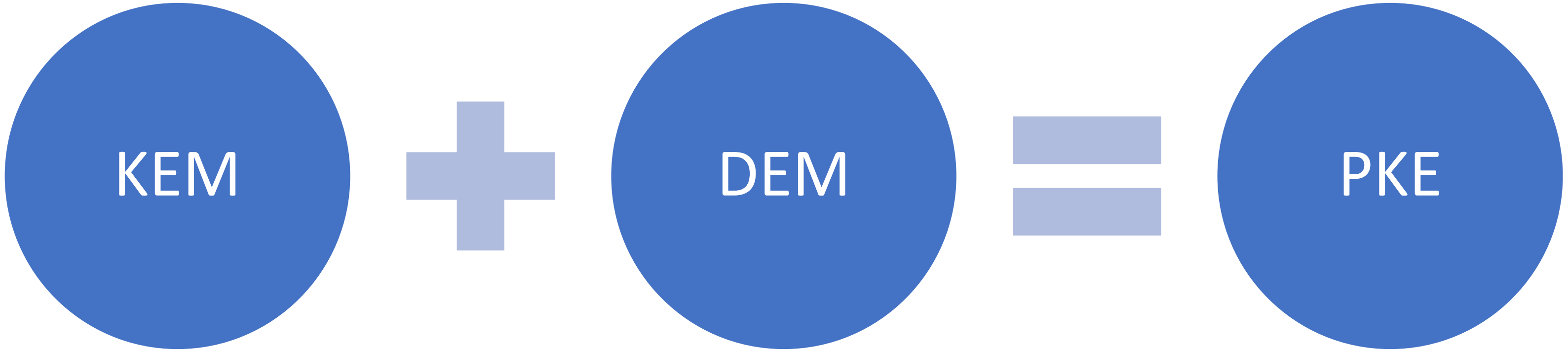
Classic McEliece

CRYSTALS-KYBER

NTRU

SABER

$KEM = (KGen, Encap, Decap)$ $DEM = (Enc^{sym}, Dec^{sym})$ $PKE = (KGen, Enc, Dec)$



IND-CCA secure

(one-time) IND-CCA secure

IND-CCA secure

KEM-DEM Paradigm

Public-Key Encryption/KEMs

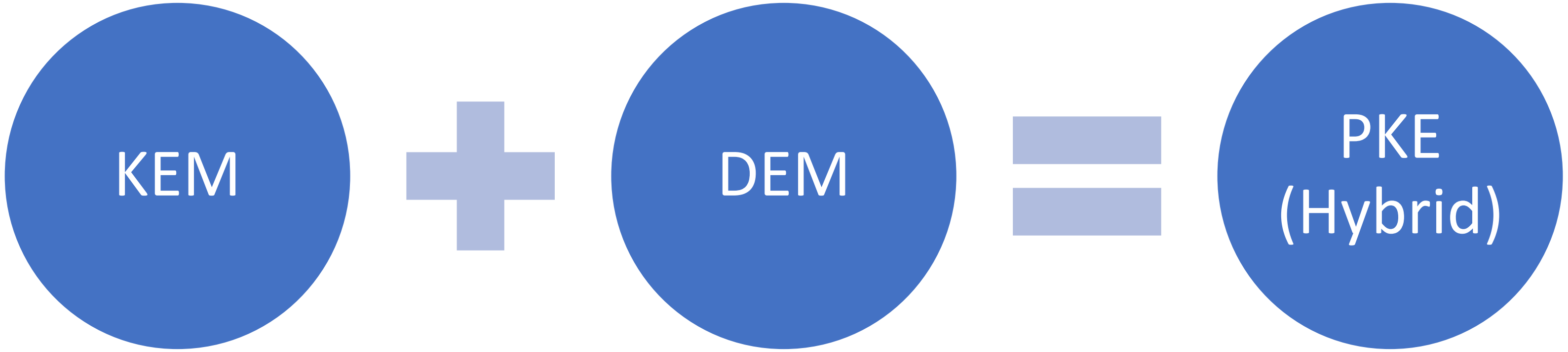
Classic McEliece

CRYSTALS-KYBER

NTRU

SABER

$KEM = (KGen, Encap, Decap)$ $DEM = (Enc^{sym}, Dec^{sym})$ $PKE = (KGen, Enc, Dec)$



IND-CCA secure

(one-time) IND-CCA secure

IND-CCA secure

KEM-DEM Paradigm

Public-Key Encryption/KEMs

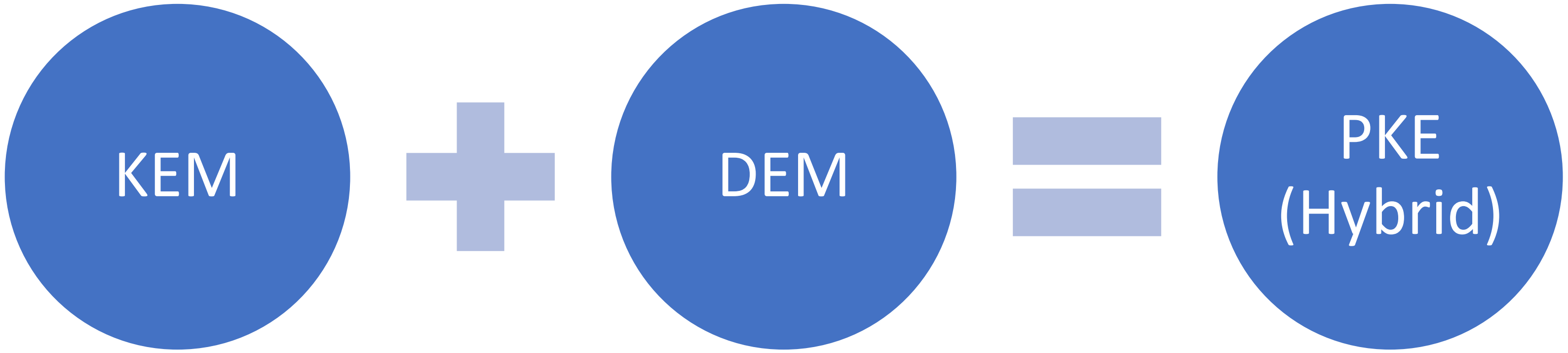
Classic McEliece

CRYSTALS-KYBER

NTRU

SABER

$KEM = (KGen, Encap, Decap)$ $DEM = (Enc^{sym}, Dec^{sym})$ $PKE = (KGen, Enc, Dec)$



$(c_{KEM}, k) \leftarrow Encap(pk_{Bob})$ $c_{DEM} \leftarrow Enc^{sym}(k, m)$ $(c_{KEM}, c_{DEM}) \leftarrow Enc(pk_{Bob}, m)$

IND-CCA secure | (one-time) IND-CCA secure | IND-CCA secure

KEM-DEM Paradigm

Public-Key Encryption/KEMs

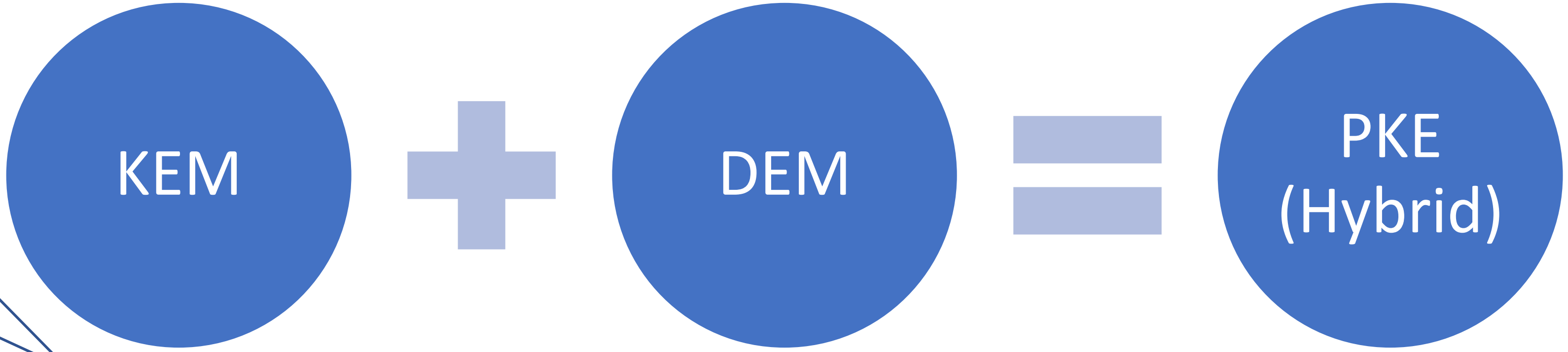
Classic McEliece

CRYSTALS-KYBER

NTRU

SABER

$KEM = (KGen, Encap, Decap)$ $DEM = (Enc^{sym}, Dec^{sym})$ $PKE = (KGen, Enc, Dec)$

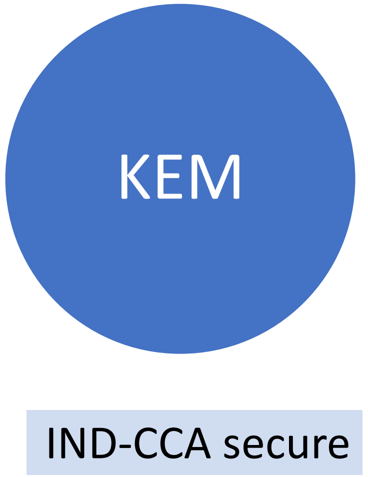


Indistinguishable from a uniformly random DEM key k'

$(c_{KEM}, k) \leftarrow Encap(pk_{Bob})$ $c_{DEM} \leftarrow Enc^{sym}(k, m)$ $(c_{KEM}, c_{DEM}) \leftarrow Enc(pk_{Bob}, m)$

IND-CCA secure | (one-time) IND-CCA secure | IND-CCA secure

Fujisaki-Okamoto Transformation



Fujisaki-Okamoto Transformation

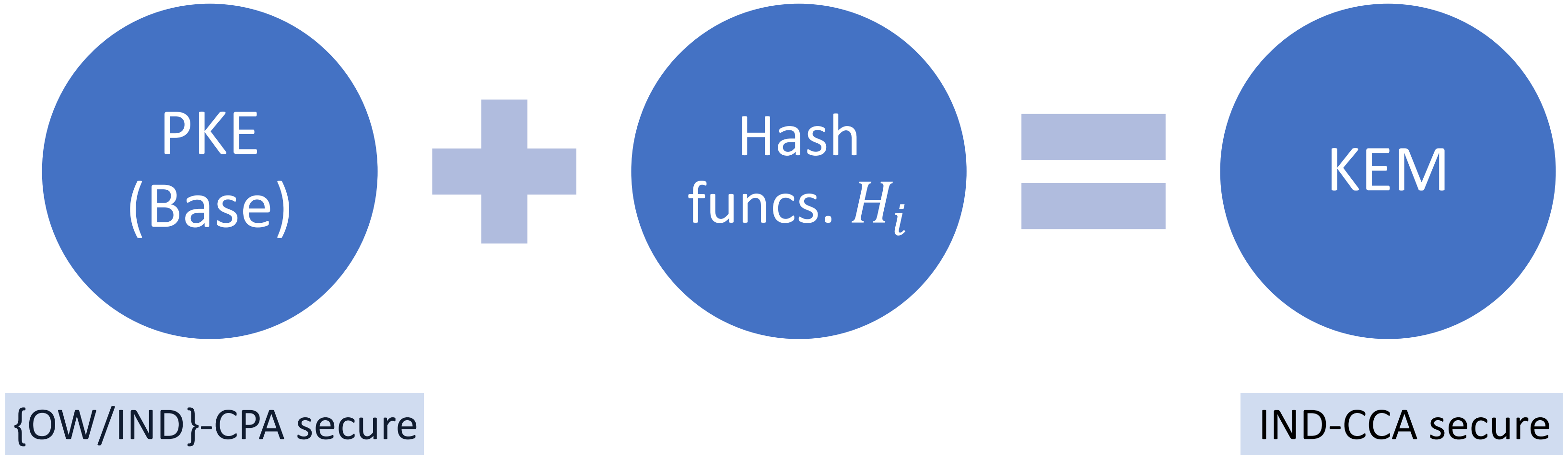
PKE
(Base)

{OW/IND}-CPA secure

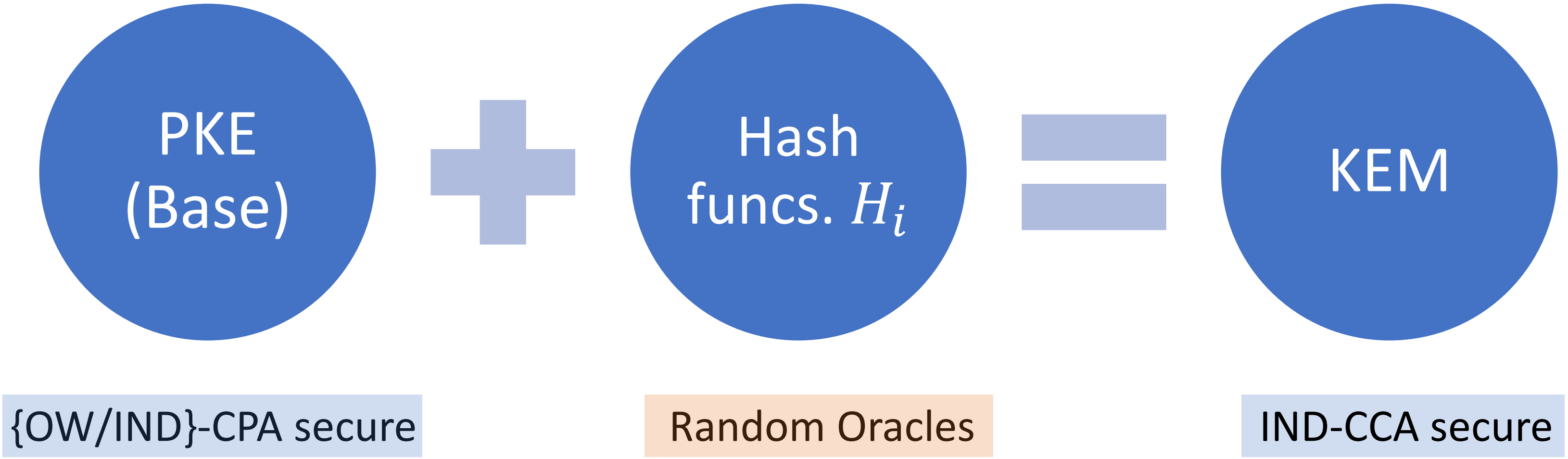
KEM

IND-CCA secure

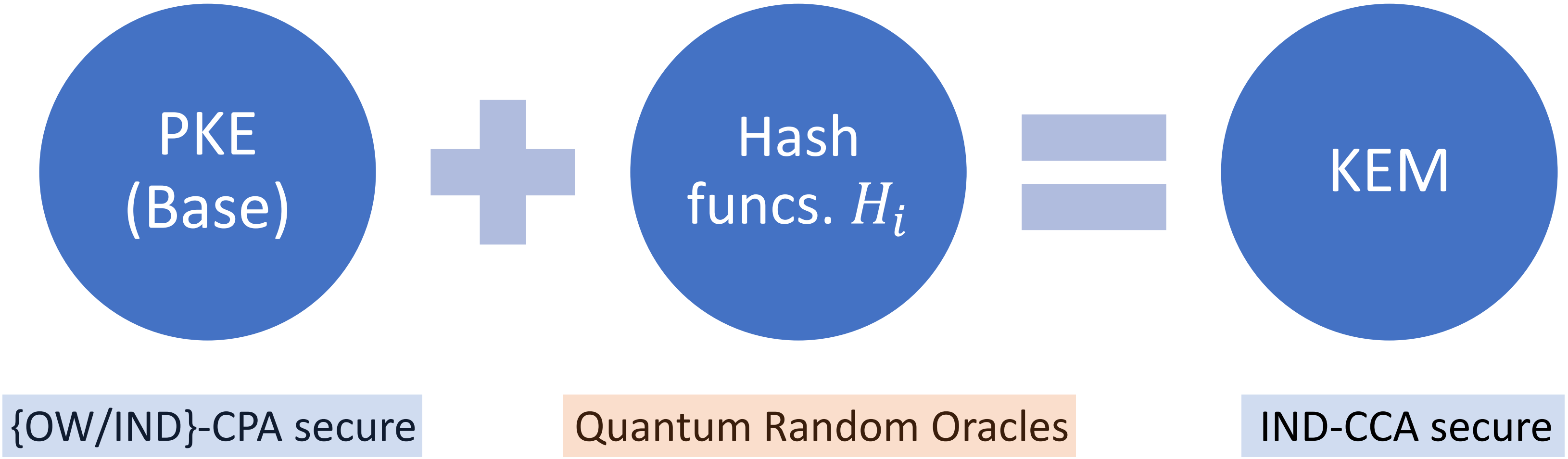
Fujisaki-Okamoto Transformation



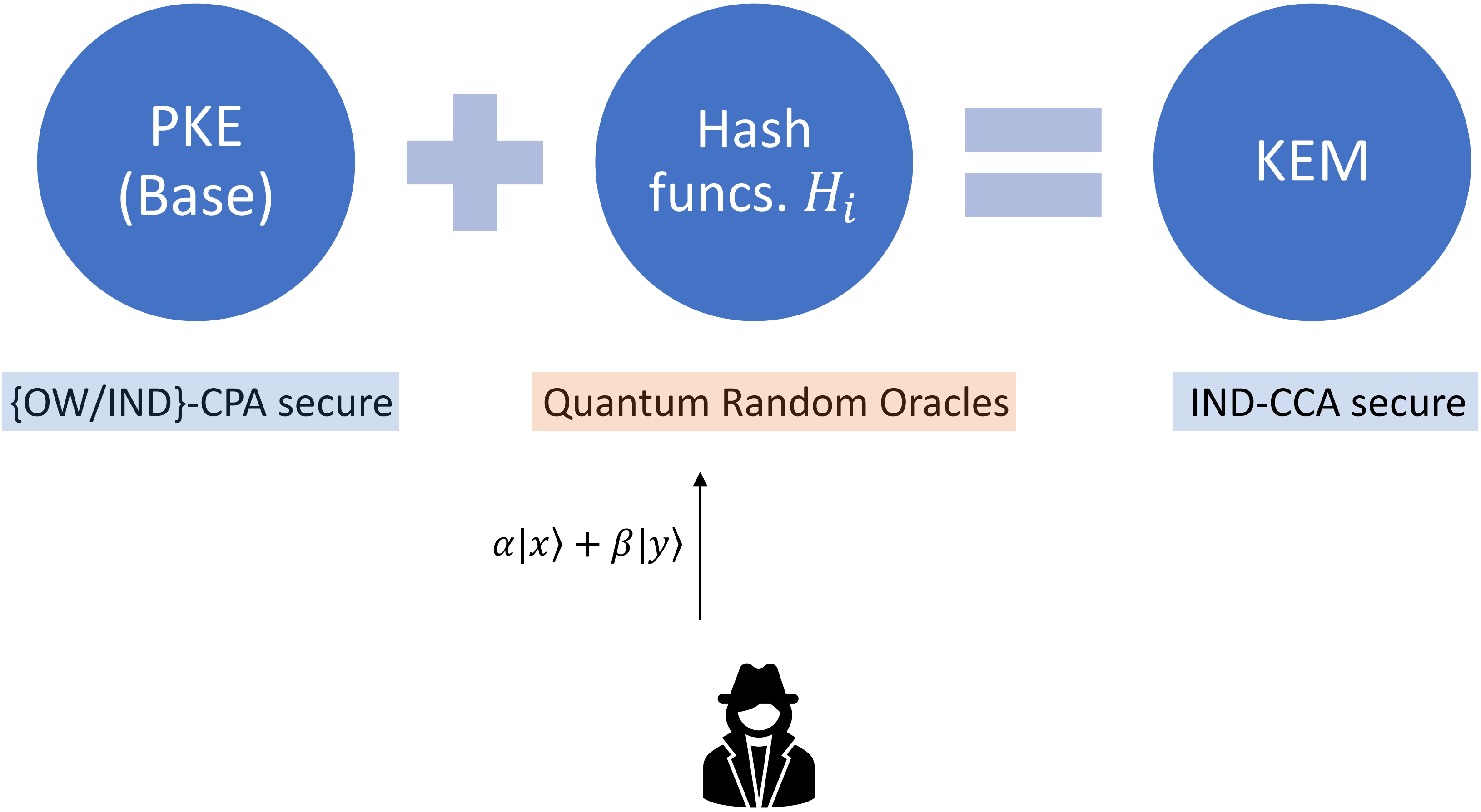
Fujisaki-Okamoto Transformation



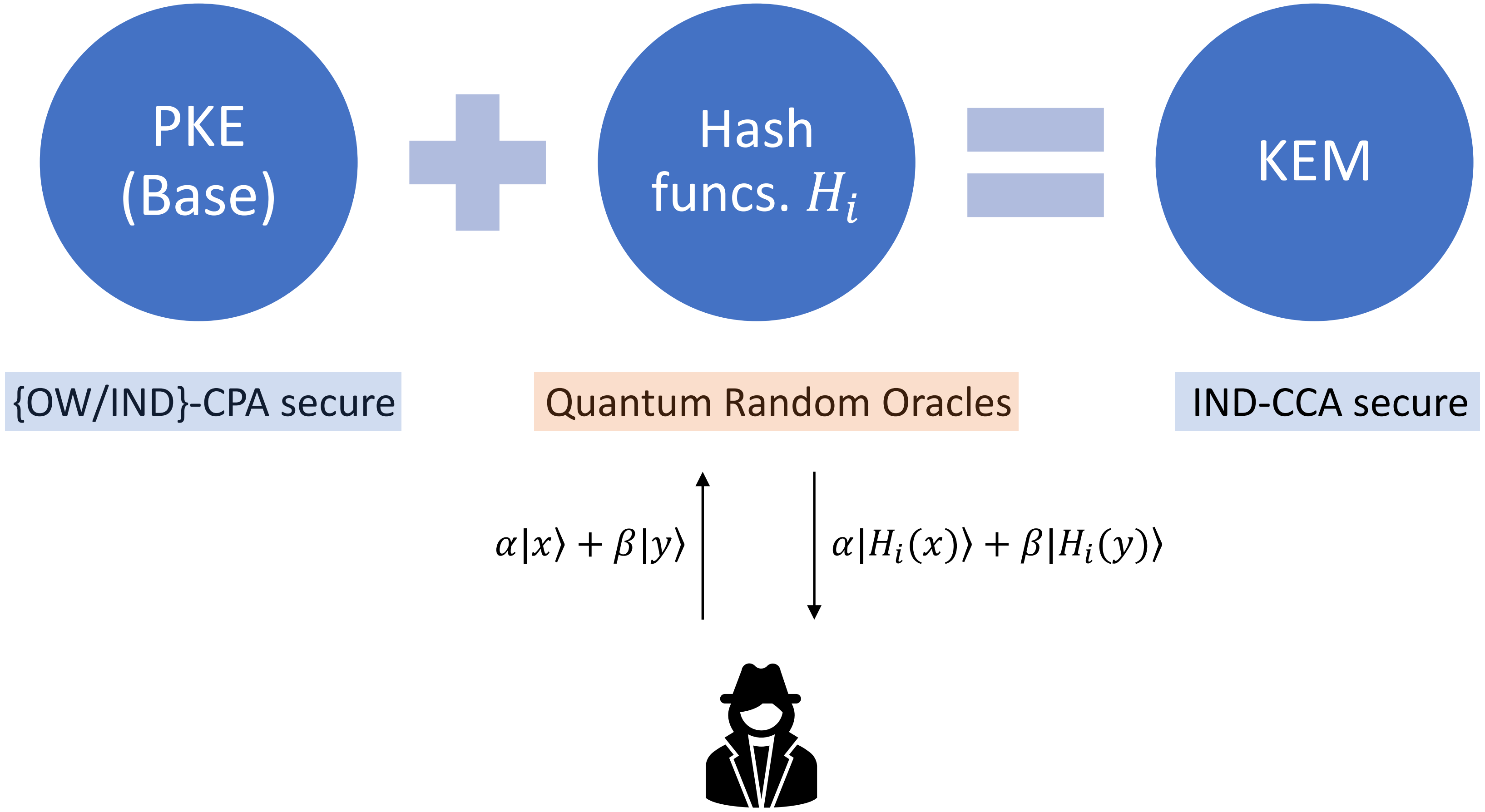
Fujisaki-Okamoto Transformation



Fujisaki-Okamoto Transformation



Fujisaki-Okamoto Transformation



Fujisaki-Okamoto Transformation

Classic McEliece

CRYSTALS-KYBER

SABER

NTRU

Fujisaki-Okamoto Transformation

Classic McEliece

CRYSTALS-KYBER

SABER

NTRU

Fujisaki-Okamoto Transformation

Classic McEliece

CRYSTALS-KYBER

SABER

KGen'	$\mathsf{Encap}(\mathsf{pk})$	$\mathsf{Decap}(\mathsf{sk}', c)$
$(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KGen}$	$m \leftarrow_{\$} \mathcal{M}$	Parse $\mathsf{sk}' = (\mathsf{sk}, s)$
$s \leftarrow_{\$} \mathcal{M}$	$c \leftarrow \mathsf{Enc}(\mathsf{pk}, m; G(m))$	$m' \leftarrow \mathsf{Dec}(\mathsf{sk}, c)$
$\mathsf{sk}' = (\mathsf{sk}, s)$	$k \leftarrow H(m, c)$	$c' \leftarrow \mathsf{Enc}(\mathsf{pk}, m'; G(m'))$
return $(\mathsf{pk}, \mathsf{sk}')$	**return** (c, k)	**if** $c' = c$ **then**
		return $H(m', c)$
		else return $H(s, c)$

The KEM $\mathsf{FO}^{\not\perp}[\mathsf{PKE}, G, H]$.

NTRU

(Image taken from *https://eprint.iacr.org/2021/708.pdf* [Grubbs-Maram-Paterson'21])

Fujisaki-Okamoto Transformation

Classic McEliece
CRYSTALS-KYBER
SABER

KGen'	$\mathsf{Encap}(\mathsf{pk})$	$\mathsf{Decap}(\mathsf{sk}', c)$
$(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KGen}$	$m \leftarrow_{\$} \mathcal{M}$	Parse $\mathsf{sk}' = (\mathsf{sk}, s)$
$s \leftarrow_{\$} \mathcal{M}$	$c \leftarrow \mathsf{Enc}(\mathsf{pk}, m; G(m))$	$m' \leftarrow \mathsf{Dec}(\mathsf{sk}, c)$
$\mathsf{sk}' = (\mathsf{sk}, s)$	$k \leftarrow H(m, c)$	$c' \leftarrow \mathsf{Enc}(\mathsf{pk}, m'; G(m'))$
return $(\mathsf{pk}, \mathsf{sk}')$	**return** (c, k)	**if** $c' = c$ **then**
		return $H(m', c)$
		else return $H(s, c)$

The KEM $\mathsf{FO}^{\not\perp}[\mathsf{PKE}, G, H]$.

NTRU

KGen'	$\mathsf{Encap}(\mathsf{pk})$	$\mathsf{Decap}(\mathsf{sk}', c)$
$(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KGen}$	$m \leftarrow_{\$} \mathcal{M}$	Parse $\mathsf{sk}' = (\mathsf{sk}, s)$
$s \leftarrow_{\$} \mathcal{M}$	$c \leftarrow \mathsf{Enc}(\mathsf{pk}, m; G(m))$	$m' \leftarrow \mathsf{Dec}(\mathsf{sk}, c)$
$\mathsf{sk}' = (\mathsf{sk}, s)$	$k \leftarrow H(m)$	$c' \leftarrow \mathsf{Enc}(\mathsf{pk}, m'; G(m'))$
return $(\mathsf{pk}, \mathsf{sk}')$	**return** (c, k)	**if** $c' = c$ **then**
		return $H(m')$
		else return $H(s, c)$

The KEM $\mathsf{FO}_m^{\not\perp}[\mathsf{PKE}, G, H]$.

(Image taken from *https://eprint.iacr.org/2021/708.pdf* [Grubbs-Maram-Paterson'21])

Fujisaki-Okamoto Transformation

Classic McEliece
CRYSTALS-KYBER
SABER

KGen′	Encap(pk)	Decap(sk′, c)
$(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KGen}$	$m \leftarrow_\$ \mathcal{M}$	Parse $\mathsf{sk}' = (\mathsf{sk}, s)$
$s \leftarrow_\$ \mathcal{M}$	$c \leftarrow \mathsf{Enc}(\mathsf{pk}, m; G(m))$	$m' \leftarrow \mathsf{Dec}(\mathsf{sk}, c)$
$\mathsf{sk}' = (\mathsf{sk}, s)$	$k \leftarrow H(m, c)$	$c' \leftarrow \mathsf{Enc}(\mathsf{pk}, m'; G(m'))$
return $(\mathsf{pk}, \mathsf{sk}')$	**return** (c, k)	**if** $c' = c$ **then**
		return $H(m', c)$
		else return $H(s, c)$

The KEM $\mathsf{FO}^{\not\perp}[\mathsf{PKE}, G, H]$.

NTRU

KGen′	Encap(pk)	Decap(sk′, c)
$(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KGen}$	$m \leftarrow_\$ \mathcal{M}$	Parse $\mathsf{sk}' = (\mathsf{sk}, s)$
$s \leftarrow_\$ \mathcal{M}$	$c \leftarrow \mathsf{Enc}(\mathsf{pk}, m; G(m))$	$m' \leftarrow \mathsf{Dec}(\mathsf{sk}, c)$
$\mathsf{sk}' = (\mathsf{sk}, s)$	$k \leftarrow H(m)$	$c' \leftarrow \mathsf{Enc}(\mathsf{pk}, m'; G(m'))$
return $(\mathsf{pk}, \mathsf{sk}')$	**return** (c, k)	**if** $c' = c$ **then**
		return $H(m')$
		else return $H(s, c)$

The KEM $\mathsf{FO}_m^{\not\perp}[\mathsf{PKE}, G, H]$.

(Image taken from https://eprint.iacr.org/2021/708.pdf [Grubbs-Maram-Paterson'21])

Fujisaki-Okamoto Transformation

Classic McEliece
CRYSTALS-KYBER
SABER

KGen'	$\mathsf{Encap}(\mathsf{pk})$	$\mathsf{Decap}(\mathsf{sk}', c)$
$(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KGen}$	$m \leftarrow_\$ \mathcal{M}$	Parse $\mathsf{sk}' = (\mathsf{sk}, s)$
$s \leftarrow_\$ \mathcal{M}$	$c \leftarrow \mathsf{Enc}(\mathsf{pk}, m; G(m))$	$m' \leftarrow \mathsf{Dec}(\mathsf{sk}, c)$
$\mathsf{sk}' = (\mathsf{sk}, s)$	$k \leftarrow H(m, c)$	$c' \leftarrow \mathsf{Enc}(\mathsf{pk}, m'; G(m'))$
return $(\mathsf{pk}, \mathsf{sk}')$	**return** (c, k)	**if** $c' = c$ **then**
		return $H(m', c)$
		else return $H(s, c)$

The KEM $\mathsf{FO}^{\not\perp}[\mathsf{PKE}, G, H]$.

NTRU

[Jiang-Zhang-Chen-Wang-Ma'18] showed the IND-CCA security of KEMs obtained from these two "standard" FO transforms in the QROM.

KGen'	$\mathsf{Encap}(\mathsf{pk})$	$\mathsf{Decap}(\mathsf{sk}', c)$
$(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KGen}$	$m \leftarrow_\$ \mathcal{M}$	Parse $\mathsf{sk}' = (\mathsf{sk}, s)$
$s \leftarrow_\$ \mathcal{M}$	$c \leftarrow \mathsf{Enc}(\mathsf{pk}, m; G(m))$	$m' \leftarrow \mathsf{Dec}(\mathsf{sk}, c)$
$\mathsf{sk}' = (\mathsf{sk}, s)$	$k \leftarrow H(m)$	$c' \leftarrow \mathsf{Enc}(\mathsf{pk}, m'; G(m'))$
return $(\mathsf{pk}, \mathsf{sk}')$	**return** (c, k)	**if** $c' = c$ **then**
		return $H(m')$
		else return $H(s, c)$

The KEM $\mathsf{FO}_m^{\not\perp}[\mathsf{PKE}, G, H]$.

(Image taken from https://eprint.iacr.org/2021/708.pdf [Grubbs-Maram-Paterson'21])

Fujisaki-Okamoto Transformation

Classic McEliece
CRYSTALS-KYBER
SABER

6.1.2 Security in the Quantum Random Oracle Model

Jiang et al. [24] provide a security reduction against a quantum adversary in the quantum random oracle model from IND-CCA security to OW-CPA security. IND-CPA with a

19

NTRU

[Jiang-Zhang-Chen-Wang-Ma'18] showed the IND-CCA security of KEMs obtained from these two "standard" FO transforms in the QROM.

KGen'	$\mathsf{Encap}(\mathsf{pk})$	$\mathsf{Decap}(\mathsf{sk}', c)$
$(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KGen}$	$m \leftarrow_\$ \mathcal{M}$	Parse $\mathsf{sk}' = (\mathsf{sk}, s)$
$s \leftarrow_\$ \mathcal{M}$	$c \leftarrow \mathsf{Enc}(\mathsf{pk}, m; G(m))$	$m' \leftarrow \mathsf{Dec}(\mathsf{sk}, c)$
$\mathsf{sk}' = (\mathsf{sk}, s)$	$k \leftarrow H(m, c)$	$c' \leftarrow \mathsf{Enc}(\mathsf{pk}, m'; G(m'))$
return $(\mathsf{pk}, \mathsf{sk}')$	**return** (c, k)	**if** $c' = c$ **then**
		return $H(m', c)$
		else return $H(s, c)$

The KEM $\mathsf{FO}^{\not\perp}[\mathsf{PKE}, G, H]$.

KGen'	$\mathsf{Encap}(\mathsf{pk})$	$\mathsf{Decap}(\mathsf{sk}', c)$
$(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KGen}$	$m \leftarrow_\$ \mathcal{M}$	Parse $\mathsf{sk}' = (\mathsf{sk}, s)$
$s \leftarrow_\$ \mathcal{M}$	$c \leftarrow \mathsf{Enc}(\mathsf{pk}, m; G(m))$	$m' \leftarrow \mathsf{Dec}(\mathsf{sk}, c)$
$\mathsf{sk}' = (\mathsf{sk}, s)$	$k \leftarrow H(m)$	$c' \leftarrow \mathsf{Enc}(\mathsf{pk}, m'; G(m'))$
return $(\mathsf{pk}, \mathsf{sk}')$	**return** (c, k)	**if** $c' = c$ **then**
		return $H(m')$
		else return $H(s, c)$

The KEM $\mathsf{FO}_m^{\not\perp}[\mathsf{PKE}, G, H]$.

(Image taken from *https://eprint.iacr.org/2021/708.pdf* [Grubbs-Maram-Paterson'21])

Anonymity from FO transforms

(Informal) **Theorem** [Grubbs-Maram-Paterson’21]:

Hybrid PKE schemes obtained from $\mathsf{FO}^{\not\perp}$ *KEMs via the generic KEM-DEM composition are also ANO-CCA secure in the QROM.**

Anonymity from FO transforms

(Informal) **Theorem** [Grubbs-Maram-Paterson'21]:

*Hybrid PKE schemes obtained from $\mathsf{FO}^{\not\perp}$ KEMs via the generic KEM-DEM composition are also ANO-CCA secure in the QROM.**

*(*Provided the base PKE scheme satisfies some additional mild security properties.)*

NTRU

(Informal) **Theorem** [Grubbs-Maram-Paterson'21]:

*Hybrid PKE schemes obtained from $\mathsf{FO}^{\not\perp}$ KEMs via the generic KEM-DEM composition are also ANO-CCA secure in the QROM.**

*(*Provided the base PKE scheme satisfies some additional mild security properties.)*

Status of NTRU (which uses a close variant of $\mathsf{FO}_m^{\not\perp}$) with respect to anonymity and robustness properties is open.

Classic McEliece (CM)

(Informal) **Theorem** [Grubbs-Maram-Paterson'21]:

Hybrid PKE schemes obtained from $\mathsf{FO}^{\not\perp}$ *KEMs via the generic KEM-DEM composition are also ANO-CCA secure in the QROM.**

*(*Provided the base PKE scheme satisfies some additional mild security properties.)*

The base PKE scheme needs to *randomized* (specifically, γ-spread).

Classic McEliece (CM)

(Informal) **Theorem** [Grubbs-Maram-Paterson'21]:

Hybrid PKE schemes obtained from $\mathsf{FO}^{\not\perp}$ *KEMs via the generic KEM-DEM composition are also ANO-CCA secure in the QROM.**

*(*Provided the base PKE scheme satisfies some additional mild security properties.)*

The base PKE scheme needs to *randomized* (specifically, γ-spread).

CM uses a *deterministic* base PKE scheme.

Classic McEliece (CM)

2.2.3 Encoding subroutine

The following algorithm ENCODE takes two inputs: a weight-t column vector $e \in \mathbb{F}_2^n$; and a public key T, i.e., an $(n-k) \times k$ matrix over $\mathbb{F}_2$. The algorithm output ENCODE(e, T) is a vector $C_0 \in \mathbb{F}_2^{n-k}$. Here is the algorithm:

1. Define $H = (I_{n-k} \mid T)$.
2. Compute and return $C_0 = He \in \mathbb{F}_2^{n-k}$.

(Image taken from *https://classic.mceliece.org/nist/mceliece-20201010.pdf*)

Classic McEliece (CM)

2.2.3 Encoding subroutine

The following algorithm ENCODE takes two inputs: a weight-t column vector $e \in \mathbb{F}_2^n$; and a public key T, i.e., an $(n-k) \times k$ matrix over $\mathbb{F}_2$. The algorithm output ENCODE(e, T) is a vector $C_0 \in \mathbb{F}_2^{n-k}$. Here is the algorithm:

1. Define $H = (I_{n-k} \mid T)$.
2. Compute and return $C_0 = He \in \mathbb{F}_2^{n-k}$.

(Image taken from *https://classic.mceliece.org/nist/mceliece-20201010.pdf*)

Classic McEliece (CM)

2.2.3 Encoding subroutine

The following algorithm ENCODE takes two inputs: a weight-t column vector $e \in \mathbb{F}_2^n$; and a public key T, i.e., an $(n-k) \times k$ matrix over $\mathbb{F}_2$. The algorithm output ENCODE(e, T) is a vector $C_0 \in \mathbb{F}_2^{n-k}$. Here is the algorithm:

1. Define $H = (I_{n-k} \mid T)$.
2. Compute and return $C_0 = He \in \mathbb{F}_2^{n-k}$.

(Image taken from https://classic.mceliece.org/nist/mceliece-20201010.pdf)

Classic McEliece (CM)

2.2.3 Encoding subroutine

The following algorithm ENCODE takes two inputs: a weight-t column vector $e \in \mathbb{F}_2^n$; and a public key T, i.e., an $(n-k) \times k$ matrix over $\mathbb{F}_2$. The algorithm output $\text{ENCODE}(e, T)$ is a vector $C_0 \in \mathbb{F}_2^{n-k}$. Here is the algorithm:

1. Define $H = (I_{n-k} \mid T)$.
2. Compute and return $C_0 = He \in \mathbb{F}_2^{n-k}$.

(Image taken from https://classic.mceliece.org/nist/mceliece-20201010.pdf)

Classic McEliece (CM)

2.2.3 Encoding subroutine

The following algorithm ENCODE takes two inputs: a weight-t column vector $e \in \mathbb{F}_2^n$; and a public key T, i.e., an $(n-k) \times k$ matrix over $\mathbb{F}_2$. The algorithm output ENCODE(e, T) is a vector $C_0 \in \mathbb{F}_2^{n-k}$. Here is the algorithm:

1. Define $H = (I_{n-k} \mid T)$.
2. Compute and return $C_0 = He \in \mathbb{F}_2^{n-k}$.

(Image taken from https://classic.mceliece.org/nist/mceliece-20201010.pdf)

Classic McEliece (CM)

2.2.3 Encoding subroutine

The following algorithm ENCODE takes two inputs: a weight-t column vector $e \in \mathbb{F}_2^n$; and a public key T, i.e., an $(n-k) \times k$ matrix over $\mathbb{F}_2$. The algorithm output ENCODE(e, T) is a vector $C_0 \in \mathbb{F}_2^{n-k}$. Here is the algorithm:

1. Define $H = (I_{n-k} \mid T)$.
2. Compute and return $C_0 = He \in \mathbb{F}_2^{n-k}$.

Fix any "message" $e = \begin{pmatrix} e_{n-k} \\ 0^k \end{pmatrix}$:

(Image taken from *https://classic.mceliece.org/nist/mceliece-20201010.pdf*)

Classic McEliece (CM)

2.2.3 Encoding subroutine

The following algorithm ENCODE takes two inputs: a weight-t column vector $e \in \mathbb{F}_2^n$; and a public key T, i.e., an $(n-k) \times k$ matrix over $\mathbb{F}_2$. The algorithm output ENCODE(e, T) is a vector $C_0 \in \mathbb{F}_2^{n-k}$. Here is the algorithm:

1. Define $H = (I_{n-k} \mid T)$.
2. Compute and return $C_0 = He \in \mathbb{F}_2^{n-k}$.

Fix any "message" $e = \begin{pmatrix} e_{n-k} \\ 0^k \end{pmatrix}$:

- $(n - k \geq t$ in all CM parameters)

(Image taken from *https://classic.mceliece.org/nist/mceliece-20201010.pdf*)

Classic McEliece (CM)

2.2.3 Encoding subroutine

The following algorithm ENCODE takes two inputs: a weight-t column vector $e \in \mathbb{F}_2^n$; and a public key T, i.e., an $(n-k) \times k$ matrix over $\mathbb{F}_2$. The algorithm output ENCODE(e, T) is a vector $C_0 \in \mathbb{F}_2^{n-k}$. Here is the algorithm:

1. Define $H = (I_{n-k} \mid T)$.
2. Compute and return $C_0 = He \in \mathbb{F}_2^{n-k}$.

Fix any "message" $e = \binom{e_{n-k}}{0^k}$:

- ($n - k \geq t$ in all CM parameters)
- $C_0 = (I_{n-k}|T)\binom{e_{n-k}}{0^k} = e_{n-k}$ – i.e., independent of public-key T.

(Image taken from https://classic.mceliece.org/nist/mceliece-20201010.pdf)

Classic McEliece (CM)

2.2.3 Encoding subroutine

The following algorithm ENCODE takes two inputs: a weight-t column vector $e \in \mathbb{F}_2^n$; and a public key T, i.e., an $(n-k) \times k$ matrix over $\mathbb{F}_2$. The algorithm output ENCODE(e, T) is a vector $C_0 \in \mathbb{F}_2^{n-k}$. Here is the algorithm:

1. Define $H = (I_{n-k} \mid T)$.
2. Compute and return $C_0 = He \in \mathbb{F}_2^{n-k}$.

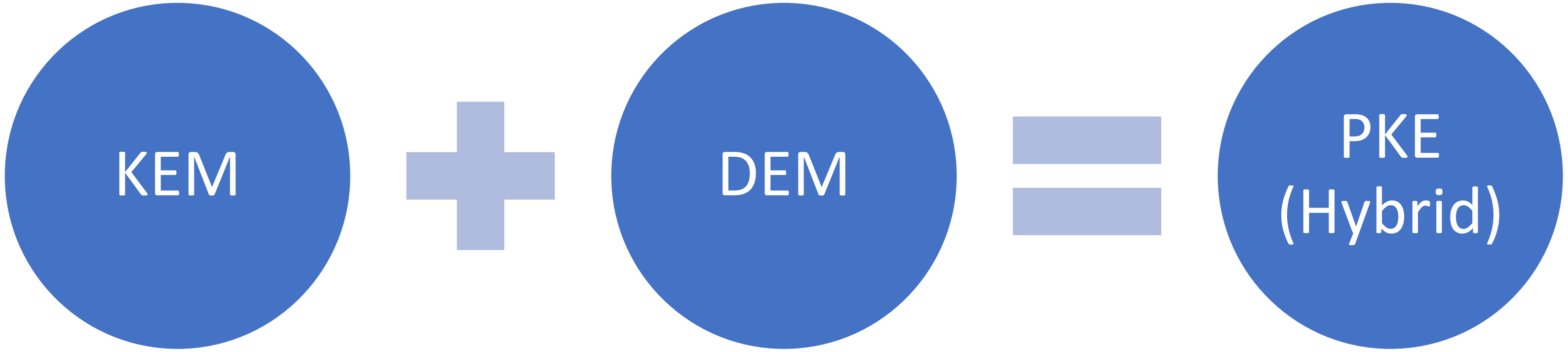
Fix any "message" $e = \binom{e_{n-k}}{0^k}$:

- $(n-k \ \geq t$ in all CM parameters)
- $C_0 = (I_{n-k}|T)\binom{e_{n-k}}{0^k} = e_{n-k}$ – i.e., independent of public-key T.
- Because of perfect correctness, C_0 must decrypt to fixed e under *any private key* of CM's base PKE scheme.

(Image taken from *https://classic.mceliece.org/nist/mceliece-20201010.pdf*)

Classic McEliece (CM)

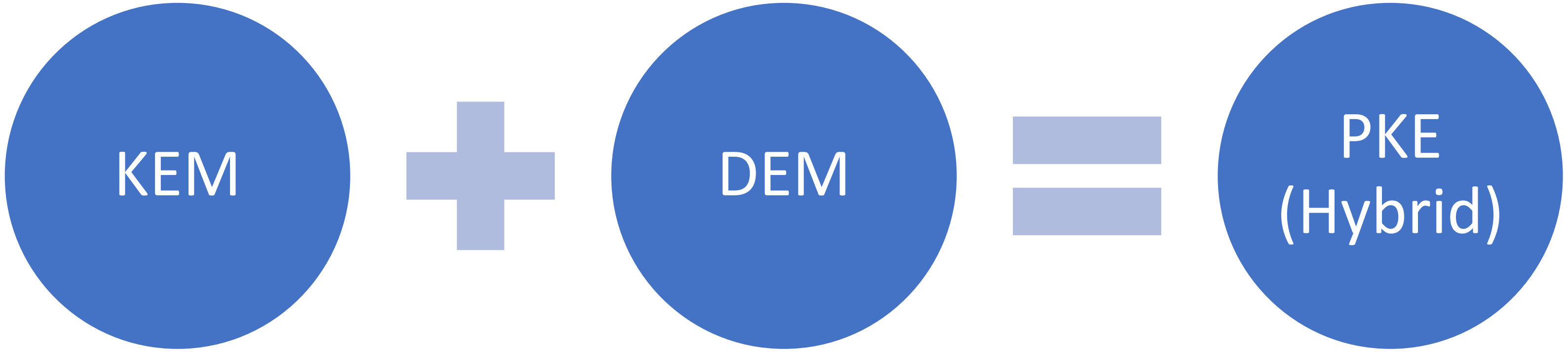
$KEM = (KGen, Encap, Decap)$ $DEM = (Enc^{sym}, Dec^{sym})$ $PKE = (KGen, Enc, Dec)$



$(c_{KEM}, k) \leftarrow Encap(pk_{Bob})$ $c_{DEM} \leftarrow Enc^{sym}(k, m)$ $(c_{KEM}, c_{DEM}) \leftarrow Enc(pk_{Bob}, m)$

Classic McEliece (CM)

$KEM = (KGen, Encap, Decap)$ $DEM = (Enc^{sym}, Dec^{sym})$ $PKE = (KGen, Enc, Dec)$



$(c_{KEM}, k) \leftarrow Encap(pk_{Bob})$ $c_{DEM} \leftarrow Enc^{sym}(k, m)$ $(c_{KEM}, c_{DEM}) \leftarrow Enc(pk_{Bob}, m)$

2.4.5 Encapsulation

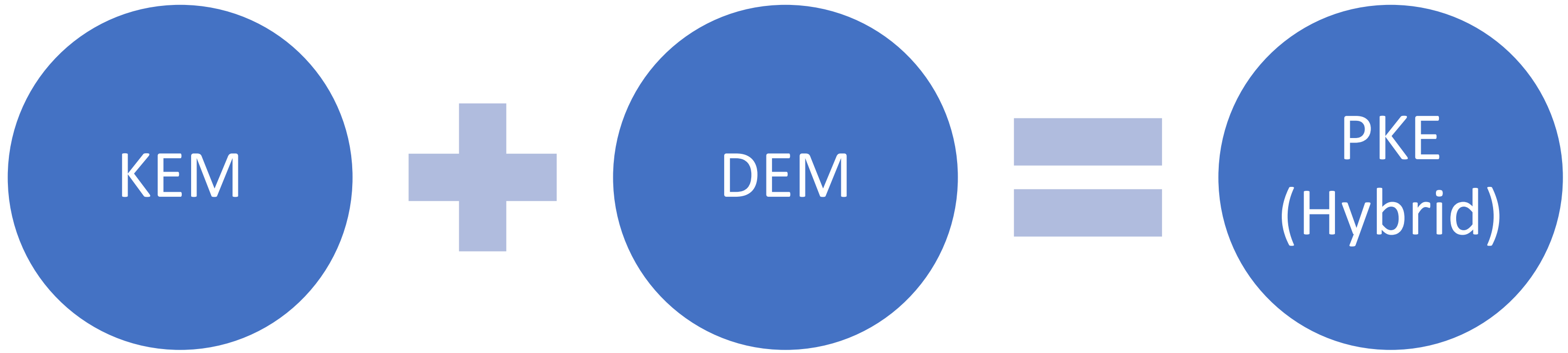
The following randomized algorithm ENCAP takes as input a public key T. It outputs a ciphertext C and a session key K. Here is the algorithm:

1. Use FIXEDWEIGHT to generate a vector $e \in \mathbb{F}_2^n$ of weight t.
2. Compute $C_0 = \text{ENCODE}(e, T)$.
3. Compute $C_1 = \mathsf{H}(2, e)$; see Section 2.5.2 for H input encodings. Put $C = (C_0, C_1)$.
4. Compute $K = \mathsf{H}(1, e, C)$; see Section 2.5.2 for H input encodings.
5. Output ciphertext C and session key K.

(Image taken from https://classic.mceliece.org/nist/mceliece-20201010.pdf)

Classic McEliece (CM)

$KEM = (KGen, Encap, Decap)$ $DEM = (Enc^{sym}, Dec^{sym})$ $PKE = (KGen, Enc, Dec)$



$(c_{KEM}, k) \leftarrow Encap(pk_{Bob})$ $c_{DEM} \leftarrow Enc^{sym}(k, m)$ $(c_{KEM}, c_{DEM}) \leftarrow Enc(pk_{Bob}, m)$

2.4.5 Encapsulation

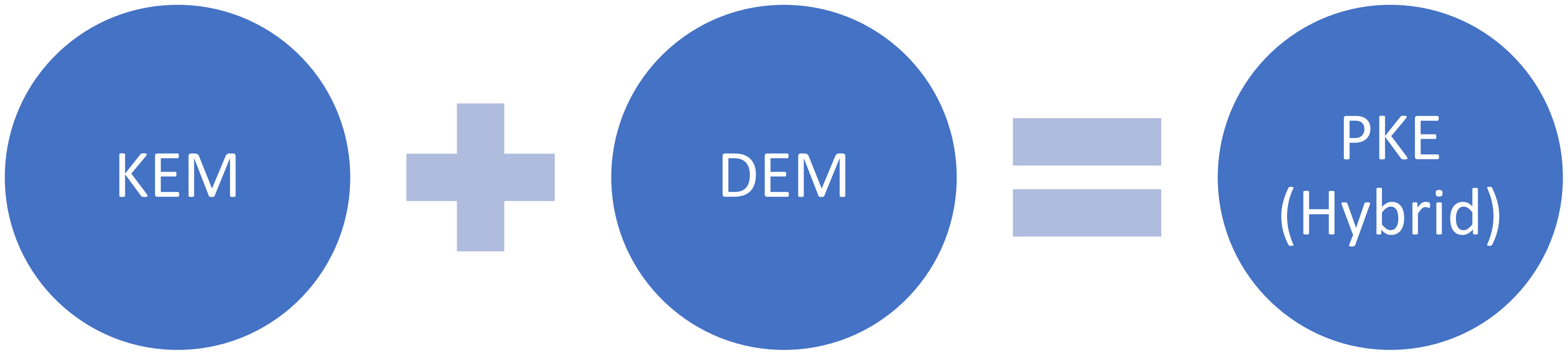
The following randomized algorithm ENCAP takes as input a public key T. It outputs a ciphertext C and a session key K. Here is the algorithm:

1. Use FIXEDWEIGHT to generate a vector $e \in \mathbb{F}_2^n$ of weight t.
2. Compute $C_0 = \text{ENCODE}(e, T)$.
3. Compute $C_1 = \mathsf{H}(2, e)$; see Section 2.5.2 for H input encodings. Put $C = (C_0, C_1)$.
4. Compute $K = \mathsf{H}(1, e, C)$; see Section 2.5.2 for H input encodings.
5. Output ciphertext C and session key K.

(Image taken from https://classic.mceliece.org/nist/mceliece-20201010.pdf)

Classic McEliece (CM)

$KEM = (KGen, Encap, Decap)$ $DEM = (Enc^{sym}, Dec^{sym})$ $PKE = (KGen, Enc, Dec)$



$(c_{KEM}, k) \leftarrow Encap(pk_{Bob})$ $c_{DEM} \leftarrow Enc^{sym}(k, m)$ $(c_{KEM}, c_{DEM}) \leftarrow Enc(pk_{Bob}, m)$

2.4.5 Encapsulation

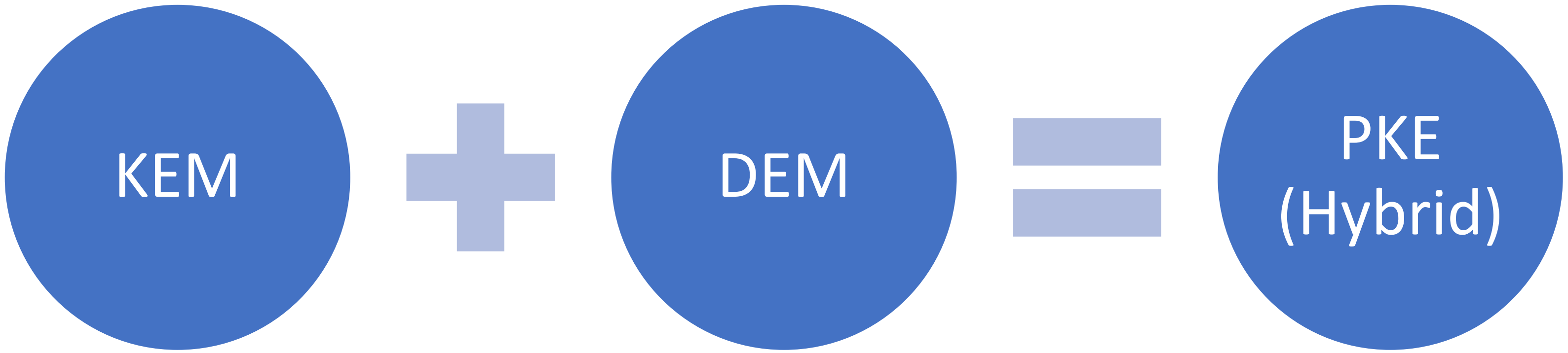
The following randomized algorithm ENCAP takes as input a public key T. It outputs a ciphertext C and a session key K. Here is the algorithm:

1. Use FIXEDWEIGHT to generate a vector $e \in \mathbb{F}_2^n$ of weight t.
2. Compute $C_0 = \text{ENCODE}(e, T)$.
3. Compute $C_1 = \mathsf{H}(2, e)$; see Section 2.5.2 for H input encodings. Put $C = (C_0, C_1)$.
4. Compute $K = \mathsf{H}(1, e, C)$; see Section 2.5.2 for H input encodings.
5. Output ciphertext C and session key K.

(Image taken from https://classic.mceliece.org/nist/mceliece-20201010.pdf)

Classic McEliece (CM)

$KEM = (KGen, Encap, Decap)$ $DEM = (Enc^{sym}, Dec^{sym})$ $PKE = (KGen, Enc, Dec)$



$(c_{KEM}, k) \leftarrow Encap(pk_{Bob})$ $c_{DEM} \leftarrow Enc^{sym}(k, m)$ $(c_{KEM}, c_{DEM}) \leftarrow Enc(pk_{Bob}, m)$

2.4.5 Encapsulation

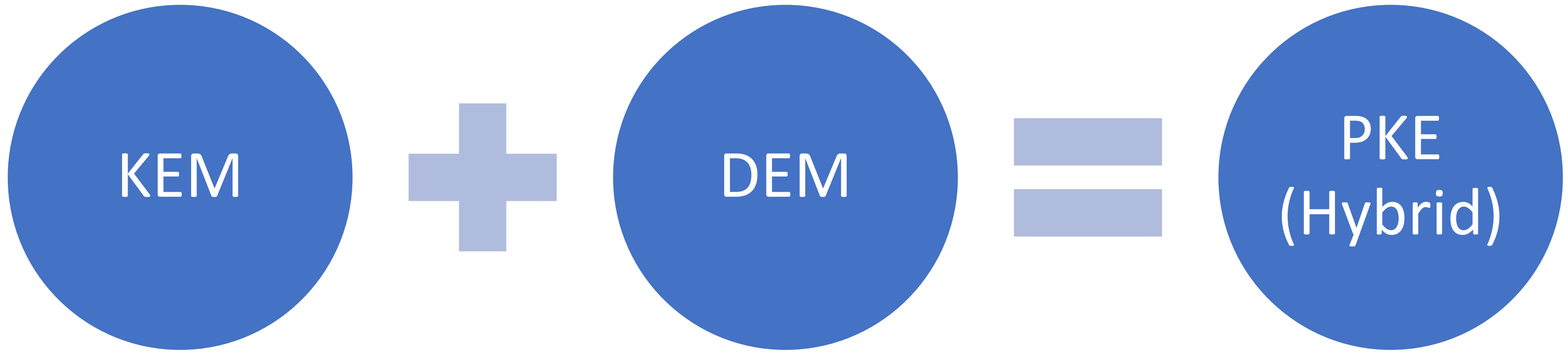
The following randomized algorithm ENCAP takes as input a public key T. It outputs a ciphertext C and a session key K. Here is the algorithm:

1. Use FIXEDWEIGHT to generate a vector $e \in \mathbb{F}_2^n$ of weight t.
2. Compute $C_0 = \text{ENCODE}(e, T)$.
3. Compute $C_1 = \mathsf{H}(2, e)$; see Section 2.5.2 for H input encodings. Put $C = (C_0, C_1)$.
4. Compute $K = \mathsf{H}(1, e, C)$; see Section 2.5.2 for H input encodings.
5. Output ciphertext C and session key K.

(Image taken from https://classic.mceliece.org/nist/mceliece-20201010.pdf)

Classic McEliece (CM)

$KEM = (KGen, Encap, Decap)$ $DEM = (Enc^{sym}, Dec^{sym})$ $PKE = (KGen, Enc, Dec)$



$(c_{KEM}, k) \leftarrow Encap(pk_{Bob})$ $c_{DEM} \leftarrow Enc^{sym}(k, m)$ $(c_{KEM}, c_{DEM}) \leftarrow Enc(pk_{Bob}, m)$

2.4.5 Encapsulation

The following randomized algorithm ENCAP takes as input a public key T. It outputs a ciphertext C and a session key K. Here is the algorithm:

1. Use FIXEDWEIGHT to generate a vector $e \in \mathbb{F}_2^n$ of weight t.
2. Compute $C_0 = \text{ENCODE}(e, T)$.
3. Compute $C_1 = \mathsf{H}(2, e)$; see Section 2.5.2 for H input encodings. Put $C = (C_0, C_1)$.
4. Compute $K = \mathsf{H}(1, e, C)$; see Section 2.5.2 for H input encodings.
5. Output ciphertext C and session key K.

(Image taken from https://classic.mceliece.org/nist/mceliece-20201010.pdf)

Classic McEliece (CM)

$PKE = (KGen, Enc, Dec)$

$(c_{KEM}, c_{DEM}) \leftarrow Enc(pk_{Bob}, m)$

2.4.5 Encapsulation

The following randomized algorithm ENCAP takes as input a public key T. It outputs a ciphertext C and a session key K. Here is the algorithm:

1. Use FIXEDWEIGHT to generate a vector $e \in \mathbb{F}_2^n$ of weight t.
2. Compute $C_0 = \text{ENCODE}(e, T)$.
3. Compute $C_1 = \mathsf{H}(2, e)$; see Section 2.5.2 for H input encodings. Put $C = (C_0, C_1)$.
4. Compute $K = \mathsf{H}(1, e, C)$; see Section 2.5.2 for H input encodings.
5. Output ciphertext C and session key K.

(Image taken from https://classic.mceliece.org/nist/mceliece-20201010.pdf)

Classic McEliece (CM)

For *any* message m:

$PKE = (KGen, Enc, Dec)$

$(c_{KEM}, c_{DEM}) \leftarrow Enc(pk_{Bob}, m)$

2.4.5 Encapsulation

The following randomized algorithm ENCAP takes as input a public key T. It outputs a ciphertext C and a session key K. Here is the algorithm:

1. Use FIXEDWEIGHT to generate a vector $e \in \mathbb{F}_2^n$ of weight t.
2. Compute $C_0 = \text{ENCODE}(e, T)$.
3. Compute $C_1 = \mathsf{H}(2, e)$; see Section 2.5.2 for H input encodings. Put $C = (C_0, C_1)$.
4. Compute $K = \mathsf{H}(1, e, C)$; see Section 2.5.2 for H input encodings.
5. Output ciphertext C and session key K.

(Image taken from *https://classic.mceliece.org/nist/mceliece-20201010.pdf*)

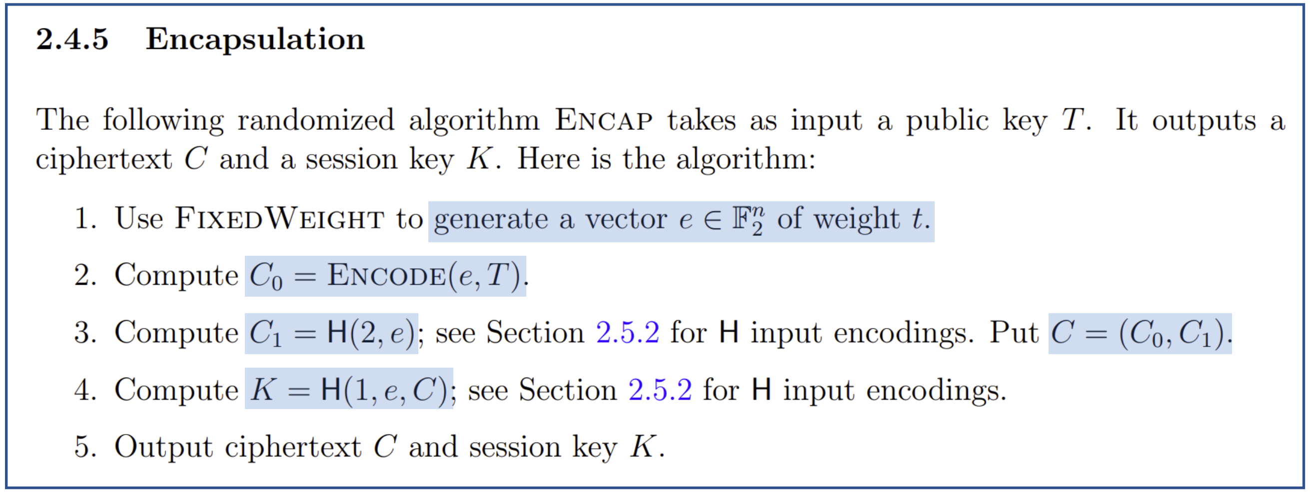
Classic McEliece (CM)

For *any* message m:

- Fix vector $e = \binom{e_{n-k}}{0^k}$.

$PKE = (KGen, Enc, Dec)$

$(c_{KEM}, c_{DEM}) \leftarrow Enc(pk_{Bob}, m)$



2.4.5 Encapsulation

The following randomized algorithm ENCAP takes as input a public key T. It outputs a ciphertext C and a session key K. Here is the algorithm:

1. Use FIXEDWEIGHT to generate a vector $e \in \mathbb{F}_2^n$ of weight t.
2. Compute $C_0 = \text{ENCODE}(e, T)$.
3. Compute $C_1 = \mathsf{H}(2, e)$; see Section 2.5.2 for H input encodings. Put $C = (C_0, C_1)$.
4. Compute $K = \mathsf{H}(1, e, C)$; see Section 2.5.2 for H input encodings.
5. Output ciphertext C and session key K.

(Image taken from https://classic.mceliece.org/nist/mceliece-20201010.pdf)

Classic McEliece (CM)

For *any* message m:

- Fix vector $e = \binom{e_{n-k}}{0^k}$.
- Set $C_0 = e_{n-k}, C_1 = \mathsf{H}(2, e)$ and $c_{KEM} \leftarrow (C_0, C_1)$.
- Compute $k = \mathsf{H}(1, e, c_{KEM})$ and $c_{DEM} \leftarrow Enc^{sym}(k, m)$.
- Return $c \leftarrow (c_{KEM}, c_{DEM})$.

$PKE = (KGen, Enc, Dec)$

$(c_{KEM}, c_{DEM}) \leftarrow Enc(pk_{Bob}, m)$

2.4.5 Encapsulation

The following randomized algorithm ENCAP takes as input a public key T. It outputs a ciphertext C and a session key K. Here is the algorithm:

1. Use FIXEDWEIGHT to generate a vector $e \in \mathbb{F}_2^n$ of weight t.
2. Compute $C_0 = \text{ENCODE}(e, T)$.
3. Compute $C_1 = \mathsf{H}(2, e)$; see Section 2.5.2 for H input encodings. Put $C = (C_0, C_1)$.
4. Compute $K = \mathsf{H}(1, e, C)$; see Section 2.5.2 for H input encodings.
5. Output ciphertext C and session key K.

(Image taken from https://classic.mceliece.org/nist/mceliece-20201010.pdf)

Classic McEliece (CM)

For *any* message m:

- Fix vector $e = \binom{e_{n-k}}{0^k}$.
- Set $C_0 = e_{n-k}, C_1 = \mathsf{H}(2, e)$ and $c_{KEM} \leftarrow (C_0, C_1)$.
- Compute $k = \mathsf{H}(1, e, c_{KEM})$ and $c_{DEM} \leftarrow Enc^{sym}(k, m)$.
- Return $c \leftarrow (c_{KEM}, c_{DEM})$.

For *any* CM private key sk_* ,

$PKE = (KGen, Enc, Dec)$

PKE (Hybrid)

$(c_{KEM}, c_{DEM}) \leftarrow Enc(pk_{Bob}, m)$

2.4.5 Encapsulation

The following randomized algorithm ENCAP takes as input a public key T. It outputs a ciphertext C and a session key K. Here is the algorithm:

1. Use FIXEDWEIGHT to generate a vector $e \in \mathbb{F}_2^n$ of weight t.
2. Compute $C_0 = \text{ENCODE}(e, T)$.
3. Compute $C_1 = \mathsf{H}(2, e)$; see Section 2.5.2 for H input encodings. Put $C = (C_0, C_1)$.
4. Compute $K = \mathsf{H}(1, e, C)$; see Section 2.5.2 for H input encodings.
5. Output ciphertext C and session key K.

(Image taken from https://classic.mceliece.org/nist/mceliece-20201010.pdf)

Classic McEliece (CM)

For *any* message m:

- Fix vector $e = \binom{e_{n-k}}{0^k}$.
- Set $C_0 = e_{n-k}, C_1 = \mathsf{H}(2, e)$ and $c_{KEM} \leftarrow (C_0, C_1)$.
- Compute $k = \mathsf{H}(1, e, c_{KEM})$ and $c_{DEM} \leftarrow Enc^{sym}(k, m)$.
- Return $c \leftarrow (c_{KEM}, c_{DEM})$.

For *any* CM private key sk_* ,

$$Dec(sk_*, c) = m \ (\neq \perp).$$

$$PKE = (KGen, Enc, Dec)$$

PKE (Hybrid)

$$(c_{KEM}, c_{DEM}) \leftarrow Enc(pk_{Bob}, m)$$

2.4.5 Encapsulation

The following randomized algorithm ENCAP takes as input a public key T. It outputs a ciphertext C and a session key K. Here is the algorithm:

1. Use FIXEDWEIGHT to generate a vector $e \in \mathbb{F}_2^n$ of weight t.
2. Compute $C_0 = \text{ENCODE}(e, T)$.
3. Compute $C_1 = \mathsf{H}(2, e)$; see Section 2.5.2 for H input encodings. Put $C = (C_0, C_1)$.
4. Compute $K = \mathsf{H}(1, e, C)$; see Section 2.5.2 for H input encodings.
5. Output ciphertext C and session key K.

(Image taken from https://classic.mceliece.org/nist/mceliece-20201010.pdf)

Classic McEliece (CM)

- Anonymity of CM can possibly be proven by other “more direct” approaches.

Classic McEliece (CM)

- Anonymity of CM can possibly be proven by other “more direct” approaches.

We’re **NOT** indicating any problems with IND-CCA security of CM.

Classic McEliece (CM)

- Anonymity of CM can possibly be proven by other “more direct” approaches.

We’re **NOT** indicating any problems with IND-CCA security of CM.

Since CM closely follows the $\mathsf{FO}^{\not\perp}$ transform to construct its KEM, the analysis of [Jiang-Zhang-Chen-Wang-Ma’18] can be extended to CM to obtain (relatively) tight security bounds in the QROM.

SaberCore vs Saber

SaberCore: the “core” scheme

Saber: the “actual” implemented scheme

SaberCore vs Saber

SaberCore: the "core" scheme

2.5.2 Saber.KEM Key Encapsulation

The Saber key encapsulation is specified by the following algorithm and makes use of Saber.PKE.Enc as specified in Algorithm 2.

Algorithm 5: $\texttt{Saber.KEM.Encaps}(pk := (seed_{\boldsymbol{A}}, \boldsymbol{b}))$

$m \leftarrow \mathcal{U}(\{0,1\}^{256})$
$(\hat{K}, r) = \mathcal{G}(\mathcal{F}(pk), m)$
$c = \texttt{Saber.PKE.Enc}(pk, m; r)$
$K = \mathcal{H}(\hat{K}, c)$
return (c, K)

Saber: the "actual" implemented scheme

8.5.2 Saber.KEM.Encaps

This function generates a session key and the ciphertext corresponding the key. The algorithm is described in Alg 21.

Algorithm 21: Algorithm Saber.KEM.Encaps for generating session key and ciphertext.

Input: $PublicKey_{cca}$: public key generated by Saber.KEM.KeyGen
Output: $SessionKey_{cca}$: session key,
$CipherText_{cca}$: cipher text corresponding to the session key

randombytes(m, SABER_KEYBYTES)
SHA3-256(m, m, SABER_KEYBYTES)
SHA3-256($hash\_pk$, $PublicKey_{cca}$, SABER_INDCPA_PUBKEYBYTES)
$buf = (hash\_pk \parallel m)$
SHA3-512(kr, buf, 2×SABER_KEYBYTES)
Split kr in two equal chunks of length SABER_KEYBYTES and obtain $(r \parallel k) = kr$
$CipherText_{cca}$ = Saber.PKE.Enc($m, r, PublicKey_{cca}$)
SHA3-256(r', $CipherText_{cca}$, SABER_BYTES_CCA_DEC)
$kr' = (r' \parallel k)$
SHA3-256($SessionKey_{cca}$, kr', 2×SABER_KEYBYTES)
return $(SessionKey_{cca}, CipherText_{cca})$

"$k \leftarrow H(\hat{k}, H(c))$"

(Image taken from https://www.esat.kuleuven.be/cosic/pqcrypto/saber/files/saberspecround3.pdf)

SaberCore vs Saber

SaberCore: the "core" scheme

2.5.2 Saber.KEM Key Encapsulation

The Saber key encapsulation is specified by the following algorithm and makes use of Saber.PKE.Enc as specified in Algorithm 2.

Algorithm 5: $\texttt{Saber.KEM.Encaps}(pk := (seed_{\boldsymbol{A}}, \boldsymbol{b}))$

$m \leftarrow \mathcal{U}(\{0,1\}^{256})$
$(\hat{K}, r) = \mathcal{G}(\mathcal{F}(pk), m)$
$c = \texttt{Saber.PKE.Enc}(pk, m; r)$
$K = \mathcal{H}(\hat{K}, c)$
return (c, K)

(Thanks to Peter Schwabe.)

Saber: the "actual" implemented scheme

8.5.2 Saber.KEM.Encaps

This function generates a session key and the ciphertext corresponding the key. The algorithm is described in Alg 21.

Algorithm 21: Algorithm Saber.KEM.Encaps for generating session key and ciphertext.

Input: $PublicKey_{cca}$: public key generated by Saber.KEM.KeyGen
Output: $SessionKey_{cca}$: session key,
$CipherText_{cca}$: cipher text corresponding to the session key

randombytes(m, SABER_KEYBYTES)
SHA3-256(m, m, SABER_KEYBYTES)
SHA3-256($hash\_pk$, $PublicKey_{cca}$, SABER_INDCPA_PUBKEYBYTES)
$buf = (hash\_pk \parallel m)$
SHA3-512(kr, buf, 2×SABER_KEYBYTES)
Split kr in two equal chunks of length SABER_KEYBYTES and obtain $(r \parallel k) = kr$
$CipherText_{cca}$ = Saber.PKE.Enc($m, r, PublicKey_{cca}$)
SHA3-256(r', $CipherText_{cca}$, SABER_BYTES_CCA_DEC)
$kr' = (r' \parallel k)$
SHA3-256($SessionKey_{cca}$, kr', 2×SABER_KEYBYTES)
return ($SessionKey_{cca}$, $CipherText_{cca}$)

"$k \leftarrow H(\hat{k}, H(c))$"

(Image taken from https://www.esat.kuleuven.be/cosic/pqcrypto/saber/files/saberspecround3.pdf)

SaberCore

Saber: Module-LWR based key exchange, CPA-secure encryption and CCA-secure KEM

Jan-Pieter D'Anvers, Angshuman Karmakar
Sujoy Sinha Roy, and Frederik Vercauteren

imec-COSIC, KU Leuven
Kasteelpark Arenberg 10, Bus 2452, B-3001 Leuven-Heverlee, Belgium
firstname.lastname@esat.kuleuven.be

Theorem 6 (QROM, Jiang et al. [32]). *For any IND-CCA quantum adversary B, making at most $q_{\mathcal{H}}$ and $q_{\mathcal{G}}$ queries to respectively the random quantum oracle $\mathcal{G}$ and $\mathcal{H}$, and q_D many (classical) queries to the decryption oracle, there exists an adversary A such that:*

$$Adv^{ind\text{-}cca}_{Saber.KEM}(B) \leqslant 2q_{\mathcal{H}}\frac{1}{\sqrt{2^{256}}} + 4q_{\mathcal{G}}\sqrt{\delta} + 2(q_{\mathcal{G}} + q_{\mathcal{H}})\sqrt{Adv^{ind\text{-}cpa}_{Saber.PKE}(A)}$$

[D'Anvers-Karmakar-Roy-Vercauteren'18]

(Image taken from https://eprint.iacr.org/2018/230.pdf)

SaberCore

Theorem 6.5. *In the quantum random oracle model, where $\mathcal{G}$ and $\mathcal{H}$ are assumed to be random oracles, for any IND-CCA quantum adversary B, making at most $q_\mathcal{H}$ and $q_\mathcal{G}$ queries to respectively the random quantum oracle $\mathcal{G}$ and $\mathcal{H}$, and q_D many (classical) queries to the decryption oracle, there exists an adversary A, with approximately the same running time as B, such that:*

$$Adv^{ind\text{-}cca}_{Saber.KEM}(B) \leqslant 2q_\mathcal{H}\frac{1}{\sqrt{2^{256}}} + 4q_\mathcal{G}\sqrt{\delta} + 2(q_\mathcal{G} + q_\mathcal{H})\sqrt{Adv^{ind\text{-}cpa}_{Saber.PKE}(A) + 1/|M|}$$

[Saber's specification document]

Theorem 6 (QROM, Jiang et al. [32]). *For any IND-CCA quantum adversary B, making at most $q_\mathcal{H}$ and $q_\mathcal{G}$ queries to respectively the random quantum oracle $\mathcal{G}$ and $\mathcal{H}$, and q_D many (classical) queries to the decryption oracle, there exists an adversary A such that:*

$$Adv^{ind\text{-}cca}_{Saber.KEM}(B) \leqslant 2q_\mathcal{H}\frac{1}{\sqrt{2^{256}}} + 4q_\mathcal{G}\sqrt{\delta} + 2(q_\mathcal{G} + q_\mathcal{H})\sqrt{Adv^{ind\text{-}cpa}_{Saber.PKE}(A)}$$

[D'Anvers-Karmakar-Roy-Vercauteren'18]

(Image taken from https://eprint.iacr.org/2018/230.pdf)

(Image taken from https://www.esat.kuleuven.be/cosic/pqcrypto/saber/files/saberspecround3.pdf)

SaberCore

(Informal) **Theorem** [Grubbs-Maram-Paterson’21]:

Hybrid PKE schemes obtained from $\mathsf{FO}^{\not\perp}$ *KEMs via the generic KEM-DEM composition are also ANO-CCA secure in the QROM.**

*(*Provided the base PKE scheme satisfies some additional mild security properties.)*

SaberCore

(Informal) **Theorem** [Grubbs-Maram-Paterson’21]:

Hybrid PKE schemes obtained from $\mathsf{FO}^{\not\perp}$ *KEMs via the generic KEM-DEM composition are also ANO-CCA secure in the QROM.**

*(*Provided the base PKE scheme satisfies some additional mild security properties.)*

6.1.2 Security in the Quantum Random Oracle Model

Jiang et al. [24] provide a security reduction against a quantum adversary in the quantum random oracle model from IND-CCA security to OW-CPA security. IND-CPA with a

19

(Image taken from https://www.esat.kuleuven.be/cosic/pqcrypto/saber/files/saberspecround3.pdf)

SaberCore

(Informal) **Theorem** [Grubbs-Maram-Paterson’21]:

Hybrid PKE schemes obtained from $\mathsf{FO}^{\not\perp}$ *KEMs via the generic KEM-DEM composition are also ANO-CCA secure in the QROM.**

*(*Provided the base PKE scheme satisfies some additional mild security properties.)*

6.1.2 Security in the Quantum Random Oracle Model

Jiang et al. [24] provide a security reduction against a quantum adversary in the quantum random oracle model from IND-CCA security to OW-CPA security. IND-CPA with a

19

(Image taken from https://www.esat.kuleuven.be/cosic/pqcrypto/saber/files/saberspecround3.pdf)

SaberCore

6.1.2 Security in the Quantum Random Oracle Model

Jiang et al. [24] provide a security reduction against a quantum adversary in the quantum random oracle model from IND-CCA security to OW-CPA security. IND-CPA with a

19

KGen'	$\mathsf{Encap}(\mathsf{pk})$	$\mathsf{Decap}(\mathsf{sk}', c)$
$(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KGen}$	$m \leftarrow_{\$} \mathcal{M}$	Parse $\mathsf{sk}' = (\mathsf{sk}, s)$
$s \leftarrow_{\$} \mathcal{M}$	$r \leftarrow G(m)$	$m' \leftarrow \mathsf{Dec}(\mathsf{sk}, c)$
$\mathsf{sk}' = (\mathsf{sk}, s)$	$c \leftarrow \mathsf{Enc}(\mathsf{pk}, m; r)$	$r' \leftarrow G(m')$
return $(\mathsf{pk}, \mathsf{sk}')$	$k \leftarrow H(m, c)$	$c' \leftarrow \mathsf{Enc}(\mathsf{pk}, m'; r')$
	return (c, k)	**if** $c' = c$ **then**
		return $H(m', c)$
		else return $H(s, c)$

$\mathsf{FO}^{\not\perp}$

(Image taken from *https://eprint.iacr.org/2021/708.pdf* [Grubbs-Maram-Paterson'21])

(Image taken from *https://www.esat.kuleuven.be/cosic/pqcrypto/saber/files/saberspecround3.pdf*)

SaberCore

6.1.2 Security in the Quantum Random Oracle Model

Jiang et al. [24] provide a security reduction against a quantum adversary in the quantum random oracle model from IND-CCA security to OW-CPA security. IND-CPA with a

19

KGen′	Encap(pk)	Decap(sk′, c)
1 : $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KGen}$	1 : $m \leftarrow_{\$} \mathcal{M}$	1 : Parse $\mathsf{sk}' = (\mathsf{sk}, s)$
2 : $s \leftarrow_{\$} \mathcal{M}$	2 : $r \leftarrow G(m)$	2 : $m' \leftarrow \mathsf{Dec}(\mathsf{sk}, c)$
3 : $\mathsf{sk}' = (\mathsf{sk}, s)$	3 : $c \leftarrow \mathsf{Enc}(\mathsf{pk}, m; r)$	3 : $r' \leftarrow G(m')$
4 : **return** $(\mathsf{pk}, \mathsf{sk}')$	4 : $k \leftarrow H(m, c)$	4 : $c' \leftarrow \mathsf{Enc}(\mathsf{pk}, m'; r')$
	5 : **return** (c, k)	5 : **if** $c' = c$ **then**
		6 : **return** $H(m', c)$
		7 : **else return** $H(s, c)$

$\mathsf{FO}^{\not\perp}$

KGen′	Encap(pk)	Decap(sk′, c)
1 : $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KGen}$	1 : $m \leftarrow_{\$} \mathcal{M}$	1 : Parse $\mathsf{sk}' = (\mathsf{sk}, \mathsf{pk}, F(\mathsf{pk}), s)$
2 : $s \leftarrow_{\$} \mathcal{M}$	2 : $(\hat{k}, r) \leftarrow G(F(pk), m)$	2 : $m' \leftarrow \mathsf{Dec}(\mathsf{sk}, c)$
3 : $\mathsf{sk}' \leftarrow (\mathsf{sk}, \mathsf{pk}, F(\mathsf{pk}), s)$	3 : $c \leftarrow \mathsf{Enc}(\mathsf{pk}, m; r)$	3 : $(\hat{k}', r') \leftarrow G(F(\mathsf{pk}), m')$
4 : **return** $(\mathsf{pk}, \mathsf{sk}')$	4 : $k \leftarrow H(\hat{k}, c)$	4 : $c' \leftarrow \mathsf{Enc}(\mathsf{pk}, m'; r')$
	5 : **return** (c, k)	5 : **if** $c' = c$ **then**
		6 : **return** $H(\hat{k}', c)$
		7 : **else return** $H(s, c)$

SaberCore

(Image taken from https://eprint.iacr.org/2021/708.pdf [Grubbs-Maram-Paterson'21])

(Image taken from https://www.esat.kuleuven.be/cosic/pqcrypto/saber/files/saberspecround3.pdf)

SaberCore

6.1.2 Security in the Quantum Random Oracle Model

Jiang et al. [24] provide a security reduction against a quantum adversary in the quantum random oracle model from IND-CCA security to OW-CPA security. IND-CPA with a

19

(Image taken from https://www.esat.kuleuven.be/cosic/pqcrypto/saber/files/saberspecround3.pdf)

SaberCore

6.1.2 Security in the Quantum Random Oracle Model

Jiang et al. [24] provide a security reduction against a quantum adversary in the quantum random oracle model from IND-CCA security to OW-CPA security. IND-CPA with a

19

- In such a reduction, we need to simulate the decapsulation oracle in the QROM without possessing the secret key.

(Image taken from *https://www.esat.kuleuven.be/cosic/pqcrypto/saber/files/saberspecround3.pdf*)

SaberCore

6.1.2 Security in the Quantum Random Oracle Model

Jiang et al. [24] provide a security reduction against a quantum adversary in the quantum random oracle model from IND-CCA security to OW-CPA security. IND-CPA with a

19

- In such a reduction, we need to simulate the decapsulation oracle in the QROM without possessing the secret key.
- Trick used in [Jiang-Zhang-Chen-Wang-Ma'18] w.r.t. $FO^{\not\perp}$:

(Image taken from *https://www.esat.kuleuven.be/cosic/pqcrypto/saber/files/saberspecround3.pdf*)

SaberCore

6.1.2 Security in the Quantum Random Oracle Model

Jiang et al. [24] provide a security reduction against a quantum adversary in the quantum random oracle model from IND-CCA security to OW-CPA security. IND-CPA with a

19

- In such a reduction, we need to simulate the decapsulation oracle in the QROM without possessing the secret key.
- Trick used in [Jiang-Zhang-Chen-Wang-Ma'18] w.r.t. $\mathsf{FO}^{\not\perp}$:
  - Replace key-derivation "$k \leftarrow H(m, c)$" with "$k \leftarrow H'(c)$", where
  - $c = g(m) = Enc\big(pk, m; G(m)\big)$ is the deterministic encryption of m and
  - H' is a secret random oracle.

(Image taken from https://www.esat.kuleuven.be/cosic/pqcrypto/saber/files/saberspecround3.pdf)

SaberCore

> **6.1.2 Security in the Quantum Random Oracle Model**
>
> Jiang et al. [24] provide a security reduction against a quantum adversary in the quantum random oracle model from IND-CCA security to OW-CPA security. IND-CPA with a
>
> 19

- In such a reduction, we need to simulate the decapsulation oracle in the QROM without possessing the secret key.
- Trick used in [Jiang-Zhang-Chen-Wang-Ma'18] w.r.t. $\mathrm{FO}^{\not\perp}$:
  - Replace key-derivation "$k \leftarrow H(m, c)$" with "$k \leftarrow H'(c)$", where
  - $c = g(m) = Enc(pk, m; G(m))$ is the deterministic encryption of m and
  - H' is a secret random oracle.
  - Replacement is justified by the *injectivity* of $g(\cdot)$, relying on the correctness of underlying encryption.

(Image taken from https://www.esat.kuleuven.be/cosic/pqcrypto/saber/files/saberspecround3.pdf)

SaberCore

6.1.2 Security in the Quantum Random Oracle Model

Jiang et al. [24] provide a security reduction against a quantum adversary in the quantum random oracle model from IND-CCA security to OW-CPA security. IND-CPA with a

19

- In such a reduction, we need to simulate the decapsulation oracle in the QROM without possessing the secret key.
- Trick used in [Jiang-Zhang-Chen-Wang-Ma'18] w.r.t. $\mathrm{FO}^{\not\perp}$:
  - Replace key-derivation "$k \leftarrow H(m, c)$" with "$k \leftarrow H'(c)$", where
  - $c = g(m) = Enc\big(pk, m; G(m)\big)$ is the deterministic encryption of m and
  - H' is a secret random oracle.
  - Replacement is justified by the *injectivity* of $g(\cdot)$, relying on the correctness of underlying encryption.
  - Effectively, $Decap(sk, c)$ can be simulated by returning $H'(c)$.

(Image taken from https://www.esat.kuleuven.be/cosic/pqcrypto/saber/files/saberspecround3.pdf)

SaberCore

> **6.1.2 Security in the Quantum Random Oracle Model**
>
> Jiang et al. [24] provide a security reduction against a quantum adversary in the quantum random oracle model from IND-CCA security to OW-CPA security. IND-CPA with a
>
> 19

- In such a reduction, we need to simulate the decapsulation oracle in the QROM without possessing the secret key.
- However, in SaberCore:

(Image taken from *https://www.esat.kuleuven.be/cosic/pqcrypto/saber/files/saberspecround3.pdf*)

SaberCore

6.1.2 Security in the Quantum Random Oracle Model

Jiang et al. [24] provide a security reduction against a quantum adversary in the quantum random oracle model from IND-CCA security to OW-CPA security. IND-CPA with a

19

- In such a reduction, we need to simulate the decapsulation oracle in the QROM without possessing the secret key.
- However, in SaberCore:
  - Encapsulated keys derived as "$k \leftarrow H(\hat{k}, c)$" (cf. "$k \leftarrow H(m, c)$" of $\mathrm{FO}^{\not\perp}$)
  - with "pre-key" $\hat{k}$ derived as a hash of m – i.e., "$(\hat{k}, r) \leftarrow G(F(pk), m)$".
  - Essentially, there is a "nested" hashing of m in the key-derivation.

(Image taken from https://www.esat.kuleuven.be/cosic/pqcrypto/saber/files/saberspecround3.pdf)

SaberCore

> **6.1.2 Security in the Quantum Random Oracle Model**
>
> Jiang et al. [24] provide a security reduction against a quantum adversary in the quantum random oracle model from IND-CCA security to OW-CPA security. IND-CPA with a
>
> 19

- In such a reduction, we need to simulate the decapsulation oracle in the QROM without possessing the secret key.
- However, in SaberCore:
  - Encapsulated keys derived as "$k \leftarrow H(\hat{k}, c)$" (cf. "$k \leftarrow H(m, c)$" of $FO^{\not\perp}$)
  - with "pre-key" $\hat{k}$ derived as a hash of m – i.e., "$(\hat{k}, r) \leftarrow G(F(pk), m)$".
  - Essentially, there is a "nested" hashing of m in the key-derivation.
  - Need some additional injectivity arguments to use the trick of [Jiang-Zhang-Chen-Wang-Ma'18].

(Image taken from https://www.esat.kuleuven.be/cosic/pqcrypto/saber/files/saberspecround3.pdf)

SaberCore

> **6.1.2 Security in the Quantum Random Oracle Model**
>
> Jiang et al. [24] provide a security reduction against a quantum adversary in the quantum random oracle model from IND-CCA security to OW-CPA security. IND-CPA with a
>
> 19

- In such a reduction, we need to simulate the decapsulation oracle in the QROM without possessing the secret key.
- However, in SaberCore:
  - Encapsulated keys derived as "$k \leftarrow H(\hat{k}, c)$" (cf. "$k \leftarrow H(m, c)$" of $\mathrm{FO}^{\not\perp}$)
  - with "pre-key" $\hat{k}$ derived as a hash of m – i.e., "$(\hat{k}, r) \leftarrow G(F(pk), m)$".
  - Essentially, there is a "nested" hashing of m in the key-derivation.
  - Need some additional injectivity arguments to use the trick of [Jiang-Zhang-Chen-Wang-Ma'18].

We were able to adapt the simulation trick to SaberCore, by observing that the nested hashing of m is *length-preserving*, i.e., $m \in \{0,1\}^{256}$ and $\hat{k} \in \{0,1\}^{256}$.

(Image taken from https://www.esat.kuleuven.be/cosic/pqcrypto/saber/files/saberspecround3.pdf)

SaberCore

- Our approach to “recover” IND-CCA security of SaberCore, with the same tightness as claimed in its third-round specification, also led to:

SaberCore

- Our approach to “recover” IND-CCA security of SaberCore, with the same tightness as claimed in its third-round specification, also led to:

(Informal) **Theorem** [Grubbs-Maram-Paterson’21]:

Hybrid PKE schemes obtained from SaberCore via the generic KEM-DEM composition in the QROM are:

- *ANO-CCA secure, and*

SaberCore

- Our approach to "recover" IND-CCA security of SaberCore, with the same tightness as claimed in its third-round specification, also led to:

(Informal) **Theorem** [Grubbs-Maram-Paterson'21]:

Hybrid PKE schemes obtained from SaberCore via the generic KEM-DEM composition in the QROM are:

- *ANO-CCA secure, and*
- *SROB-CCA secure, provided the DEM satisfies an appropriate notion of robustness.*

FrodoKEM

In addition, the following eight candidate algorithms will advance to the third round:

Alternate Candidates

Public-Key Encryption/KEMs
BIKE;
FrodoKEM
HQC
NTRU Prime
SIKE

FrodoKEM uses the same FO-type transform as SaberCore.

(Image taken from *https://www.nist.gov/news-events/news/2020/07/pqc-standardization-process-third-round-candidate-announcement*)

FrodoKEM

In addition, the following eight candidate algorithms will advance to the third round:

Alternate Candidates

Public-Key Encryption/KEMs

BIKE;

FrodoKEM

HQC

NTRU Prime

SIKE

FrodoKEM uses the same FO-type transform as SaberCore.

We expect our results on SaberCore to be applicable to the "actual" FrodoKEM.

(Image taken from *https://www.nist.gov/news-events/news/2020/07/pqc-standardization-process-third-round-candidate-announcement*)

CRYSTALS-KYBER, Saber

(Informal) **Theorem** [Grubbs-Maram-Paterson'21]:

*Hybrid PKE schemes obtained from $\mathsf{FO}^{\not\perp}$ KEMs via the generic KEM-DEM composition are also ANO-CCA secure in the QROM.**

*(*Provided the base PKE scheme satisfies some additional mild security properties.)*

CRYSTALS-KYBER and Saber use a transform that deviates *even further* from $\mathsf{FO}^{\not\perp}$ when compared to SaberCore/FrodoKEM.

CRYSTALS-KYBER, Saber

KGen'	$\mathsf{Encap}(\mathsf{pk})$	$\mathsf{Decap}(\mathsf{sk}', c)$
$(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KGen}$	$m \leftarrow_{\$} \mathcal{M}$	$\text{Parse } \mathsf{sk}' = (\mathsf{sk}, \mathsf{pk}, F(\mathsf{pk}), s)$
$s \leftarrow_{\$} \mathcal{M}$	$(\hat{k}, r) \leftarrow G(F(pk), m)$	$m' \leftarrow \mathsf{Dec}(\mathsf{sk}, c)$
$\mathsf{sk}' \leftarrow (\mathsf{sk}, \mathsf{pk}, F(\mathsf{pk}), s)$	$c \leftarrow \mathsf{Enc}(\mathsf{pk}, m; r)$	$(\hat{k}', r') \leftarrow G(F(\mathsf{pk}), m')$
$\mathbf{return}\ (\mathsf{pk}, \mathsf{sk}')$	$k \leftarrow H(\hat{k}, c)$	$c' \leftarrow \mathsf{Enc}(\mathsf{pk}, m'; r')$
	$\mathbf{return}\ (c, k)$	$\mathbf{if}\ c' = c\ \mathbf{then}$
		$\quad \mathbf{return}\ H(\hat{k}', c)$
		$\mathbf{else\ return}\ H(s, c)$

SaberCore

(Image taken from https://eprint.iacr.org/2021/708.pdf [Grubbs-Maram-Paterson'21])

CRYSTALS-KYBER, Saber

KGen'	$\mathsf{Encap}(\mathsf{pk})$	$\mathsf{Decap}(\mathsf{sk}', c)$
$(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KGen}$	$m \leftarrow_\$ \mathcal{M}$	Parse $\mathsf{sk}' = (\mathsf{sk}, \mathsf{pk}, F(\mathsf{pk}), s)$
$s \leftarrow_\$ \mathcal{M}$	$(\hat{k}, r) \leftarrow G(F(pk), m)$	$m' \leftarrow \mathsf{Dec}(\mathsf{sk}, c)$
$\mathsf{sk}' \leftarrow (\mathsf{sk}, \mathsf{pk}, F(\mathsf{pk}), s)$	$c \leftarrow \mathsf{Enc}(\mathsf{pk}, m; r)$	$(\hat{k}', r') \leftarrow G(F(\mathsf{pk}), m')$
return $(\mathsf{pk}, \mathsf{sk}')$	$k \leftarrow H(\hat{k}, c)$	$c' \leftarrow \mathsf{Enc}(\mathsf{pk}, m'; r')$
	return (c, k)	**if** $c' = c$ **then**
		return $H(\hat{k}', c)$
		else return $H(s, c)$

SaberCore

KGen'	$\mathsf{Encap}(\mathsf{pk})$	$\mathsf{Decap}(\mathsf{sk}', c)$
$(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KGen}$	$m \leftarrow_\$ \mathcal{M}$	Parse $\mathsf{sk}' = (\mathsf{sk}, \mathsf{pk}, F(\mathsf{pk}), s)$
$s \leftarrow_\$ \mathcal{M}$	$m \leftarrow F(m)$	$m' \leftarrow \mathsf{Dec}(\mathsf{sk}, c)$
$\mathsf{sk}' \leftarrow (\mathsf{sk}, \mathsf{pk}, F(\mathsf{pk}), s)$	$(\hat{k}, r) \leftarrow G(F(pk), m)$	$(\hat{k}', r') \leftarrow G(F(\mathsf{pk}), m')$
return $(\mathsf{pk}, \mathsf{sk}')$	$c \leftarrow \mathsf{Enc}(\mathsf{pk}, m; r)$	$c' \leftarrow \mathsf{Enc}(\mathsf{pk}, m'; r')$
	$k \leftarrow \mathsf{KDF}(\hat{k}, F(c))$	**if** $c' = c$ **then**
	return (c, k)	**return** $\mathsf{KDF}(\hat{k}', F(c))$
		else return $\mathsf{KDF}(s, F(c))$

CRYSTALS-KYBER, Saber

(Image taken from https://eprint.iacr.org/2021/708.pdf [Grubbs-Maram-Paterson'21])

CRYSTALS-KYBER, Saber

- The “nested” (compressing) hash of ciphertext, namely "$F(c)$", in the key-derivation "$k \leftarrow KDF(\hat{k}, F(c))$" acts as a barrier w.r.t. establishing the IND-CCA security of CRYSTALS-KYBER and Saber in the QROM with the claimed tightness.

CRYSTALS-KYBER, Saber

- The “nested” (compressing) hash of ciphertext, namely "$F(c)$", in the key-derivation "$k \leftarrow KDF(\hat{k}, F(c))$" acts as a barrier w.r.t. establishing the IND-CCA security of CRYSTALS-KYBER and Saber in the QROM with the claimed tightness.
- Also acts as a barrier in our attempts to extend the anonymity and robustness analysis of SaberCore to the schemes.

CRYSTALS-KYBER, Saber

- The "nested" (compressing) hash of ciphertext, namely "$F(c)$", in the key-derivation "$k \leftarrow KDF(\hat{k}, F(c))$" acts as a barrier w.r.t. establishing the IND-CCA security of CRYSTALS-KYBER and Saber in the QROM with the claimed tightness.
- Also acts as a barrier in our attempts to extend the anonymity and robustness analysis of SaberCore to the schemes.

We suggest CRYSTALS-KYBER and Saber use the same FO-type transform of SaberCore/FrodoKEM:

CRYSTALS-KYBER, Saber

- The "nested" (compressing) hash of ciphertext, namely "$F(c)$", in the key-derivation "$k \leftarrow KDF(\hat{k}, F(c))$" acts as a barrier w.r.t. establishing the IND-CCA security of CRYSTALS-KYBER and Saber in the QROM with the claimed tightness.
- Also acts as a barrier in our attempts to extend the anonymity and robustness analysis of SaberCore to the schemes.

We suggest CRYSTALS-KYBER and Saber use the same FO-type transform of SaberCore/FrodoKEM:

- *Because we recover a corresponding tight IND-CCA security proof in the QROM,*

CRYSTALS-KYBER, Saber

- The “nested” (compressing) hash of ciphertext, namely "$F(c)$", in the key-derivation "$k \leftarrow KDF(\hat{k}, F(c))$" acts as a barrier w.r.t. establishing the IND-CCA security of CRYSTALS-KYBER and Saber in the QROM with the claimed tightness.
- Also acts as a barrier in our attempts to extend the anonymity and robustness analysis of SaberCore to the schemes.

We suggest CRYSTALS-KYBER and Saber use the same FO-type transform of SaberCore/FrodoKEM:

- *Because we recover a corresponding tight IND-CCA security proof in the QROM,*
- *and obtain similarly tight security proofs of anonymity and robustness.*

Conclusions

Conclusions

- Hybrid PKE schemes derived from Classic McEliece, via the generic KEM-DEM composition, are not strongly robust.

Conclusions

- Hybrid PKE schemes derived from Classic McEliece, via the generic KEM-DEM composition, are not strongly robust.
- SaberCore results in anonymous (and robust, for appropriate DEMs) hybrid PKE schemes in the QROM. Along the way, we repaired technical gaps in the IND-CCA security claims of SaberCore in the QROM. Our results should similarly apply to the FrodoKEM.

Conclusions

- Hybrid PKE schemes derived from Classic McEliece, via the generic KEM-DEM composition, are not strongly robust.
- SaberCore results in anonymous (and robust, for appropriate DEMs) hybrid PKE schemes in the QROM. Along the way, we repaired technical gaps in the IND-CCA security claims of SaberCore in the QROM. Our results should similarly apply to the FrodoKEM.
- However, the “actual” Saber differs from SaberCore and uses the same FO-type transform as that of CRYSTALS-KYBER. And this transform has similar issues with its *concrete* IND-CCA security in the QROM.

Conclusions

- Hybrid PKE schemes derived from Classic McEliece, via the generic KEM-DEM composition, are not strongly robust.
- SaberCore results in anonymous (and robust, for appropriate DEMs) hybrid PKE schemes in the QROM. Along the way, we repaired technical gaps in the IND-CCA security claims of SaberCore in the QROM. Our results should similarly apply to the FrodoKEM.
- However, the "actual" Saber differs from SaberCore and uses the same FO-type transform as that of CRYSTALS-KYBER. And this transform has similar issues with its *concrete* IND-CCA security in the QROM.
- We suggest CRYSTALS-KYBER and Saber, in essence, use the same transform as SaberCore/FrodoKEM.

Conclusions

- Hybrid PKE schemes derived from Classic McEliece, via the generic KEM-DEM composition, are not strongly robust.
- SaberCore results in anonymous (and robust, for appropriate DEMs) hybrid PKE schemes in the QROM. Along the way, we repaired technical gaps in the IND-CCA security claims of SaberCore in the QROM. Our results should similarly apply to the FrodoKEM.
- However, the “actual” Saber differs from SaberCore and uses the same FO-type transform as that of CRYSTALS-KYBER. And this transform has similar issues with its *concrete* IND-CCA security in the QROM.
- We suggest CRYSTALS-KYBER and Saber, in essence, use the same transform as SaberCore/FrodoKEM.
- Determining anonymity and robustness of NTRU is an open question.

Conclusions

- Hybrid PKE schemes derived from Classic McEliece, via the generic KEM-DEM composition, are not strongly robust.
- SaberCore results in anonymous (and robust, for appropriate DEMs) hybrid PKE schemes in the QROM. Along the way, we repaired technical gaps in the IND-CCA security claims of SaberCore in the QROM. Our results should similarly apply to the FrodoKEM.
- However, the "actual" Saber differs from SaberCore and uses the same FO-type transform as that of CRYSTALS-KYBER. And this transform has similar issues with its *concrete* IND-CCA security in the QROM.
- We suggest CRYSTALS-KYBER and Saber, in essence, use the same transform as SaberCore/FrodoKEM.
- Determining anonymity and robustness of NTRU is an open question.

Need to re-evaluate the IND-CCA security claims of the finalists in the QROM.