Lattice-Based Updatable KEM for Group Messaging

Joël Alwen (AWS Wickr)

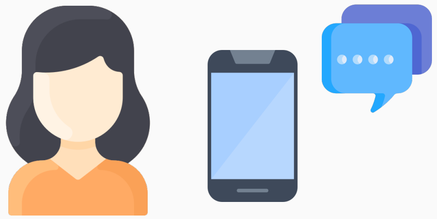
Georg Fuchsbauer (TU Wien)

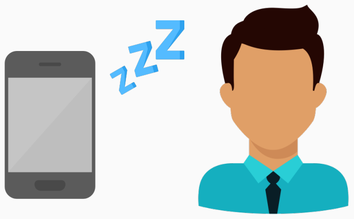
Marta Mularczyk (AWS Wickr)

Doreen Riepel (CISPA)

CRYPTO, August 2026

Secure (Group) Messaging

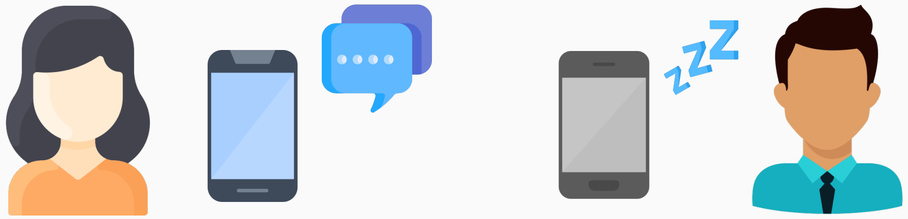




Security properties

- Authenticity
- Privacy
- Forward secrecy
- Post-compromise security

Secure (Group) Messaging



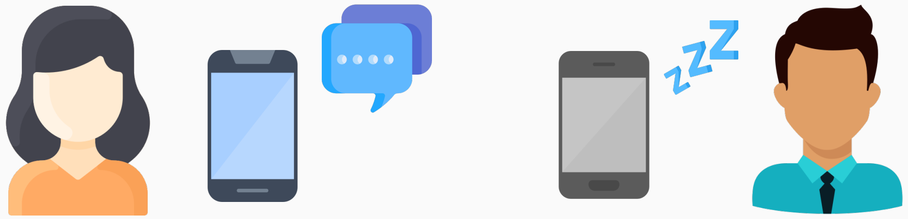
Security properties

- Authenticity
- Privacy
- Forward secrecy
- Post-compromise security

Message-Layer Security (MLS)

- Encrypts data to subsets of group members who know a PKE secret key
- ACDT19: replacing PKE with an updatable PKE (UPKE) improves forward secrecy

Secure (Group) Messaging



Security properties

- Authenticity
- Privacy
- Forward secrecy
- Post-compromise security

Message-Layer Security (MLS)

- Encrypts data to subsets of group members who know a PKE secret key
- ACDT19: replacing PKE with an updatable PKE (UPKE) improves forward secrecy

⇒ Follow-up works on UPKE (and UKEM), proposing stronger notions of security and post-quantum constructions

Updatable Key Encapsulation Mechanisms

Updatable Key Encapsulation Mechanism

(Short) syntax from [AFM24]

$\mathsf{UKEM} = (\mathsf{Gen}, \mathsf{Encaps}, \mathsf{Verify}_{\mathsf{mt}}, \mathsf{Decaps})$

Updatable Key Encapsulation Mechanism

(Short) syntax from [AFM24]

$\mathsf{UKEM} = (\mathsf{Gen}, \mathsf{Encaps}, \mathsf{Verify}_{\mathsf{mt}}, \mathsf{Decaps})$

$\mathsf{Gen} \xrightarrow{\$} (\mathsf{pk}_0, \mathsf{sk}_0)$ // initial public/secret key pair

Updatable Key Encapsulation Mechanism

(Short) syntax from [AFM24]

$\mathsf{UKEM} = (\mathsf{Gen}, \mathsf{Encaps}, \mathsf{Verify}_{\mathsf{mt}}, \mathsf{Decaps})$

$\mathsf{Gen} \xrightarrow{\$} (\mathsf{pk}_0, \mathsf{sk}_0)$ // initial public/secret key pair

$\mathsf{Encaps}(\mathsf{pk}_i) \xrightarrow{\$} (K, c, \mathsf{pk}_{i+1}, \mathsf{mt}_{i+1})$ // every encapsulation also updates the public key

Updatable Key Encapsulation Mechanism

(Short) syntax from [AFM24]

$\mathsf{UKEM} = (\mathsf{Gen}, \mathsf{Encaps}, \mathsf{Verify}_{\mathsf{mt}}, \mathsf{Decaps})$

$\mathsf{Gen} \xrightarrow{\$} (\mathsf{pk}_0, \mathsf{sk}_0)$ // initial public/secret key pair

$\mathsf{Encaps}(\mathsf{pk}_i) \xrightarrow{\$} (K, c, \mathsf{pk}_{i+1}, \mathsf{mt}_{i+1})$ // every encapsulation also updates the public key

// member-tags ensure that pk_{i+1} can be trusted (by parties not knowing sk_i)

Updatable Key Encapsulation Mechanism

(Short) syntax from [AFM24]

$\mathsf{UKEM} = (\mathsf{Gen}, \mathsf{Encaps}, \mathsf{Verify}_{\mathsf{mt}}, \mathsf{Decaps})$

$\mathsf{Gen} \xrightarrow{\$} (\mathsf{pk}_0, \mathsf{sk}_0)$ // initial public/secret key pair

$\mathsf{Encaps}(\mathsf{pk}_i) \xrightarrow{\$} (K, c, \mathsf{pk}_{i+1}, \mathsf{mt}_{i+1})$ // every encapsulation also updates the public key

$\mathsf{Verify}_{\mathsf{mt}}(\mathsf{pk}_i, \mathsf{pk}_{i+1}, \mathsf{mt}_{i+1}) \to 0/1$ // member-tags ensure that pk_{i+1} can be trusted (by parties not knowing sk_i)

Updatable Key Encapsulation Mechanism

(Short) syntax from [AFM24]

$\mathsf{UKEM} = (\mathsf{Gen}, \mathsf{Encaps}, \mathsf{Verify}_{\mathsf{mt}}, \mathsf{Decaps})$

$\mathsf{Gen} \xrightarrow{\$} (\mathsf{pk}_0, \mathsf{sk}_0)$ // initial public/secret key pair

$\mathsf{Encaps}(\mathsf{pk}_i) \xrightarrow{\$} (K, c, \mathsf{pk}_{i+1}, \mathsf{mt}_{i+1})$ // every encapsulation also updates the public key

$\mathsf{Verify}_{\mathsf{mt}}(\mathsf{pk}_i, \mathsf{pk}_{i+1}, \mathsf{mt}_{i+1}) \to 0/1$ // member-tags ensure that pk_{i+1} can be trusted (by parties not knowing sk_i)

$\mathsf{Decaps}(\mathsf{sk}_i, c, \mathsf{pk}_{i+1}) \to (K, \mathsf{sk}_{i+1})$

Updatable Key Encapsulation Mechanism

(Short) syntax from [AFM24]

$\mathsf{UKEM} = (\mathsf{Gen}, \mathsf{Encaps}, \mathsf{Verify}_{\mathsf{mt}}, \mathsf{Decaps})$

$\mathsf{Gen} \xrightarrow{\$} (\mathsf{pk}_0, \mathsf{sk}_0)$ // initial public/secret key pair

$\mathsf{Encaps}(\mathsf{pk}_i) \xrightarrow{\$} (K, c, \mathsf{pk}_{i+1}, \mathsf{mt}_{i+1})$ // every encapsulation also updates the public key

$\mathsf{Verify}_{\mathsf{mt}}(\mathsf{pk}_i, \mathsf{pk}_{i+1}, \mathsf{mt}_{i+1}) \to 0/1$ // member-tags ensure that pk_{i+1} can be trusted (by parties not knowing sk_i)

$\mathsf{Decaps}(\mathsf{sk}_i, c, \mathsf{pk}_{i+1}) \to (K, \mathsf{sk}_{i+1})$

Security

- IND: $K \approx K_{\$}$

Updatable Key Encapsulation Mechanism

(Short) syntax from [AFM24]

$\mathsf{UKEM} = (\mathsf{Gen}, \mathsf{Encaps}, \mathsf{Verify}_{\mathsf{mt}}, \mathsf{Decaps})$

$\mathsf{Gen} \xrightarrow{\$} (\mathsf{pk}_0, \mathsf{sk}_0)$ // initial public/secret key pair

$\mathsf{Encaps}(\mathsf{pk}_i) \xrightarrow{\$} (K, c, \mathsf{pk}_{i+1}, \mathsf{mt}_{i+1})$ // every encapsulation also updates the public key

$\mathsf{Verify}_{\mathsf{mt}}(\mathsf{pk}_i, \mathsf{pk}_{i+1}, \mathsf{mt}_{i+1}) \to 0/1$ // member-tags ensure that pk_{i+1} can be trusted (by parties not knowing sk_i)

$\mathsf{Decaps}(\mathsf{sk}_i, c, \mathsf{pk}_{i+1}) \to (K, \mathsf{sk}_{i+1})$

Security

- IND: $K \approx K_{\$}$
- CCA: via decapsulation oracle
  - How to incorporate updated public and secret keys?
  - Corruptions?

Existing Definitions

Chosen-Update Security [DKW21]

- The adversary can perform publicly verifiable chosen updates of the public key

Existing Definitions

Chosen-Update Security [DKW21]

- The adversary can perform publicly verifiable chosen updates of the public key



Existing Definitions

Chosen-Update Security [DKW21]

- The adversary can perform publicly verifiable chosen updates of the public key



Existing Definitions

Chosen-Update Security [DKW21]

- The adversary can perform publicly verifiable chosen updates of the public key



Existing Definitions

Chosen-Update Security [DKW21]

- The adversary can perform publicly verifiable chosen updates of the public key



Existing Definitions

Chosen-Update Security [DKW21]

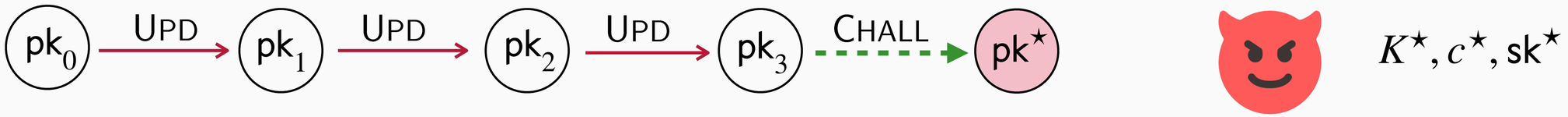
- The adversary can perform publicly verifiable chosen updates of the public key



Existing Definitions

Chosen-Update Security [DKW21]

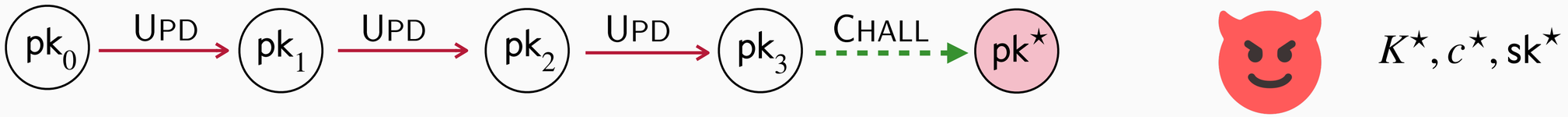
- The adversary can perform publicly verifiable chosen updates of the public key



Existing Definitions

Chosen-Update Security [DKW21]

- The adversary can perform publicly verifiable chosen updates of the public key



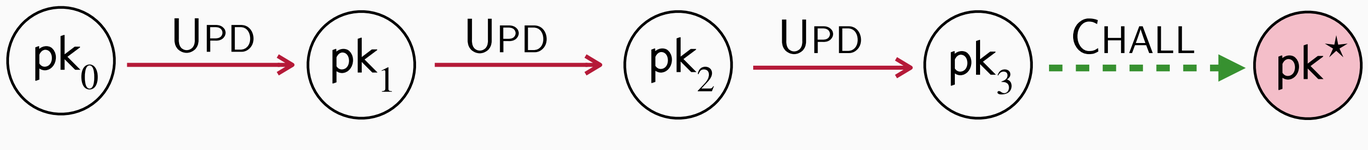
Member Security [AFM24]

- Takes into account that a public key is shared across users (as in MLS)
- The adversary can ask for honest encryptions or public verifiability chosen updates and create forks

Existing Definitions

Chosen-Update Security [DKW21]

- The adversary can perform publicly verifiable chosen updates of the public key





$K^\star, c^\star, \mathsf{sk}^\star$

Member Security [AFM24]

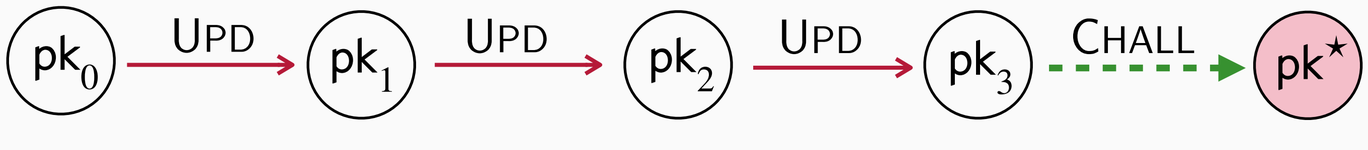
- Takes into account that a public key is shared across users (as in MLS)
- The adversary can ask for honest encryptions or public verifiability chosen updates and create forks



Existing Definitions

Chosen-Update Security [DKW21]

- The adversary can perform publicly verifiable chosen updates of the public key

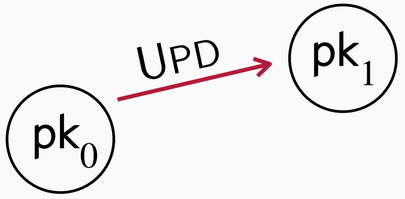




$K^\star, c^\star, \mathsf{sk}^\star$

Member Security [AFM24]

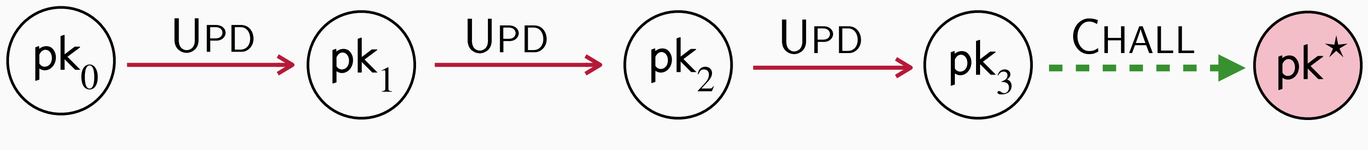
- Takes into account that a public key is shared across users (as in MLS)
- The adversary can ask for honest encryptions or public verifiability chosen updates and create forks



Existing Definitions

Chosen-Update Security [DKW21]

- The adversary can perform publicly verifiable chosen updates of the public key

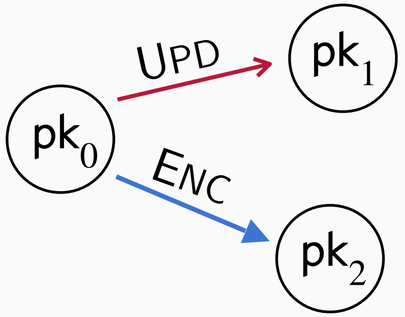




$K^\star, c^\star, \mathsf{sk}^\star$

Member Security [AFM24]

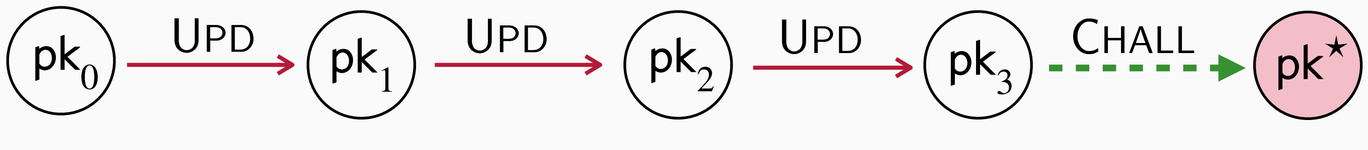
- Takes into account that a public key is shared across users (as in MLS)
- The adversary can ask for honest encryptions or public verifiability chosen updates and create forks



Existing Definitions

Chosen-Update Security [DKW21]

- The adversary can perform publicly verifiable chosen updates of the public key

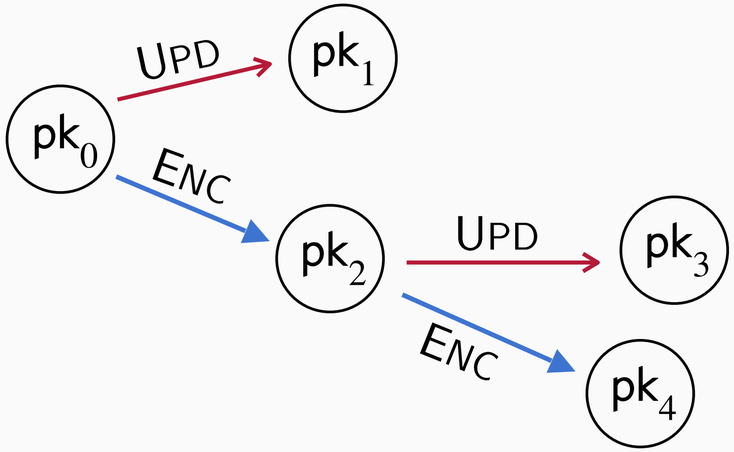




$K^\star, c^\star, \mathsf{sk}^\star$

Member Security [AFM24]

- Takes into account that a public key is shared across users (as in MLS)
- The adversary can ask for honest encryptions or public verifiability chosen updates and create forks



Existing Definitions

Chosen-Update Security [DKW21]

- The adversary can perform publicly verifiable chosen updates of the public key

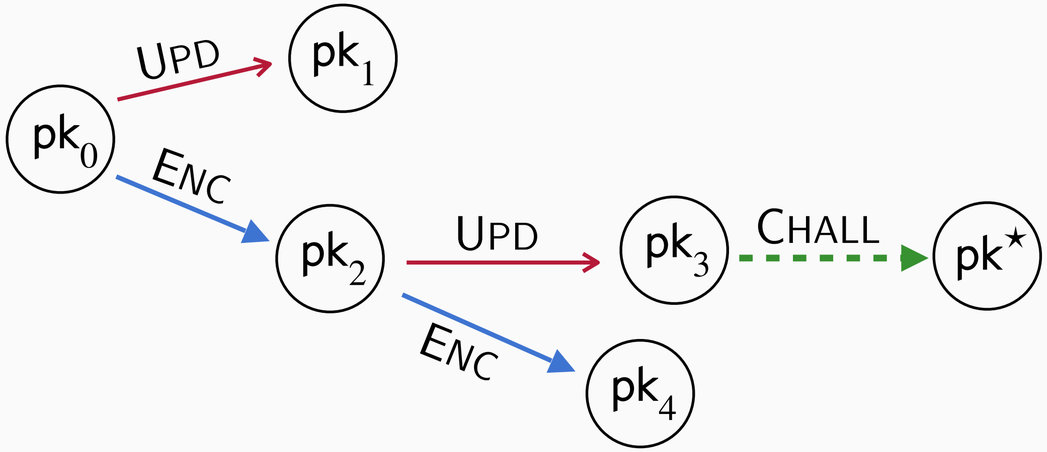




$K^\star, c^\star, \mathsf{sk}^\star$

Member Security [AFM24]

- Takes into account that a public key is shared across users (as in MLS)
- The adversary can ask for honest encryptions or public verifiability chosen updates and create forks



Existing Definitions

Chosen-Update Security [DKW21]

- The adversary can perform publicly verifiable chosen updates of the public key

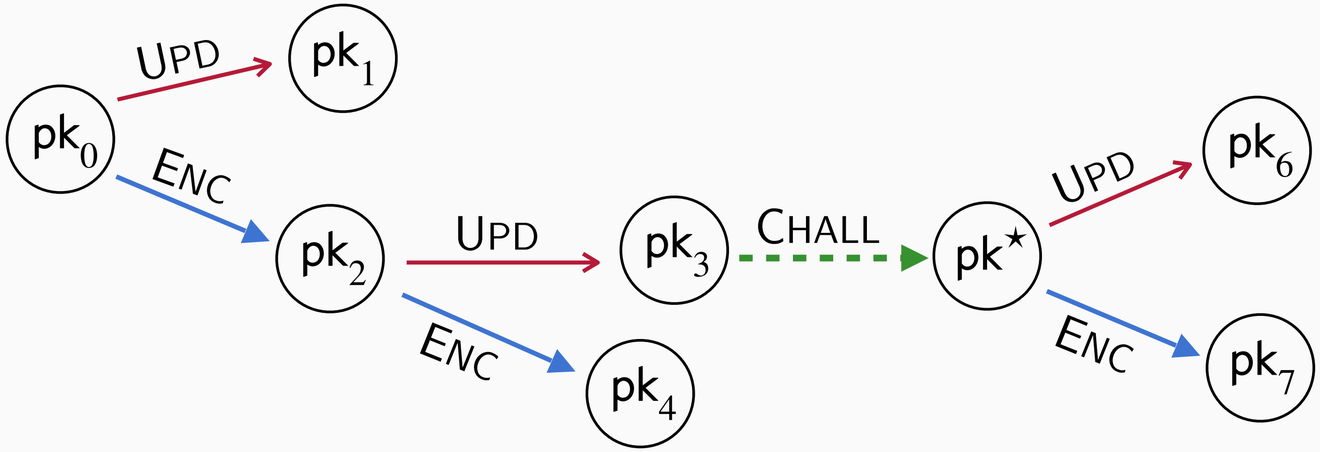




$K^\star, c^\star, \mathsf{sk}^\star$

Member Security [AFM24]

- Takes into account that a public key is shared across users (as in MLS)
- The adversary can ask for honest encryptions or public verifiability chosen updates and create forks



Existing Definitions

Chosen-Update Security [DKW21]

- The adversary can perform publicly verifiable chosen updates of the public key

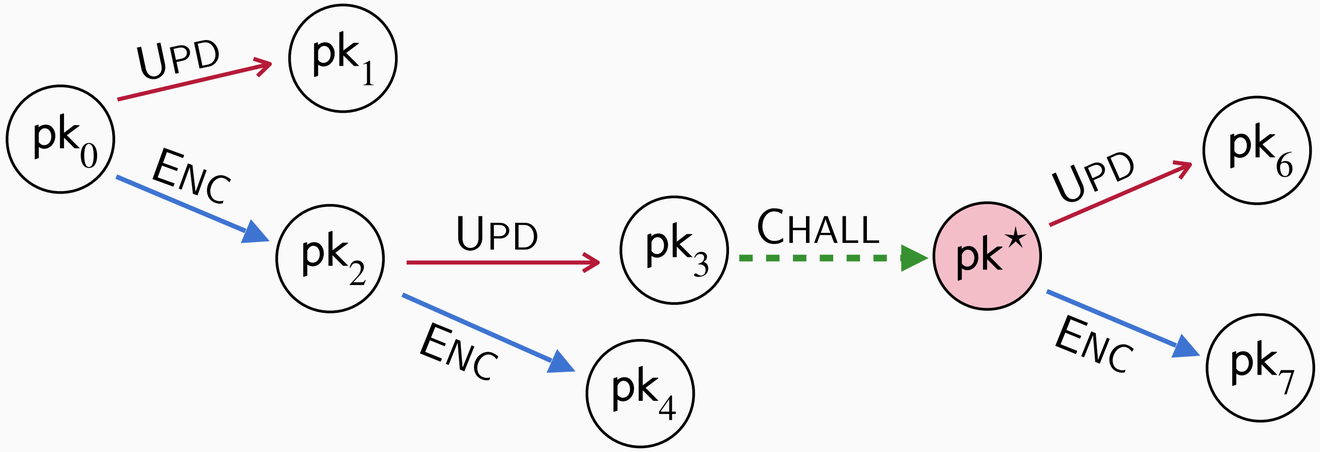




$K^\star, c^\star, \mathsf{sk}^\star$

Member Security [AFM24]

- Takes into account that a public key is shared across users (as in MLS)
- The adversary can ask for honest encryptions or public verifiability chosen updates and create forks





$K^\star, c^\star, \mathsf{sk}^\star$

Existing Definitions

Chosen-Update Security [DKW21]

- The adversary can perform publicly verifiable chosen updates of the public key

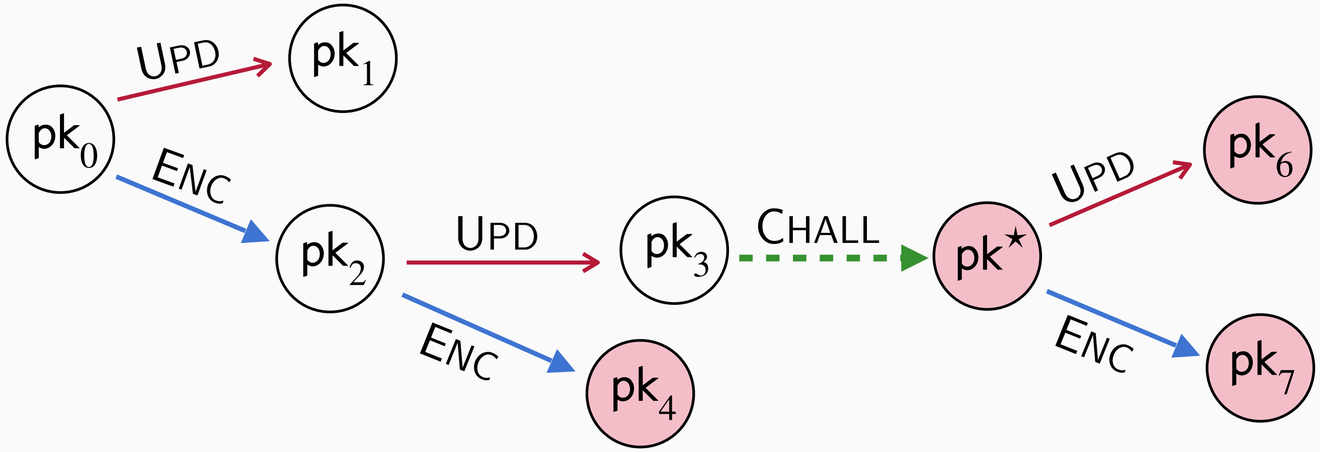




$K^\star, c^\star, \mathsf{sk}^\star$

Member Security [AFM24]

- Takes into account that a public key is shared across users (as in MLS)
- The adversary can ask for honest encryptions or public verifiability chosen updates and create forks





$K^\star, c^\star, \mathsf{sk}^\star$

Can also reveal/compute $\mathsf{sk}_4, \mathsf{sk}_6, \mathsf{sk}_7$

Prior Work

	Assumption	Member CCA	Unbounded updates	Practical(ish)

Prior Work

	Assumption	Member CCA	Unbounded updates	Practical(ish)
[ACDT19]	DH	✗	✓	✓

Prior Work

	Assumption	Member CCA	Unbounded updates	Practical(ish)
[ACDT19]	DH	✗	✓	✓
[DKW21]	Lattices	✗	✗	✗

Prior Work

	Assumption	Member CCA	Unbounded updates	Practical(ish)
[ACDT19]	DH	✗	✓	✓
[DKW21]	Lattices	✗	✗	✗
[APS23]	Lattices	✗	✗	✓

Parameters do not provide forking security

Prior Work

	Assumption	Member CCA	Unbounded updates	Practical(ish)
[ACDT19]	DH	✗	✓	✓
[DKW21]	Lattices	✗	✗	✗
[APS23]	Lattices	✗	✗	✓
[AFM24]	DH	✓	✓	✓

Parameters do not provide forking security

Prior Work

	Assumption	Member CCA	Unbounded updates	Practical(ish)
[ACDT19]	DH	✗	✓	✓
[DKW21]	Lattices	✗	✗	✗
[APS23]	Lattices	✗	✗	✓
[AFM24]	DH	✓	✓	✓
[ABL25]	Lattices	(✓)	✓	✗

← Parameters do not provide forking security

← Unbounded updates comes with efficiency penalty

Prior Work

	Assumption	Member CCA	Unbounded updates	Practical(ish)
[ACDT19]	DH	✗	✓	✓
[DKW21]	Lattices	✗	✗	✗
[APS23]	Lattices	✗	✗	✓
[AFM24]	DH	✓	✓	✓
[ABL25]	Lattices	(✓)	✓	✗
[BFK+26]	Isogenies	✗	✓	✓

← Parameters do not provide forking security

← Unbounded updates comes with efficiency penalty

Prior Work

	Assumption	Member CCA	Unbounded updates	Practical(ish)
[ACDT19]	DH	✗	✓	✓
[DKW21]	Lattices	✗	✗	✗
[APS23]	Lattices	✗	✗	✓
[AFM24]	DH	✓	✓	✓
[ABL25]	Lattices	(✓)	✓	✗
[BFK+26]	Isogenies	✗	✓	✓
Our Goal	Lattices	✓	✗	✓

Parameters do not provide forking security

Unbounded updates comes with efficiency penalty

Limited number of updates is acceptable for applications like MLS

Our Contributions

Modular UKEM construction suitable for lattice-based instantiation

Our Contributions

Modular UKEM construction suitable for lattice-based instantiation

New proof strategy for embedding challenges

Our Contributions

Modular UKEM construction suitable for lattice-based instantiation

Attacks against parameters of prior lattice-based schemes

New proof strategy for embedding challenges

Our Contributions

Modular UKEM construction suitable for lattice-based instantiation

Attacks against parameters of prior lattice-based schemes

New proof strategy for embedding challenges

Extension to (a weak form of) joiner security

Our Contributions

Modular UKEM construction suitable for lattice-based instantiation

Attacks against parameters of prior lattice-based schemes

New proof strategy for embedding challenges

Extension to (a weak form of) joiner security

Technical Overview

Our Construction

Initial key generation: $(\mathsf{pk}_0, \mathsf{sk}_0) \leftarrow^{\$} \mathsf{PKE.Gen}$

Our Construction

Initial key generation: $(\mathsf{pk}_0, \mathsf{sk}_0) \leftarrow^{\$} \mathsf{PKE.Gen}$

PKE is key-homomorphic

- function $pk : \mathcal{SK} \to \mathcal{PK}$ s.t.
  $pk(\mathsf{sk} + \mathsf{sk}') = pk(\mathsf{sk}) + pk(\mathsf{sk}')$

Our Construction

Initial key generation: $(\mathsf{pk}_0, \mathsf{sk}_0) \leftarrow^{\$} \mathsf{PKE.Gen}$

Encapsulation: FO transform and proof of knowledge (PoK)

PKE is key-homomorphic

- function $pk : \mathcal{SK} \to \mathcal{PK}$ s.t. $pk(\mathsf{sk} + \mathsf{sk}') = pk(\mathsf{sk}) + pk(\mathsf{sk}')$

Our Construction

Initial key generation: $(\mathsf{pk}_0, \mathsf{sk}_0) \leftarrow^{\$} \mathsf{PKE.Gen}$

Encapsulation: FO transform and proof of knowledge (PoK)



PKE is key-homomorphic

- function $pk : \mathcal{SK} \to \mathcal{PK}$ s.t. $pk(\mathsf{sk} + \mathsf{sk}') = pk(\mathsf{sk}) + pk(\mathsf{sk}')$

Our Construction

Initial key generation: $(\mathsf{pk}_0, \mathsf{sk}_0) \leftarrow^{\$} \mathsf{PKE.Gen}$

Encapsulation: FO transform and proof of knowledge (PoK)



PKE is key-homomorphic

- function $pk : \mathcal{SK} \to \mathcal{PK}$ s.t. $pk(\mathsf{sk} + \mathsf{sk}') = pk(\mathsf{sk}) + pk(\mathsf{sk}')$

Our Construction

Initial key generation: $(\mathsf{pk}_0, \mathsf{sk}_0) \leftarrow^{\$} \mathsf{PKE.Gen}$

Encapsulation: FO transform and proof of knowledge (PoK)



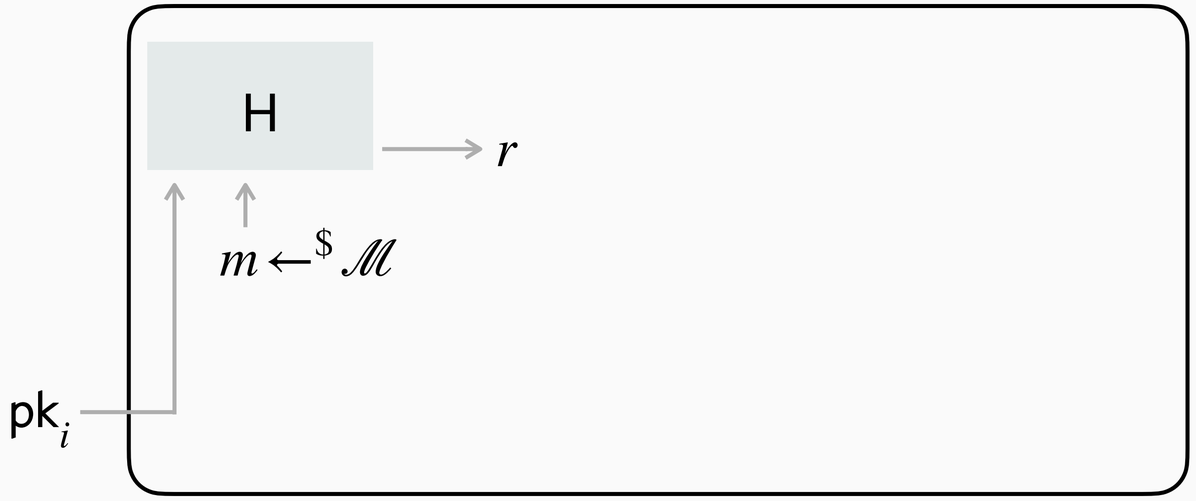
PKE is key-homomorphic

- function $pk : \mathcal{SK} \to \mathcal{PK}$ s.t. $pk(\mathsf{sk} + \mathsf{sk}') = pk(\mathsf{sk}) + pk(\mathsf{sk}')$

Our Construction

Initial key generation: $(\mathsf{pk}_0, \mathsf{sk}_0) \leftarrow^{\$} \mathsf{PKE.Gen}$

Encapsulation: FO transform and proof of knowledge (PoK)



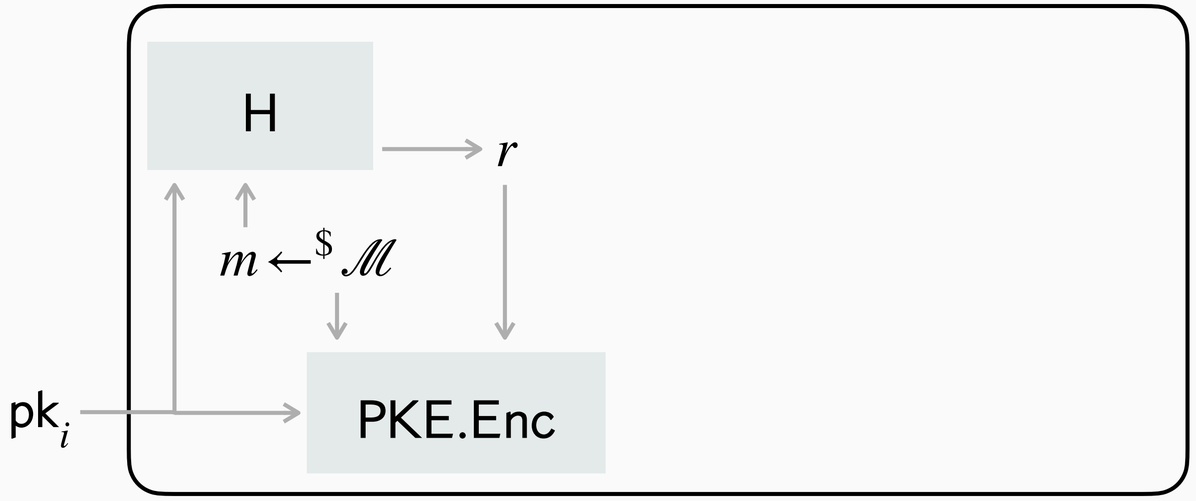
PKE is key-homomorphic

- function $pk : \mathcal{SK} \to \mathcal{PK}$ s.t. $pk(\mathsf{sk} + \mathsf{sk}') = pk(\mathsf{sk}) + pk(\mathsf{sk}')$

Our Construction

Initial key generation: $(\mathsf{pk}_0, \mathsf{sk}_0) \leftarrow^{\$} \mathsf{PKE.Gen}$

Encapsulation: FO transform and proof of knowledge (PoK)



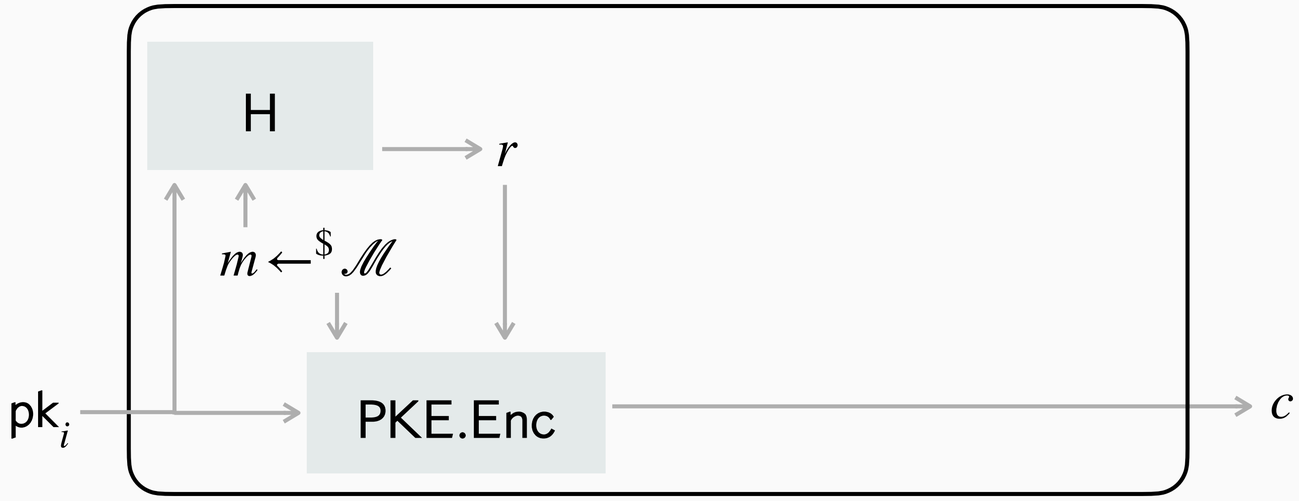
PKE is key-homomorphic

- function $pk : \mathcal{SK} \to \mathcal{PK}$ s.t. $pk(\mathsf{sk} + \mathsf{sk}') = pk(\mathsf{sk}) + pk(\mathsf{sk}')$

Our Construction

Initial key generation: $(\mathsf{pk}_0, \mathsf{sk}_0) \leftarrow^{\$} \mathsf{PKE.Gen}$

Encapsulation: FO transform and proof of knowledge (PoK)



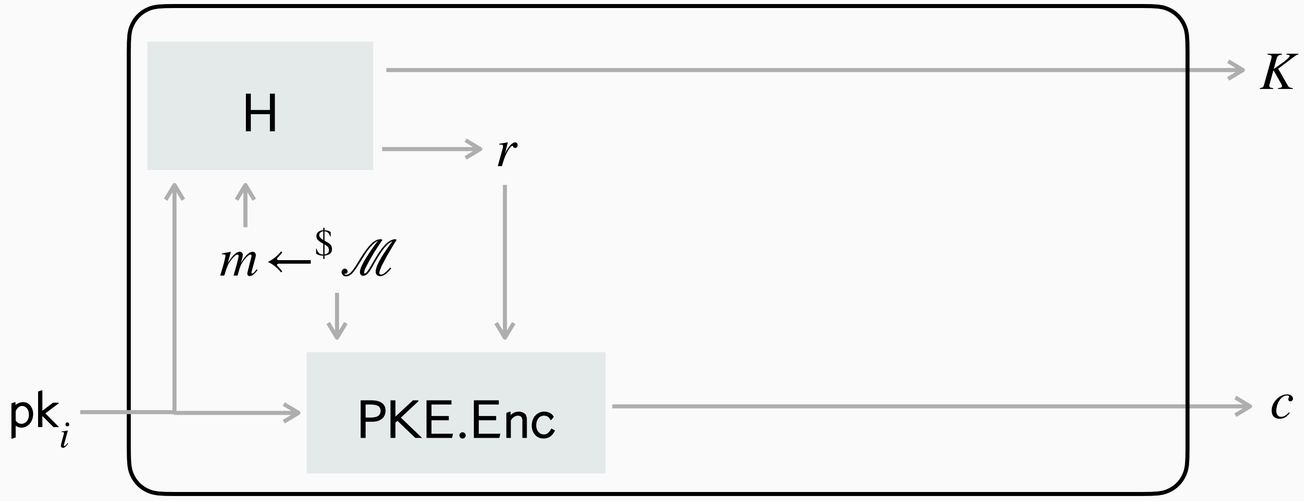
PKE is key-homomorphic

- function $pk : \mathcal{SK} \to \mathcal{PK}$ s.t. $pk(\mathsf{sk} + \mathsf{sk}') = pk(\mathsf{sk}) + pk(\mathsf{sk}')$

Our Construction

Initial key generation: $(\mathsf{pk}_0, \mathsf{sk}_0) \leftarrow^{\$} \mathsf{PKE.Gen}$

Encapsulation: FO transform and proof of knowledge (PoK)



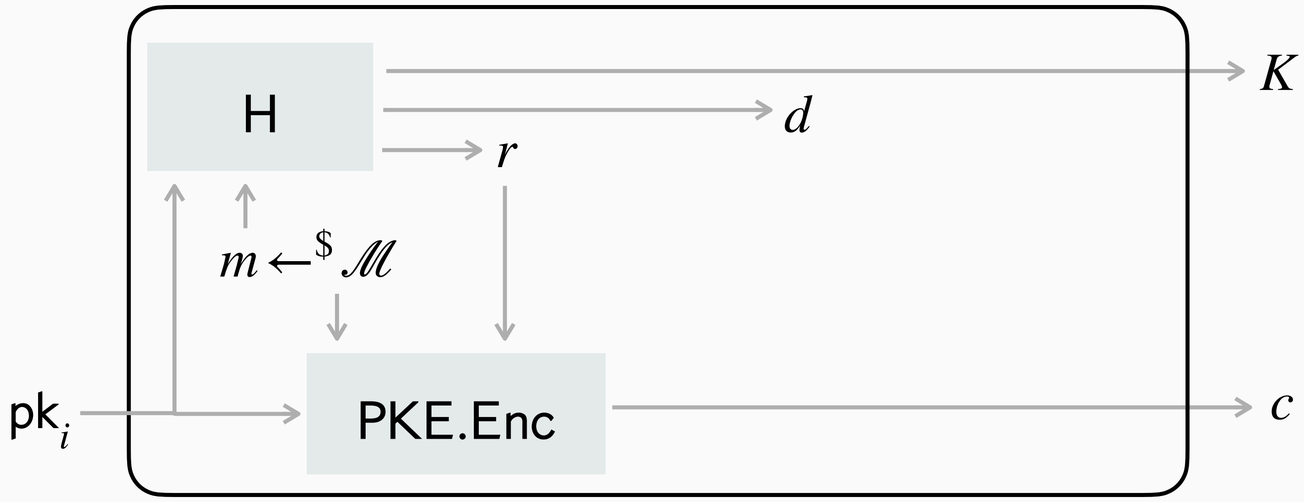
PKE is key-homomorphic

- function $pk : \mathcal{SK} \to \mathcal{PK}$ s.t.
  $pk(\mathsf{sk} + \mathsf{sk}') = pk(\mathsf{sk}) + pk(\mathsf{sk}')$

Our Construction

Initial key generation: $(\mathsf{pk}_0, \mathsf{sk}_0) \leftarrow^{\$} \mathsf{PKE.Gen}$

Encapsulation: FO transform and proof of knowledge (PoK)



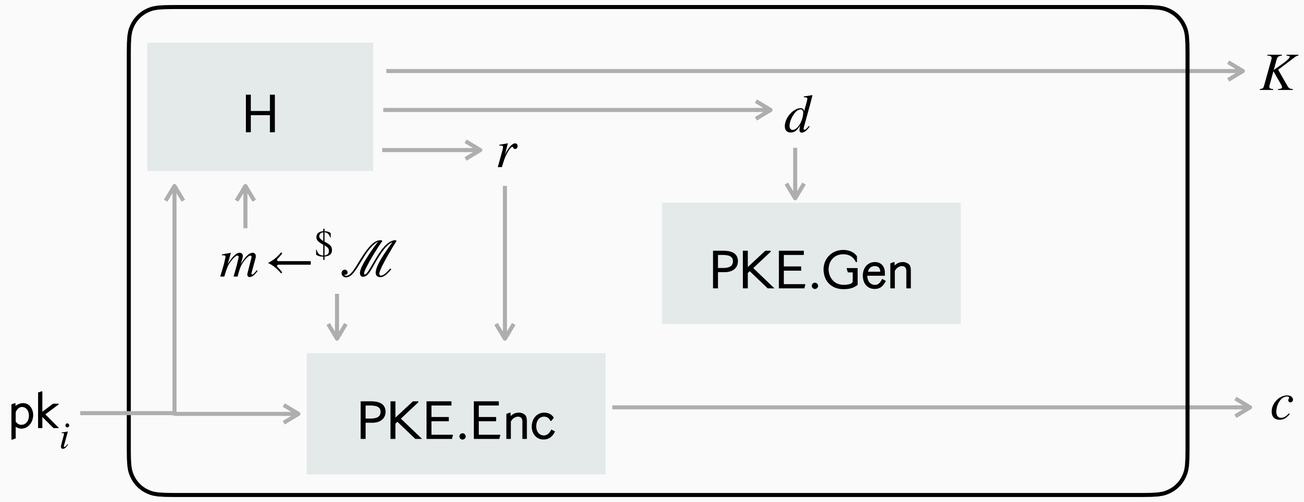
PKE is key-homomorphic

- function $pk : \mathcal{SK} \to \mathcal{PK}$ s.t.
  $pk(\mathsf{sk} + \mathsf{sk}') = pk(\mathsf{sk}) + pk(\mathsf{sk}')$

Our Construction

Initial key generation: $(\mathsf{pk}_0, \mathsf{sk}_0) \leftarrow^{\$} \mathsf{PKE.Gen}$

Encapsulation: FO transform and proof of knowledge (PoK)



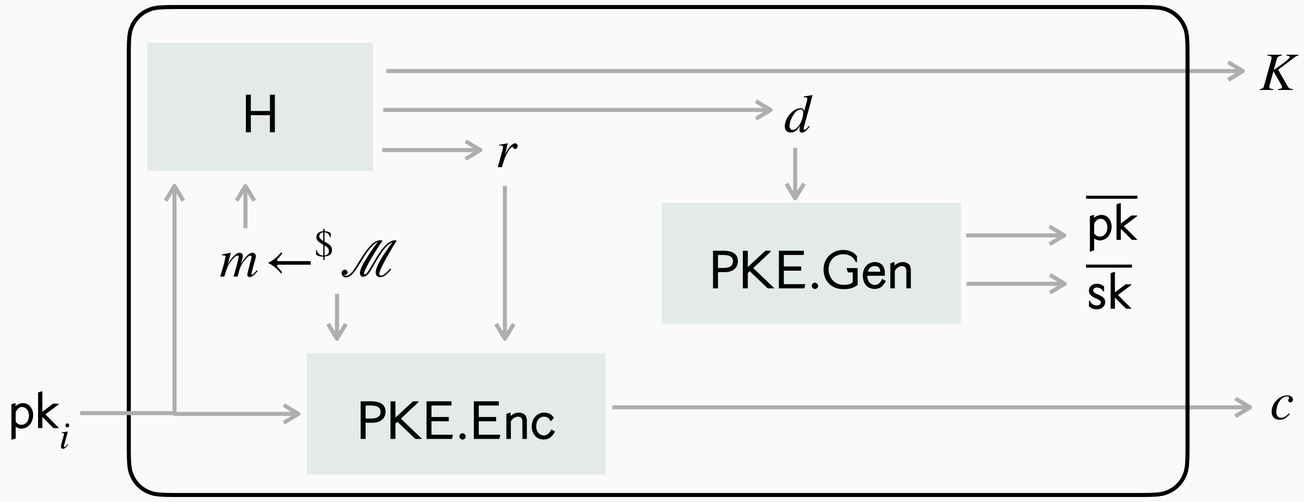
PKE is key-homomorphic

- function $pk : \mathcal{SK} \to \mathcal{PK}$ s.t. $pk(\mathsf{sk} + \mathsf{sk}') = pk(\mathsf{sk}) + pk(\mathsf{sk}')$

Our Construction

Initial key generation: $(\mathsf{pk}_0, \mathsf{sk}_0) \leftarrow^{\$} \mathsf{PKE.Gen}$

Encapsulation: FO transform and proof of knowledge (PoK)



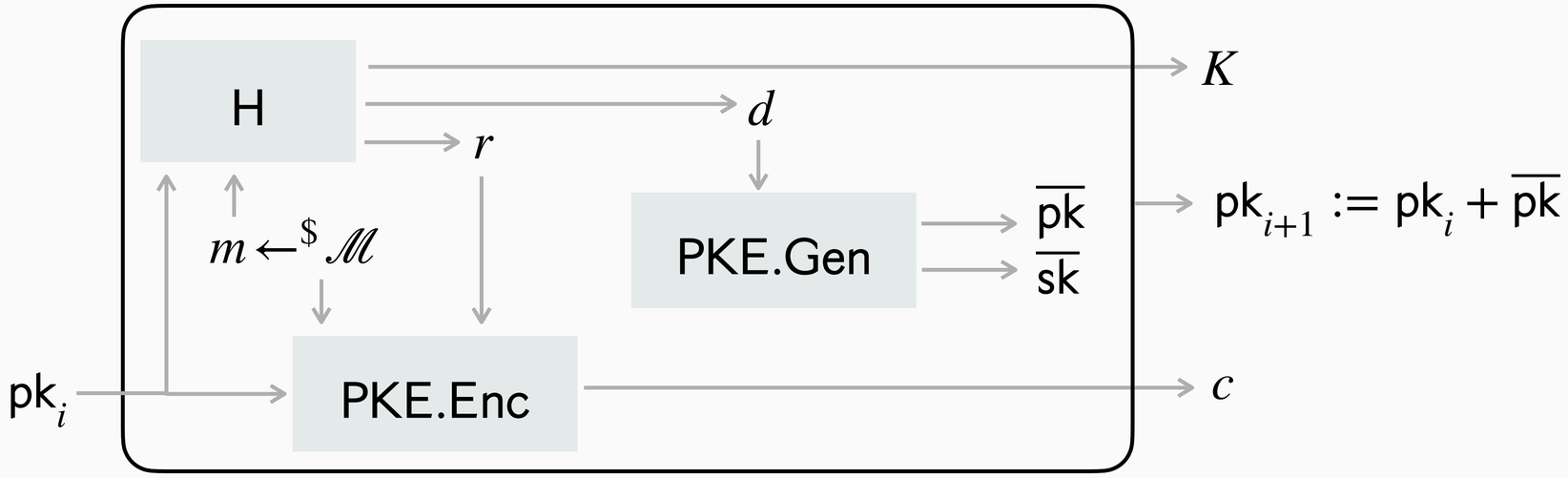
PKE is key-homomorphic

- function $pk : \mathcal{SK} \to \mathcal{PK}$ s.t. $pk(\mathsf{sk} + \mathsf{sk}') = pk(\mathsf{sk}) + pk(\mathsf{sk}')$

Our Construction

Initial key generation: $(\mathsf{pk}_0, \mathsf{sk}_0) \leftarrow^{\$} \mathsf{PKE.Gen}$

Encapsulation: FO transform and proof of knowledge (PoK)



PKE is key-homomorphic

- function $pk : \mathcal{SK} \to \mathcal{PK}$ s.t. $pk(\mathsf{sk} + \mathsf{sk}') = pk(\mathsf{sk}) + pk(\mathsf{sk}')$

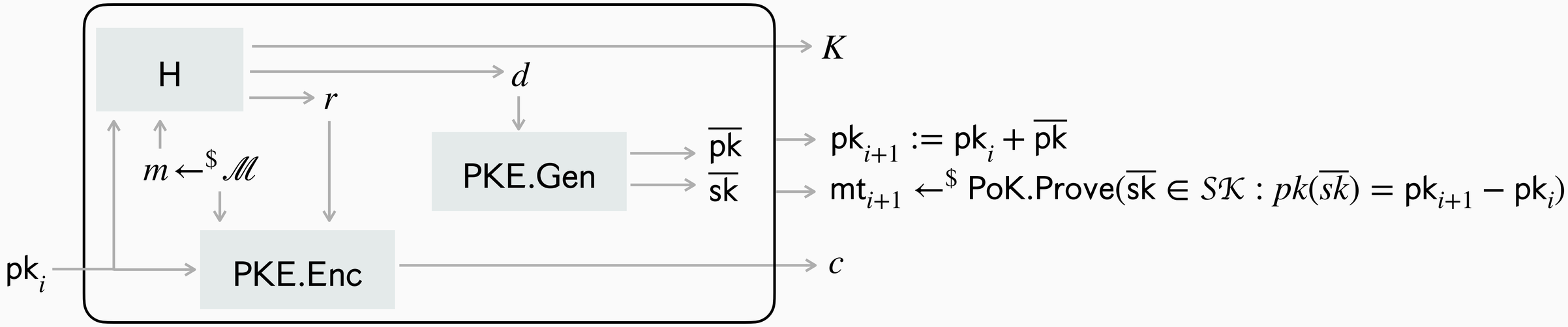
Our Construction

Initial key generation: $(\mathsf{pk}_0, \mathsf{sk}_0) \leftarrow^{\$} \mathsf{PKE.Gen}$

Encapsulation: FO transform and proof of knowledge (PoK)

PKE is key-homomorphic

- function $pk : \mathcal{SK} \to \mathcal{PK}$ s.t. $pk(\mathsf{sk} + \mathsf{sk}') = pk(\mathsf{sk}) + pk(\mathsf{sk}')$



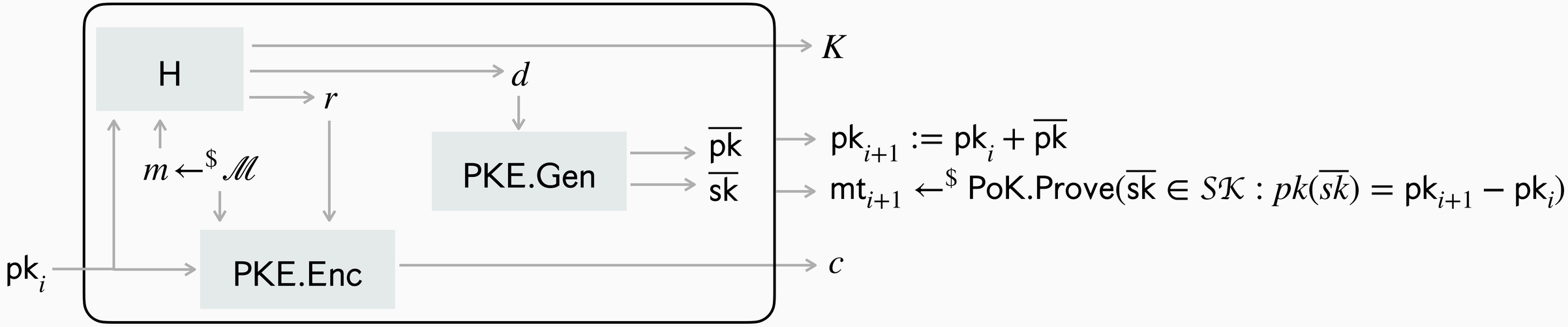
Our Construction

Initial key generation: $(\mathsf{pk}_0, \mathsf{sk}_0) \leftarrow^{\$} \mathsf{PKE.Gen}$

Encapsulation: FO transform and proof of knowledge (PoK)

PKE is key-homomorphic

- function $pk : \mathcal{SK} \to \mathcal{PK}$ s.t. $pk(\mathsf{sk} + \mathsf{sk}') = pk(\mathsf{sk}) + pk(\mathsf{sk}')$



Decapsulation: decrypt and re-encrypt, update secret key as $\mathsf{sk}_{i+1} := \mathsf{sk}_i + \overline{\mathsf{sk}}$

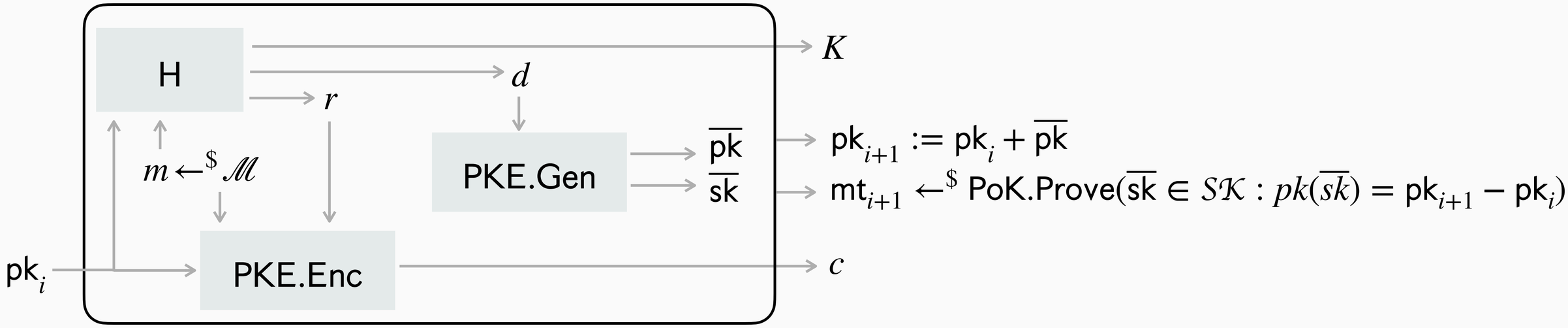
Our Construction

Initial key generation: $(\mathsf{pk}_0, \mathsf{sk}_0) \leftarrow^{\$} \mathsf{PKE.Gen}$

Encapsulation: FO transform and proof of knowledge (PoK)

PKE is key-homomorphic

- function $pk : \mathcal{SK} \to \mathcal{PK}$ s.t. $pk(\mathsf{sk} + \mathsf{sk}') = pk(\mathsf{sk}) + pk(\mathsf{sk}')$



Decapsulation: decrypt and re-encrypt, update secret key as $\mathsf{sk}_{i+1} := \mathsf{sk}_i + \overline{\mathsf{sk}}$

Member-tag verification: $0/1 \leftarrow \mathsf{PoK.Verify}(\mathsf{pk}_i, \mathsf{pk}_{i+1}, \mathsf{mt}_{i+1})$

Properties of PKE for Member Security

Shifty-IND-CPA

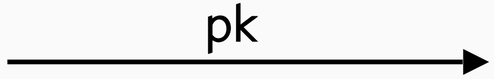


Properties of PKE for Member Security

Shifty-IND-CPA

$(\mathsf{pk}, \mathsf{sk}) \leftarrow^{\$} \mathsf{Gen}$

$b \leftarrow^{\$} \{0,1\}$



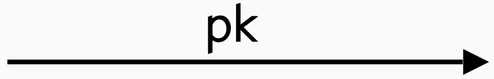


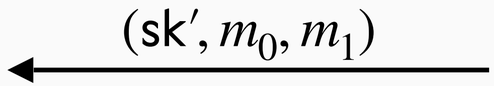
Properties of PKE for Member Security

Shifty-IND-CPA

$(\mathsf{pk}, \mathsf{sk}) \leftarrow^{\$} \mathsf{Gen}$

$b \leftarrow^{\$} \{0,1\}$







Properties of PKE for Member Security

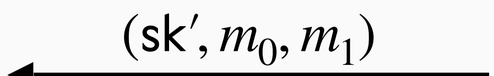
Shifty-IND-CPA

$(\mathsf{pk}, \mathsf{sk}) \leftarrow^{\$} \mathsf{Gen}$

$b \leftarrow^{\$} \{0,1\}$







$c \leftarrow^{\$} \mathsf{Enc}(\mathsf{pk} + pk(\mathsf{sk}'), m_b)$

Properties of PKE for Member Security

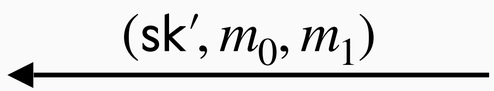
Shifty-IND-CPA

$(\mathsf{pk}, \mathsf{sk}) \leftarrow^{\$} \mathsf{Gen}$

$b \leftarrow^{\$} \{0,1\}$







$c \leftarrow^{\$} \mathsf{Enc}(\mathsf{pk} + pk(\mathsf{sk}'), m_b)$

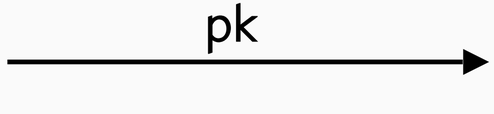
$(\overline{\mathsf{pk}}, \overline{\mathsf{sk}}) \leftarrow^{\$} \mathsf{Gen}$

Properties of PKE for Member Security

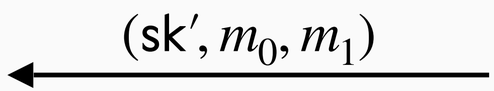
Shifty-IND-CPA

$(\mathsf{pk}, \mathsf{sk}) \leftarrow^{\$} \mathsf{Gen}$

$b \leftarrow^{\$} \{0,1\}$

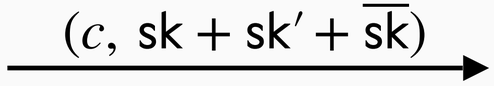






$c \leftarrow^{\$} \mathsf{Enc}(\mathsf{pk} + pk(\mathsf{sk}'), m_b)$

$(\overline{\mathsf{pk}}, \overline{\mathsf{sk}}) \leftarrow^{\$} \mathsf{Gen}$



Properties of PKE for Member Security

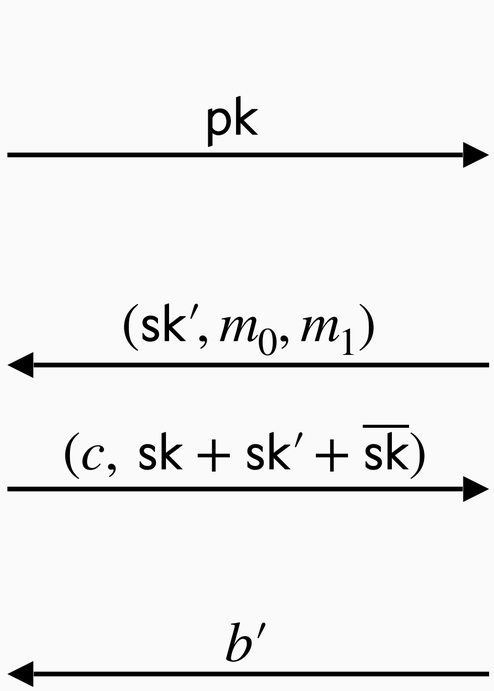
Shifty-IND-CPA

$(\mathsf{pk}, \mathsf{sk}) \leftarrow^{\$} \mathsf{Gen}$

$b \leftarrow^{\$} \{0,1\}$

$c \leftarrow^{\$} \mathsf{Enc}(\mathsf{pk} + pk(\mathsf{sk}'), m_b)$

$(\overline{\mathsf{pk}}, \overline{\mathsf{sk}}) \leftarrow^{\$} \mathsf{Gen}$





Properties of PKE for Member Security

Shifty-IND-CPA

$(\mathsf{pk}, \mathsf{sk}) \leftarrow^{\$} \mathsf{Gen}$

$b \leftarrow^{\$} \{0,1\}$

pk →

← (sk', m_0, m_1)

$c \leftarrow^{\$} \mathsf{Enc}(\mathsf{pk} + pk(\mathsf{sk}'), m_b)$

$(\overline{\mathsf{pk}}, \overline{\mathsf{sk}}) \leftarrow^{\$} \mathsf{Gen}$

→ $(c,\ \mathsf{sk} + \mathsf{sk}' + \overline{\mathsf{sk}})$

← b'

$\mathcal{A}$ wins if $b = b'$



Properties of PKE for Member Security

Shifty-IND-CPA

$(\mathsf{pk}, \mathsf{sk}) \leftarrow^{\$} \mathsf{Gen}$

$b \leftarrow^{\$} \{0,1\}$

pk →

← (sk', m_0, m_1)

$c \leftarrow^{\$} \mathsf{Enc}(\mathsf{pk} + pk(\mathsf{sk}'), m_b)$

$(\overline{\mathsf{pk}}, \overline{\mathsf{sk}}) \leftarrow^{\$} \mathsf{Gen}$

→ $(c,\ \mathsf{sk} + \mathsf{sk}' + \overline{\mathsf{sk}})$

← b'

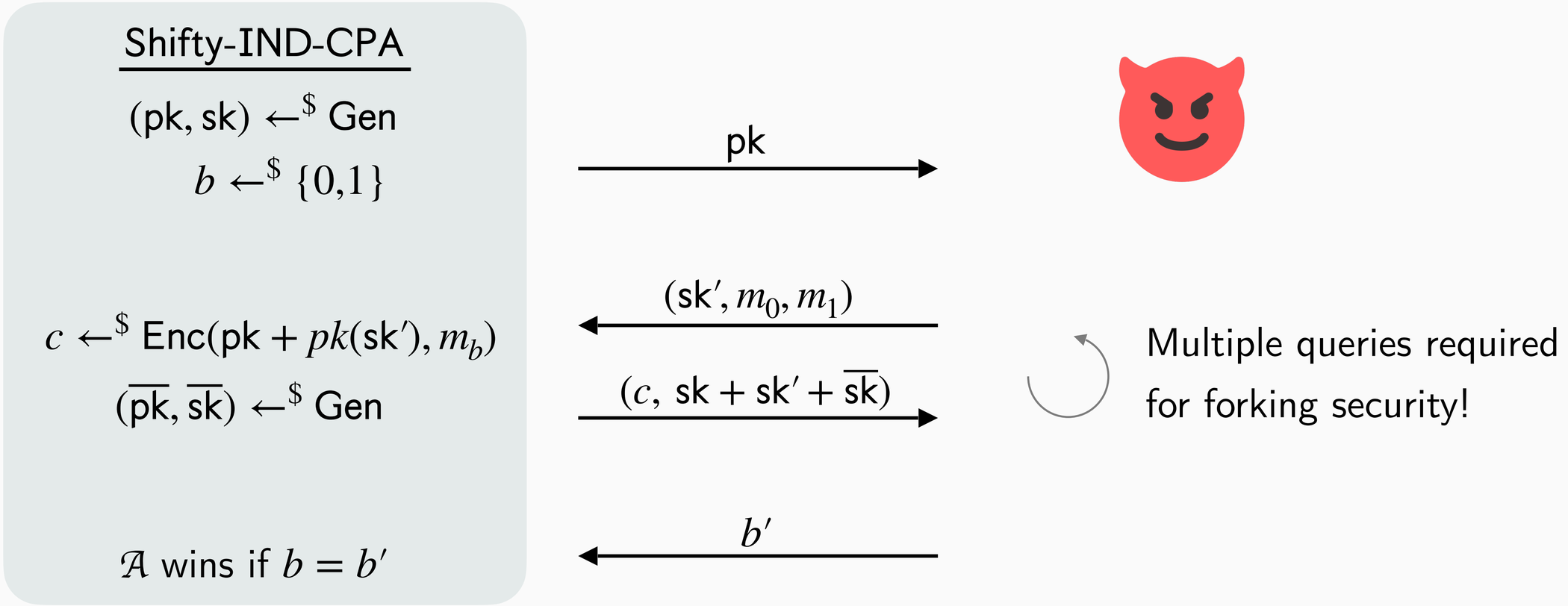
$\mathcal{A}$ wins if $b = b'$





Multiple queries required for forking security!

Properties of PKE for Member Security



New proof strategy: Shifty security instead of random self-reducibility

- Updates change distribution of sk

  $\Rightarrow$ we embed challenge as the update $\overline{\mathsf{sk}}$ instead of sk_{i+1} as in [AFM24]

Concrete Lattice-Based Instantiation

Key-Homomorphic PKE

Proof of Knowledge

Concrete Lattice-Based Instantiation

Key-Homomorphic PKE

- Based on CPA-secure UPKE from [APS23]
  - Adapted proof to multi-challenge setting, which affects choice of parameters
- Concrete parameters for the module LWE variant

Proof of Knowledge

Concrete Lattice-Based Instantiation

Key-Homomorphic PKE

- Based on CPA-secure UPKE from [APS23]
  - Adapted proof to multi-challenge setting, which affects choice of parameters
- Concrete parameters for the module LWE variant

Proof of Knowledge

- Use "encrypt-to-the-sky" paradigm for straight-line simulation-extractability
- Estimate proof sizes via LaZer library [LSS24]

Concrete Lattice-Based Instantiation

Key-Homomorphic PKE

- Based on CPA-secure UPKE from [APS23]
  - Adapted proof to multi-challenge setting, which affects choice of parameters
- Concrete parameters for the module LWE variant

Proof of Knowledge

- Use "encrypt-to-the-sky" paradigm for straight-line simulation-extractability
- Estimate proof sizes via LaZer library [LSS24]

Our final scheme improves upon efficiency despite requiring larger parameters (for forking security)

- For small number of updates: **61 KB** [APS23] vs. **53 KB** [ours]
- Updates are performed differently and simpler NIZK statement

Summary

We present the first practical(ish) lattice-based UKEM secure in the setting with forks, therefore suitable for the group messaging setting.

Summary

We present the first practical(ish) lattice-based UKEM secure in the setting with forks, therefore suitable for the group messaging setting.

- Compared to existing DH and lattice constructions, we
  - modularize and simplify the construction (in particular the NIZK statement)
  - adapt the proof strategy
  - strengthen the assumptions to achieve forking security

Summary

We present the first practical(ish) lattice-based UKEM secure in the setting with forks, therefore suitable for the group messaging setting.

- Compared to existing DH and lattice constructions, we
  - modularize and simplify the construction (in particular the NIZK statement)
  - adapt the proof strategy
  - strengthen the assumptions to achieve forking security

Additionally, we provide

- Attacks against parameters of previous schemes in the forking setting
- An extension to (a weak form of) joiner security

Summary

We present the first practical(ish) lattice-based UKEM secure in the setting with forks, therefore suitable for the group messaging setting.

- Compared to existing DH and lattice constructions, we
  - modularize and simplify the construction (in particular the NIZK statement)
  - adapt the proof strategy
  - strengthen the assumptions to achieve forking security

Additionally, we provide

- Attacks against parameters of previous schemes in the forking setting
- An extension to (a weak form of) joiner security

ia.cr/2025/365

Summary

We present the first practical(ish) lattice-based UKEM secure in the setting with forks, therefore suitable for the group messaging setting.

- Compared to existing DH and lattice constructions, we
  - modularize and simplify the construction (in particular the NIZK statement)
  - adapt the proof strategy
  - strengthen the assumptions to achieve forking security

Additionally, we provide

- Attacks against parameters of previous schemes in the forking setting
- An extension to (a weak form of) joiner security

ia.cr/2025/365

Thank you!