Échange de clé authentifié par mot de passe post-quantique

Emre Ucar Guillaume Chirache Timothée Fisher Thomas Sauvage
Christopher Calvet

Encadrés par Melissa Rossi

Coordinateur au LIX : Gilles Schaeffer

2023-2024

Partie 1

Introduction

Introduction

Travail basé sur la publication d'une équipe de chercheurs à laquelle appartient notre tutrice Mme Mélissa Rossi.

Beguinet, H., Chevalier, C., Pointcheval, D., Ricosset, T. et Rossi, M. (2023). GeT a CAKE : Generic Transformations from Key Encaspulation Mechanisms to Password Authenticated Key Exchanges. 21st International Conference on Applied Cryptography and Network Security (2023).

→ **Preuve de faisabilité** d'un échange de clés authentifié par mot de passe (PAKE) à partir du mécanisme d'encapsulation de clés `Kyber`.

Nos objectifs :

- Comprendre le protocole proposé et ses contraintes de sécurité.
- Résoudre les problèmes théoriques d'implémentation qui se posent (*ideal cipher* notamment).
- Faire l'implémentation en C et en Python.
- Réaliser un démonstrateur physique et des études de performance.

Plan

1. Introduction

2. Présentation des PAKE post-quantiques
   - L'authentification sur les échanges de clé
   - Sécurité d'un PAKE
   - Démonstration avec une Raspberry Pi

3. Le problème de l'*ideal cipher*

4. Utilisation de l'*ideal cipher* pour obtenir **Epk** et **Ec**
   - Clé publique
   - Chiffré

5. Hybridation

6. Implémentation

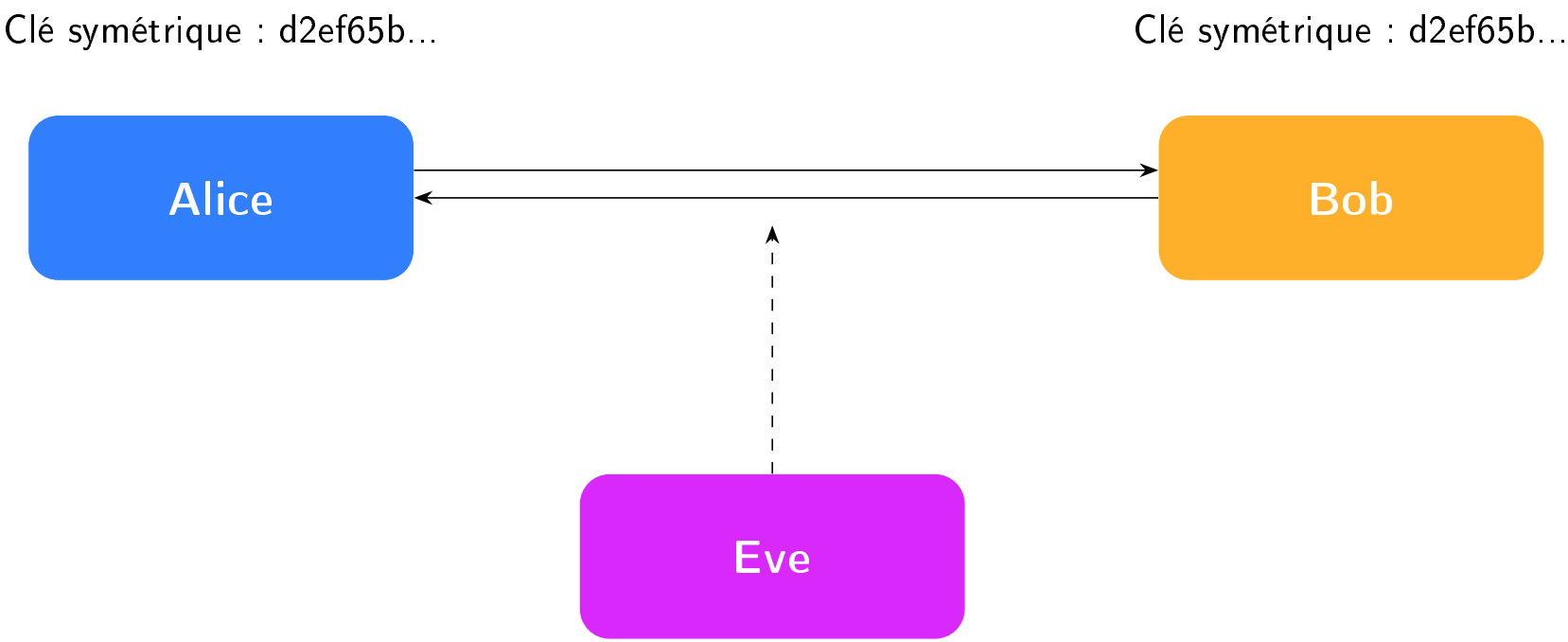
7. Conclusion

Partie 2

Présentation des PAKE post-quantiques

Le chiffrement symétrique

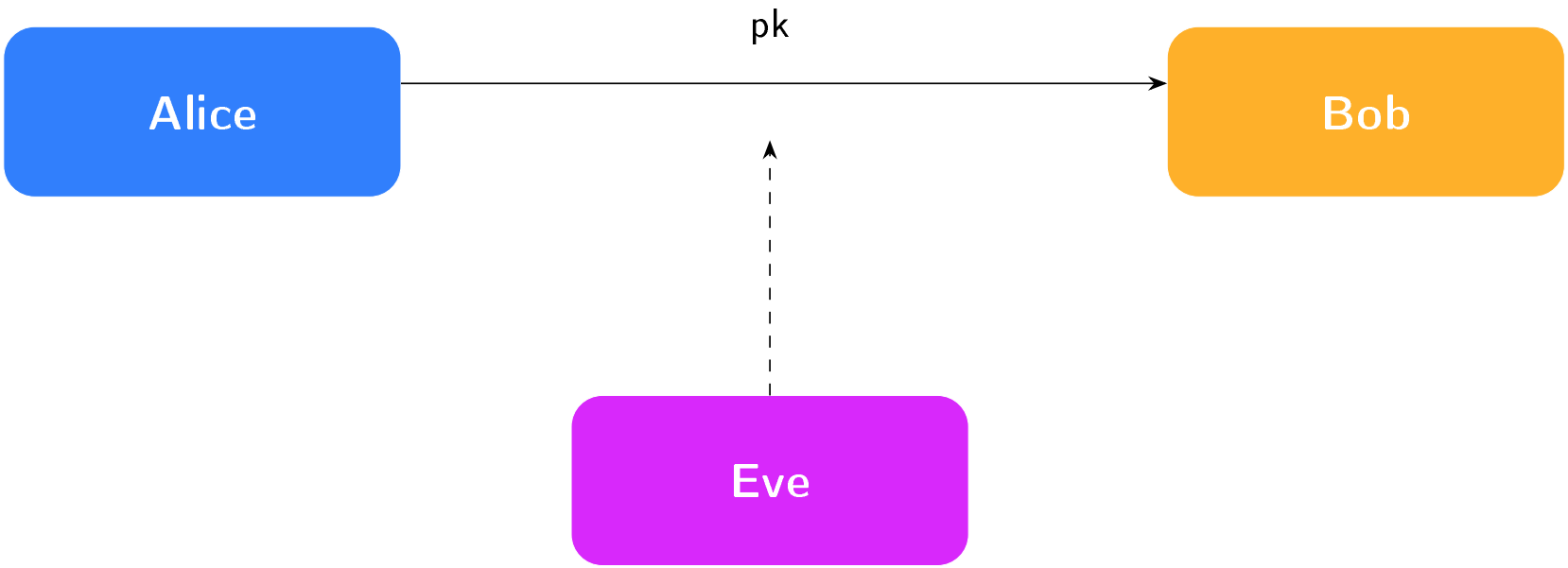
- Chiffrement symétrique : Même clé pour chiffrer et déchiffrer
- Comment échanger la clé ?



Key Encapsulation Mechanism (KEM)

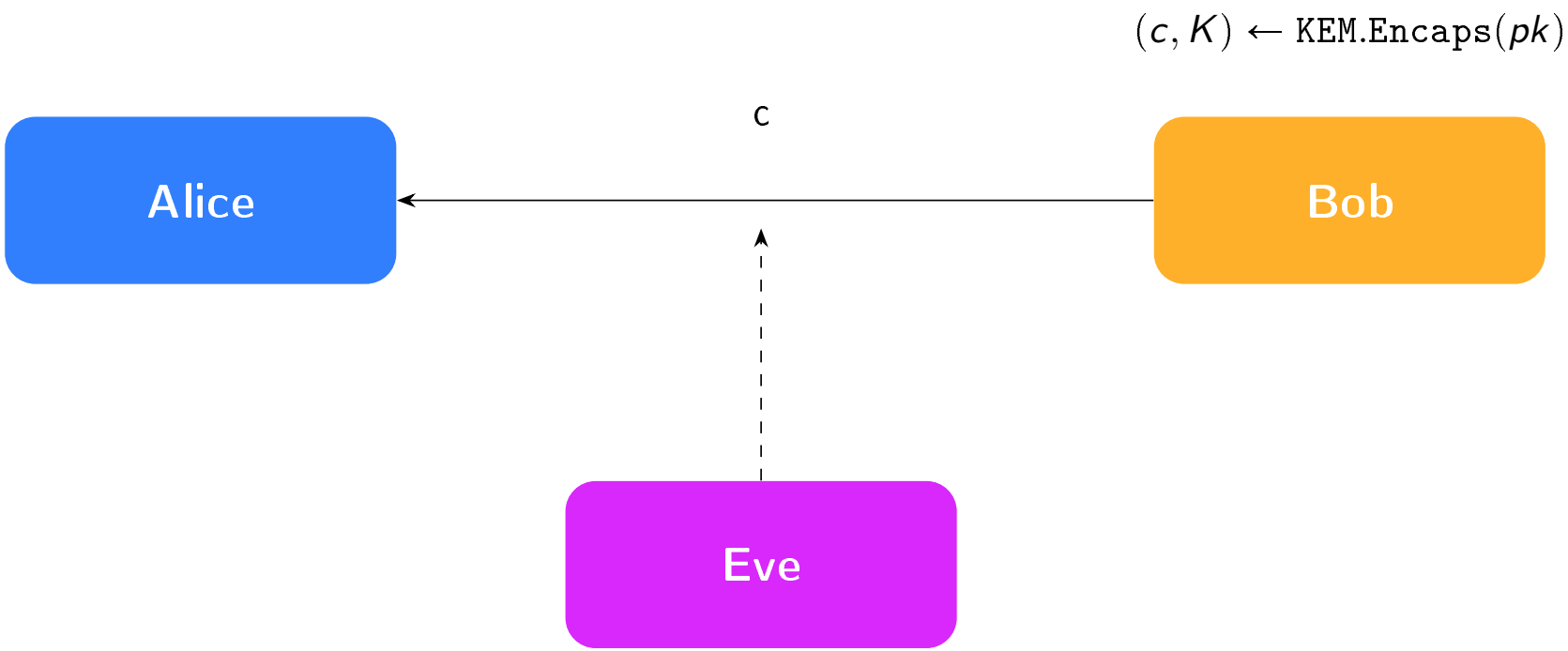
- KEM : Mécanisme par lequel deux parties échangent une clé symétrique
- Utilisation de la cryptographie asymétrique
- `KEM.KeyGen`, `KEM.Encaps`, `KEM.Decaps`

$(pk, sk) \leftarrow \texttt{KEM.KeyGen}()$



Key Encapsulation Mechanism (KEM)

- KEM : Mécanisme par lequel deux parties échangent une clé symétrique
- Utilisation de la cryptographie asymétrique
- `KEM.KeyGen`, `KEM.Encaps`, `KEM.Decaps`



Key Encapsulation Mechanism (KEM)

- KEM : Mécanisme par lequel deux parties échangent une clé symétrique
- Utilisation de la cryptographie asymétrique
- `KEM.KeyGen`, `KEM.Encaps`, `KEM.Decaps`

$K \leftarrow \texttt{KEM.Decaps}(sk, c)$

$d2ef65b... \leftarrow \texttt{hash}(K)$

$d2ef65b... \leftarrow \texttt{hash}(K)$

Alice

Bob

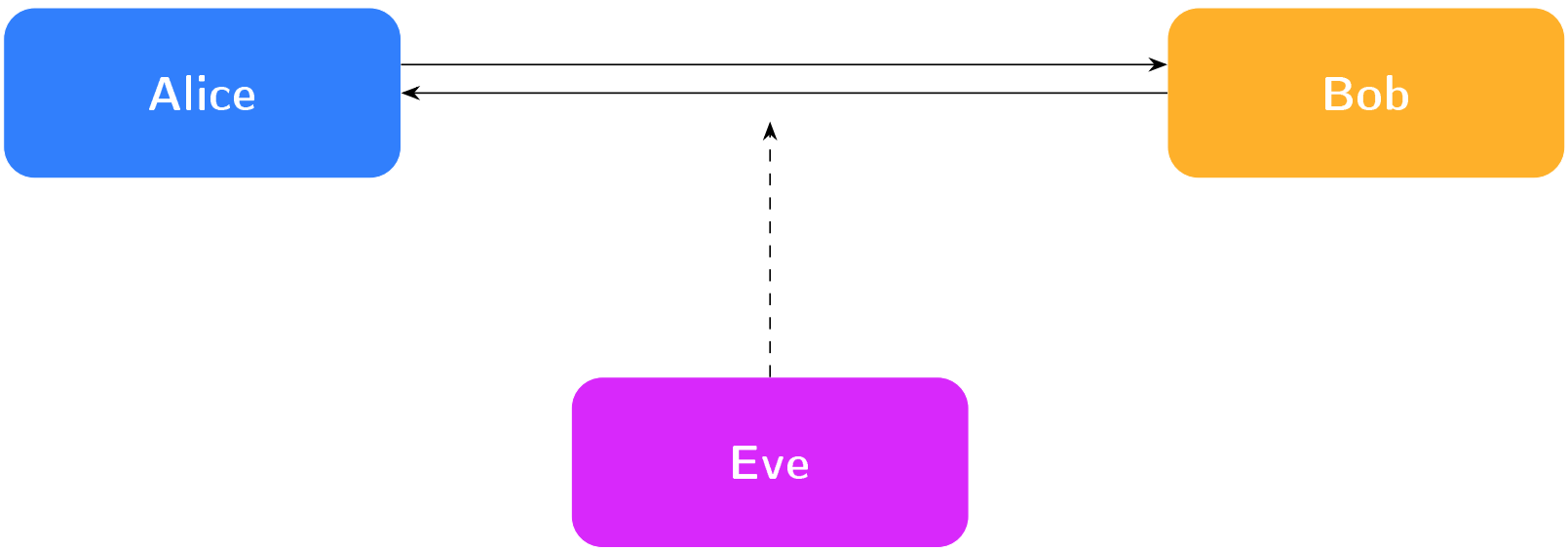
Eve

Key Encapsulation Mechanism (KEM)

- KEM : Mécanisme par lequel deux parties échangent une clé symétrique
- Utilisation de la cryptographie asymétrique
- `KEM.KeyGen`, `KEM.Encaps`, `KEM.Decaps`
- Si Eve ne fait qu'écouter, elle n'accède pas à la clé symétrique

Clé symétrique : d2ef65b...

Clé symétrique : d2ef65b...



L'authentification des échanges de clé

- S'assurer que l'interlocuteur est bien celui qu'il prétend être
- L'attaque par homme du milieu

Clés publique et privée de Alice

Clés publique et privée de Eve

L'authentification des échanges de clé

- S'assurer que l'interlocuteur est bien celui qu'il prétend être
- L'attaque par homme du milieu

Clés publique et privée de Alice

Clés publique et privée de Eve

L'authentification des échanges de clé

- S'assurer que l'interlocuteur est bien celui qu'il prétend être
- L'attaque par homme du milieu

Clé symétrique : 4f5bd...

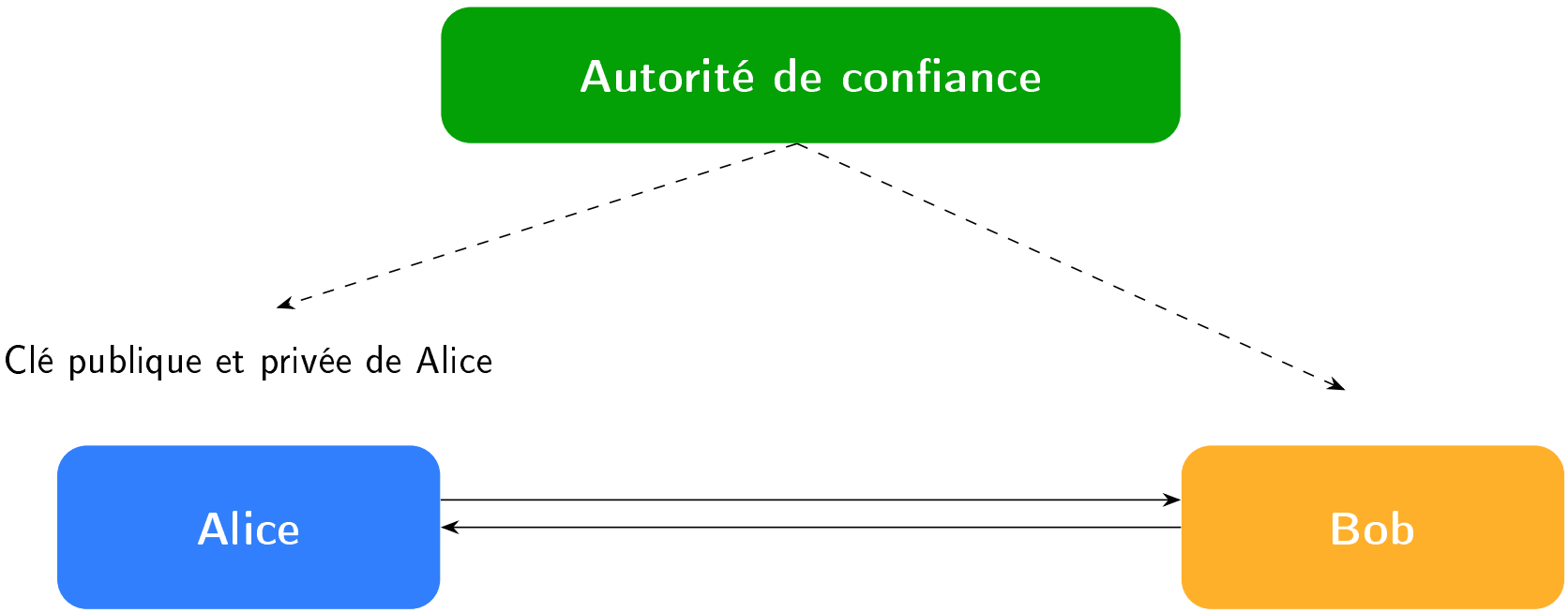
Clé symétrique : 7ed56...

Clé symétrique : 4f5bd...

Clé symétrique : 7ed56...

Première solution, la PKI

- PKI : *Public Key Infrastructure* (Infrastructure à clé publique)
- Un tiers de confiance certifie la clé publique d'Alice pour Bob



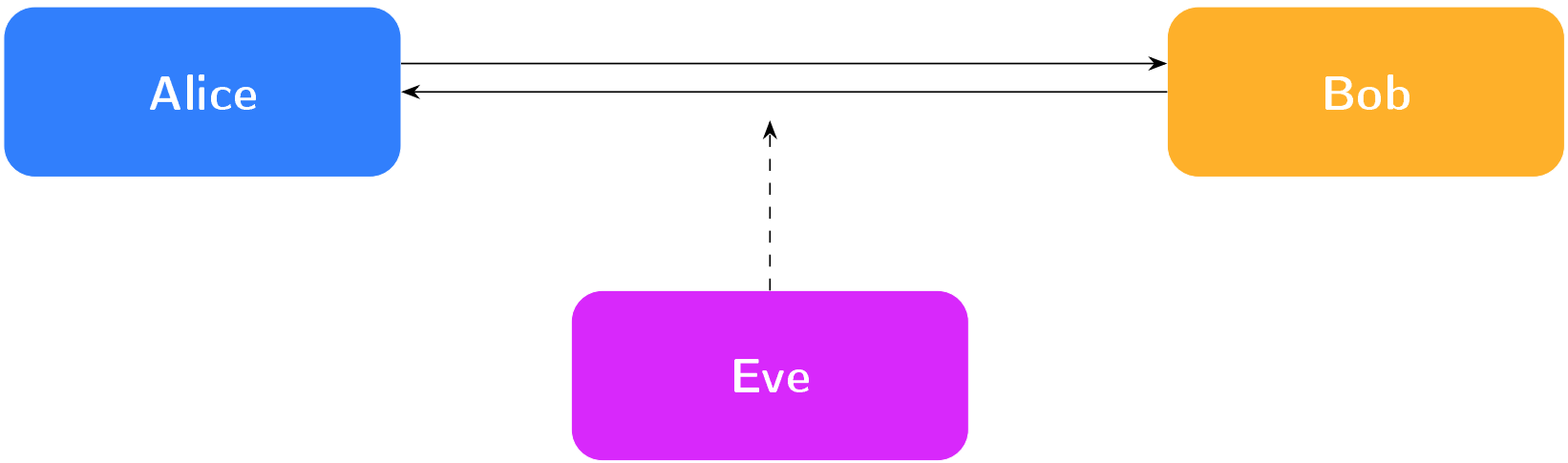
Seconde solution, le PAKE

- PAKE : *Password Authenticated Key Exchange* (Échange de clé authentifié par mot de passe)
- Alice et Bob se prouvent mutuellement qu'ils ont le mot de passe

Clé publique et privée de Alice

Mot de passe : 543718

Mot de passe : 543718



Mot de passe inconnu

Sécurité d'un PAKE

- Pour compromettre un échange, il faut deviner le secret (ici le mot de passe) : nécessité d'un test d'arrêt
- **Test d'arrêt** : condition permettant de discriminer le bon secret parmi tous les secrets possibles
- Spécificité du PAKE : mot de passe de faible entropie
- Deux cadres d'attaques possibles : en ligne et hors-ligne

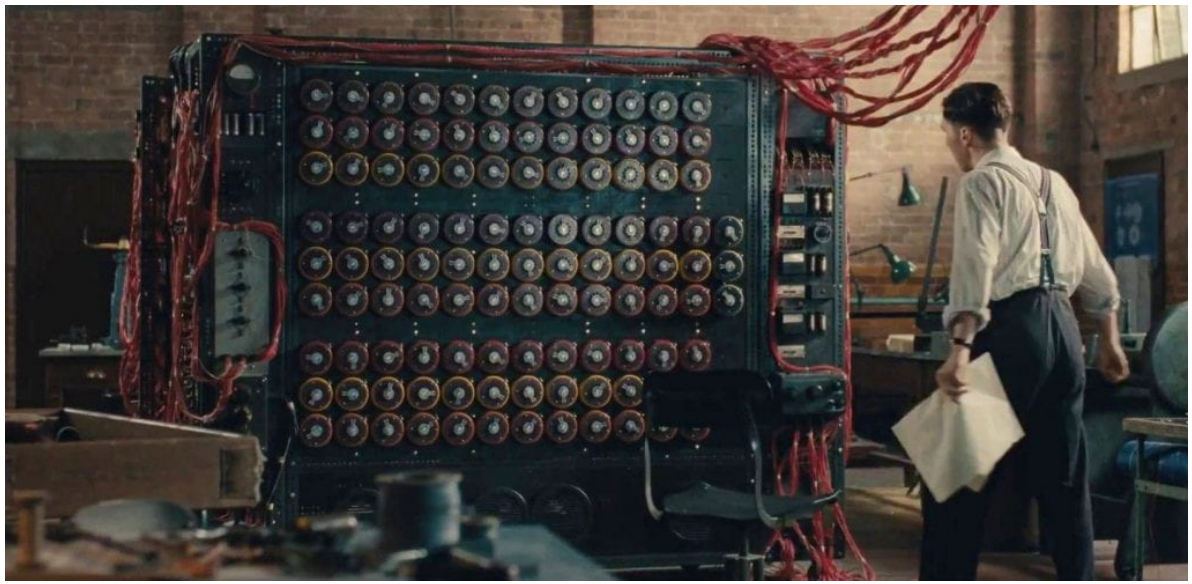


Figure 1 – Le codage Enigma a été cassé grâce à la prédictibilité des messages chiffrés qui fournissaient un test d'arrêt

Attaques en ligne (*online attacks*)

- L'attaquant peut interagir avec les deux parties du protocole
- Attaque de l'homme du milieu (*man-in-the-middle attack*)
- Test d'arrêt très simple
- Solution : protocole lent (limitations matérielles ou algorithmiques)

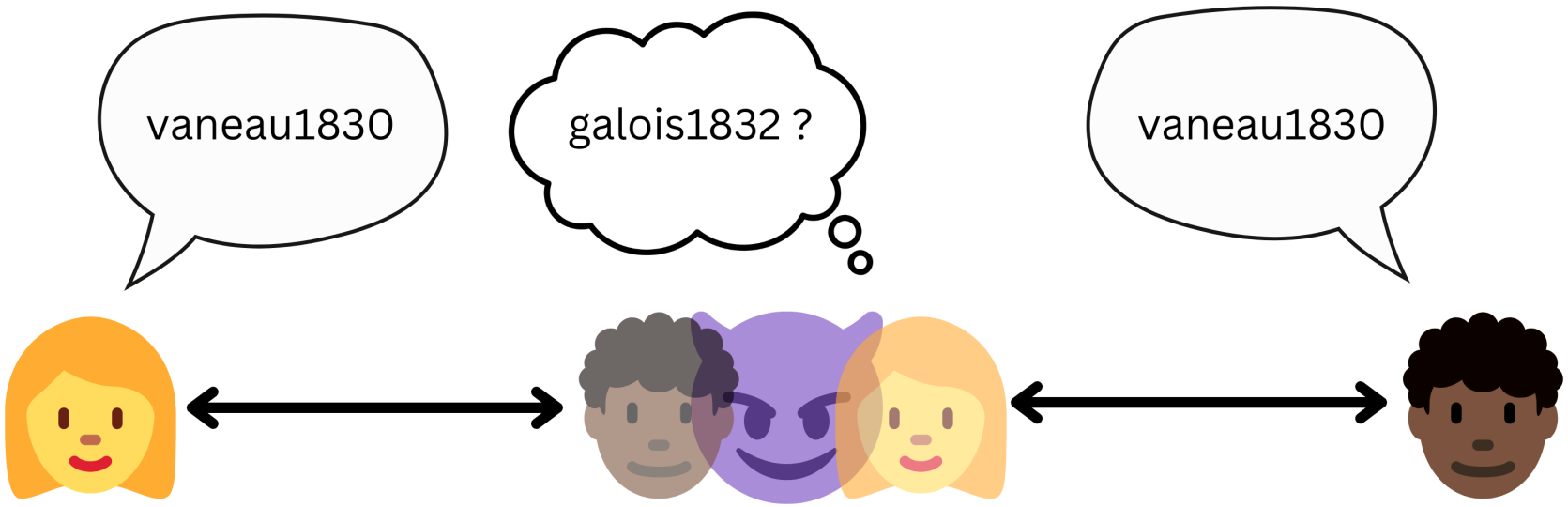


Figure 2 – Schéma d'une attaque de l'homme du milieu

Attaques hors-ligne (*offline attacks*)

- Analyse des échanges entre les deux parties *a posteriori*
- L'attaquant peut tester tous les mots de passe possibles

- Solution : pas de test d'arrêt pour discriminer le bon mot de passe
- Pour un PAKE : messages déchiffrées **indiscernables** d'un tirage aléatoire de bits (taille fixe)

Post-quantique

- Chiffrement asymétrique (et donc les PAKE actuels) menacé par des algorithmes exécutables sur des ordinateurs quantiques (notamment Shor)
- Nécessité de création d'algorithmes post-quantiques :
  - Exécutés sur des ordinateurs classiques.
  - Mais résistants aux attaques par ordinateur classique **et** quantique.

Le protocole CAKE

- **Principe général** : convertir un KEM en PAKE en chiffrant pk et c avec une clé dérivée de pw.

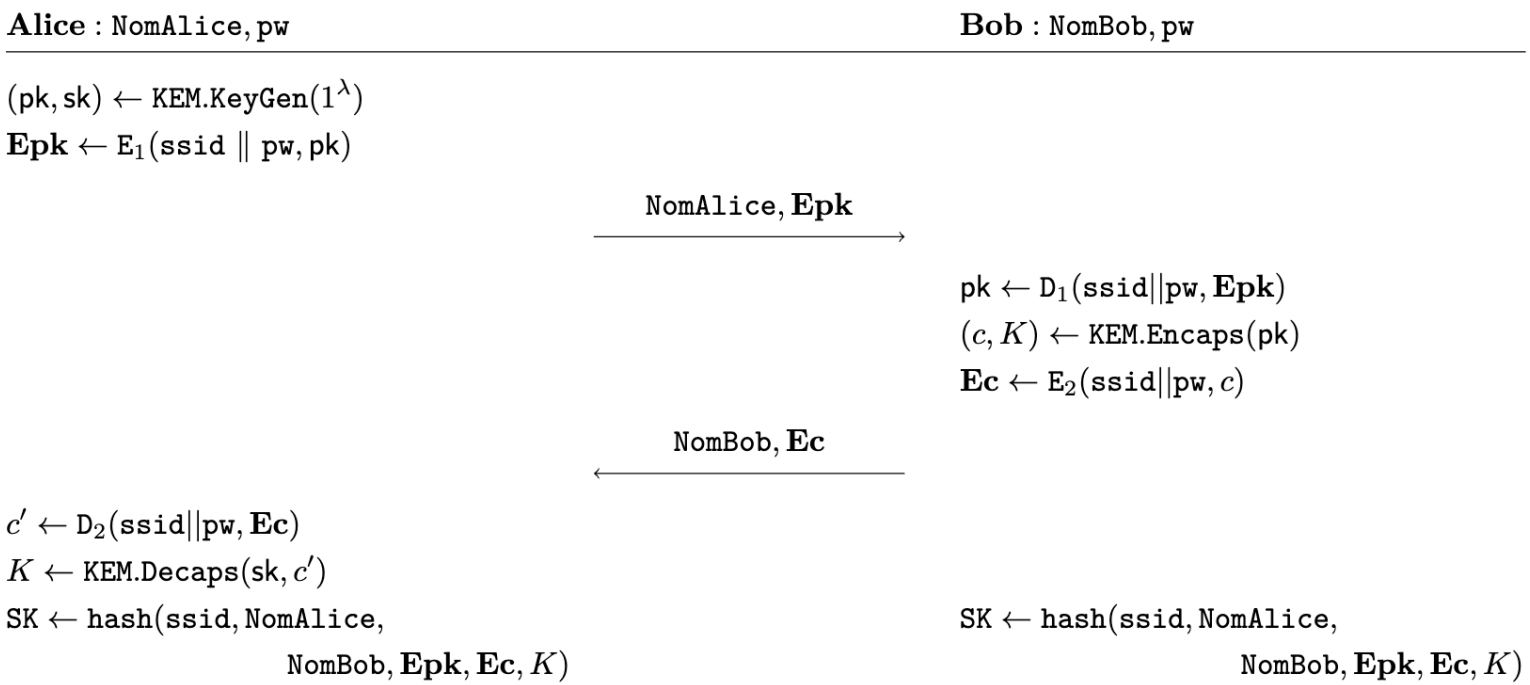


Figure 3 – Fonctionnement du protocole CAKE

Sécurité du protocole CAKE

- Grâce aux propriétés du KEM, une éventuelle attaque fructueuse serait celle de l'**homme du milieu**.
  - Requiert le mot de passe → L'enjeu est de le protéger.

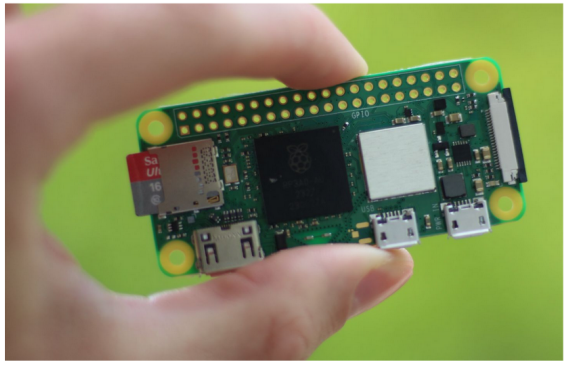
Il faut garantir l'absence de test d'arrêt *offline* sur les données échangées **Epk** et **Ec**.

Conditions pour atteindre cet objectif

- Clé publique indiscernable d'un tirage aléatoire (*fuzziness*).
- *Ciphertext* indiscernable d'un tirage aléatoire (*anonymity*).
- (E_1, D_1) et (E_2, D_2) indiscernables d'une permutation aléatoire sur les espaces de bits associés (*ideal cipher*).

Le KEM `Kyber` vérifie ces deux premières propriétés.

Démonstration



Carte d'identité

Raspberry Pi Zero W 2

Bluetooth

Ordinateur du douanier

Partie 3

Le problème de l'*ideal cipher*

Description du problème

Construire un chiffrement (E, D) de $[\![0, 2^l - 1]\!]$ vérifiant la propriétés essentielle suivante (*ideal cipher*) :

Pour chaque clé k, E_k est indiscernable d'un tirage aléatoire.

En particulier, pour chaque couple clé-message (k_1, m_1), il est statistiquement impossible de trouver un autre couple (k_2, m_2) tel que $E_{k_1}(m_1) = E_{k_2}(m_2)$ sans recourir au dechiffrement D.

Problème

Ici $l \approx 12000$.

Conséquences du modèle de l'*ideal cipher*

Les modes usuels de l'AES ne conviennent pas.

Le chiffrement de Feistel

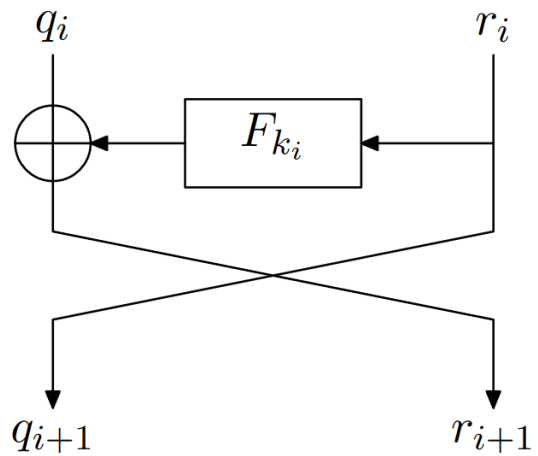


Figure 4 – Le i-ème tour du chiffrement de Feistel

- Division du message en 2 blocs, application de F_{k_i} sur la partie droite, puis échange des 2 blocs.
- Choix : $F_{k_i}(m) = H(k \parallel i \parallel m)$
- Si H est un oracle aléatoire, 14 tours suffisent pour obtenir un *ideal cipher* [Coron *et al.*, 2016].
- Choix d'implémentation : Adaptation de `SHA-512` à ≈ 12000 bits.

Chiffrement

On a donc :
$q_{i+1} = r_i$
$r_{i+1} = q_i \oplus F_{k_i}(r_i)$

Déchiffrement

On remarque que :
$r_i = q_{i+1}$
$q_i = r_{i+1} \oplus F_{k_i}(q_{i+1})$

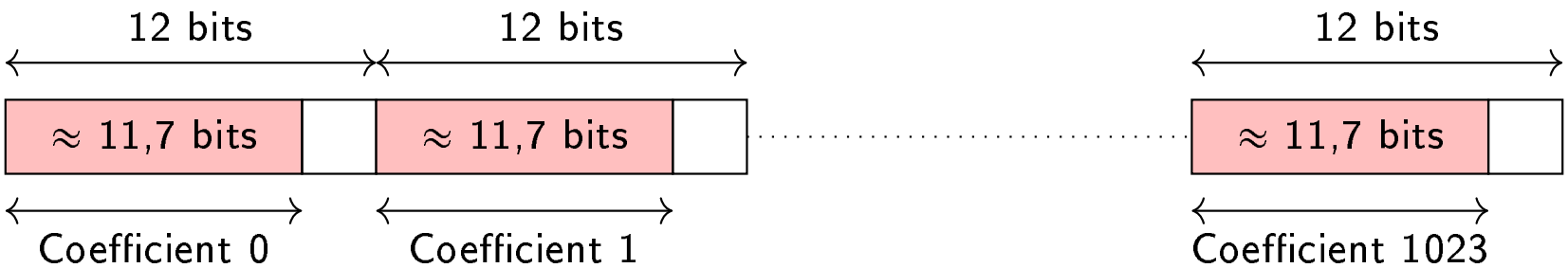
Partie 4

Utilisation de l'*ideal cipher* pour obtenir **Epk** et **Ec**

Encodage de la partie polynomiale de la clé publique

Clé publique ≈ 1024 coefficients $c_0, \ldots, c_{1023}$ entre 0 et $q - 1 = 3328$.

Encodage de la spécification de `Kyber`



Encodage « compact »

$$S = \sum_{i=0}^{1023} c_i \cdot 3329^i, \quad \text{encodé sur 11982 bits}$$

$\rightarrow$ Passage de l'un à l'autre par des opérations de modulo et de *bitshift*.

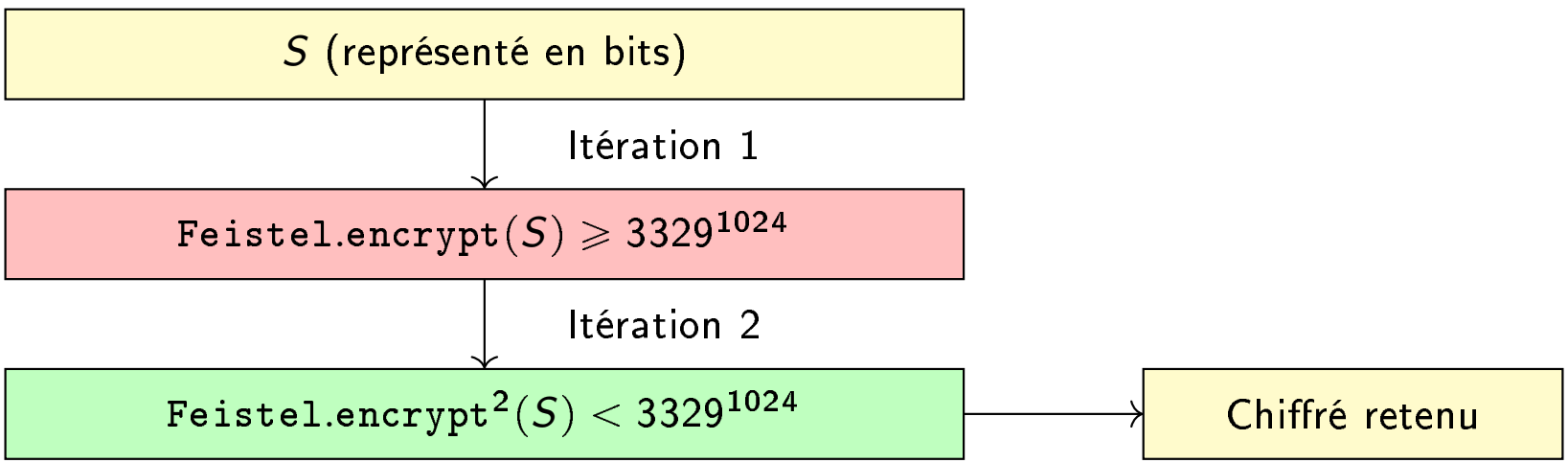
Chiffrement de la clé publique

- L'ensemble des clés publiques : $[\![0, q^{nk}[\![$ avec $q^{nk} = 3329^{1024}$
- $log_2(q^{nk}) \approx 11981,7$ bits
- Chiffrement de Feistel, bijection $E_{\mathtt{pw}} : [\![0, 2^{11982}[\![\rightarrow [\![0, 2^{11982}[\![$ où pw la clé de chiffrement

Problème

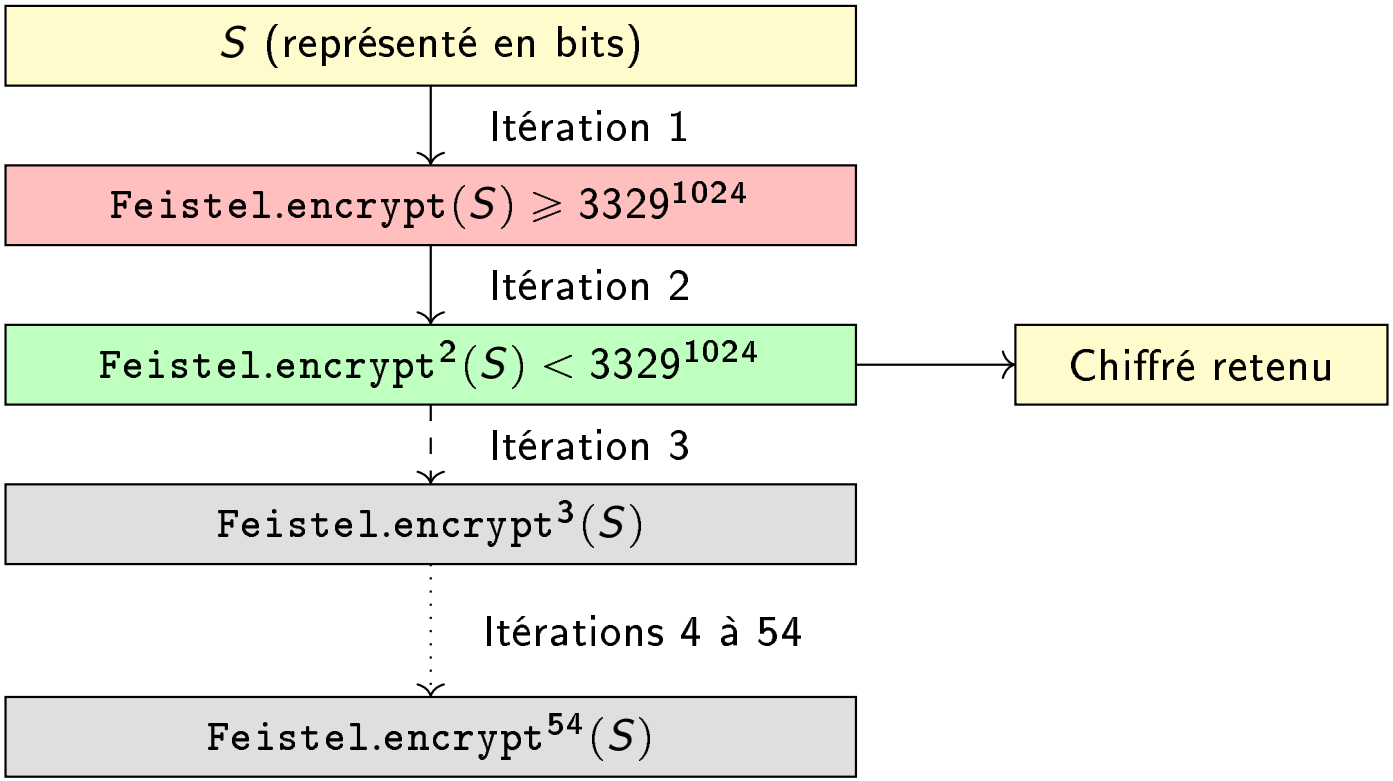
- $E_{\mathtt{pw}}([\![0, q^{nk}[\![) \subsetneq [\![0, 2^{11982}[\![$
- $E_{\mathtt{pw}}([\![0, q^{nk}[\![)$ donne des informations sur pw

- On cherche une nouvelle bijection : $E'_{\mathtt{pw}} : [\![0, q^{nk}[\![\rightarrow [\![0, q^{nk}[\![$



Résistance aux attaques par canaux auxiliaires (dont temporelles)

- Problème : Le temps d'exécution du nouveau chiffrement E'_k dépend de k et S



- Combien faut-il d'itérations ?
  - Probabilité de succès par tour : $p = \frac{3329^{1024}}{2^{11982}} \approx 0,81$
  - Probabilité d'échec après N tours : $(1-p)^N$
  - $(1-p)^N \leqslant 2^{-128} \iff N \geqslant 54$

Prise en charge du chiffré

- $c \in \mathbb{F}_{q^{4n}}[X] \times \mathbb{F}_{q^n}[X] \cong [\![0\ ;\ q-1]\!]^{5n}$
- Pas besoin de compresser comme pour la pk : c est déjà compressé par les implémentations de `Kyber`
- Il suffit seulement de chiffrer avec Feistel

Alice		**Bob**
		$(c, K) \leftarrow \texttt{KEM.Encaps}(\texttt{pk})$
		$\mathbf{Ec} \leftarrow \texttt{E}_\mathbf{2}(\texttt{ssid}\|\|\texttt{pw}, c)$
	$\mathbf{Ec} \longleftarrow$	
$c' \leftarrow \texttt{D}_\mathbf{2}(\texttt{ssid}\|\|\texttt{pw}, \mathbf{Ec})$		

Figure 5 – Transmission du chiffré **Ec**

Partie 5

Hybridation

Hybridation pré et post-quantique : contexte

- **Objectif de l'hybridation** : éviter des régressions de sécurité imputable à la faible maturité des algorithmes post-quantiques.
- Recommandée au moins jusqu'en 2030 par l'ANSSI et le BSI.

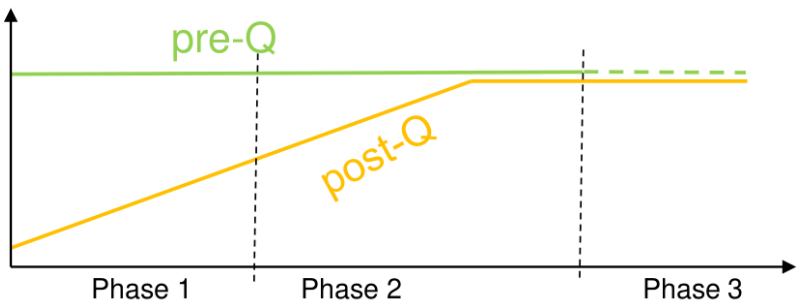


Figure 6 – Chronologie recommandée par l'ANSSI pour la migration post-quantique (schéma par Mélissa Rossi)

- Dans le protocole PAKE, ne concerne que le KEM `Kyber` → Il suffit d'hybrider ce KEM.
  - Utilisation de ElGamal pour la partie pré-quantique.
  - Concaténation puis *sha512* des clés de session.

Création d'un KEM hybride

KEM.KeyGen(1^λ)

1 : $(\mathrm{pk}_{\mathrm{EG}}, \mathrm{sk}_{\mathrm{EG}}) \leftarrow \mathtt{EG.KeyGen}(1^\lambda)$
2 : $(\mathrm{pk}_{\mathrm{Kyber}}, \mathrm{sk}_{\mathrm{Kyber}}) \leftarrow \mathtt{Kyber.KeyGen}(1^\lambda)$
3 : $\mathrm{sk} \leftarrow (\mathrm{sk}_{\mathrm{EG}}, \mathrm{sk}_{\mathrm{Kyber}})$
4 : $\mathrm{pk} \leftarrow \mathrm{pk}_{\mathrm{EG}} \parallel \mathrm{pk}_{\mathrm{Kyber}}$
5 : return $(\mathrm{pk}, \mathrm{sk})$

KEM.Encaps(pk)

1 : $\mathrm{pk}_{\mathrm{EG}} \parallel \mathrm{pk}_{\mathrm{Kyber}} \leftarrow \mathrm{pk}$
2 : $(c_{\mathrm{EG}}, K_{\mathrm{EG}}) \leftarrow \mathtt{EG.Encaps}(\mathrm{pk}_{\mathrm{EG}})$
3 : $(c_{\mathrm{Kyber}}, K_{\mathrm{Kyber}}) \leftarrow \mathtt{Kyber.Encaps}(\mathrm{pk}_{\mathrm{Kyber}})$
4 : $K \leftarrow \mathtt{sha512}(K_{\mathrm{EG}} \parallel K_{\mathrm{Kyber}})$
5 : $c \leftarrow c_{\mathrm{EG}} \parallel c_{\mathrm{Kyber}}$
6 : return (c, K)

KEM.Decaps(sk,c)

1 : $(\mathrm{sk}_{\mathrm{EG}}, \mathrm{pk}_{\mathrm{Kyber}}) \leftarrow \mathrm{sk}$
2 : $c_{\mathrm{EG}} \parallel c_{\mathrm{Kyber}} \leftarrow c$
3 : $K_{\mathrm{EG}} = \mathtt{EG.Decaps}(\mathrm{sk}_{\mathrm{EG}}, c_{\mathrm{EG}})$
4 : $K_{\mathrm{Kyber}} = \mathtt{Kyber.Decaps}(\mathrm{sk}_{\mathrm{Kyber}}, c_{\mathrm{Kyber}})$
5 : return $K \leftarrow \mathtt{sha512}(K_{\mathrm{EG}} \parallel K_{\mathrm{Kyber}})$

- Pas encore présent dans nos implémentations.
- Sécurité augmentée si défaillance du chiffrement.
- Risque si défaillance de l'*anonymity* ou de la *fuzziness*.

Partie 6

Implémentation

Prototype en Python

Alice

```
alice = Alice(
 int(0).to_bytes(),
 b'183754')
alice.generate_keypair()
```

alice.encrypted_public_key, alice.name →

Bob

```
bob = Bob(
 int(0).to_bytes(),
 b'183754')
```

```
bob.generate_symmetric_key(
 alice.encrypted_public_key,
 alice.name)
```

← bob.encrypted_ciphertext, bob.name

```
alice.decrypt_ciphertext(
 bob.encrypted_ciphertext,
 bob.name)
alice.session_key
```

```
bob.session_key
```

Implémentation finale

- Objectifs :
  - Démonstrateur qui se rapproche le plus possible d'une solution réelle
  - Résoudre le plus de difficultés techniques
  - Créer une API simple d'utilisation pour le développeur
- Choix du langage de programmation : **C**
  - Performances
  - Portabilité ($\neq$ assembleur ou VHDL)
  - Ubiquité dans les systèmes embarqués
- Choix de la bibliothèque `Kyber` : `PQClean`

Publication

- Notre travail est en ligne sur nos dépôts GitHub sous licence libre
- `https://github.com/pq-pake/pqpake-python`
- `https://github.com/pq-pake/pqpake-c`

Partie 7

Conclusion

Conclusion

Nos contributions

- Résolution des problèmes théoriques
  - Une proposition d'*ideal cipher* : le chiffrement de Feistel
  - Génération de hashs de taille arbitraire
  - L'encodage compact de la clé publique
  - Protection contre les attaques par canaux auxiliaires
- Hybridation
- Implémentation en C et en Python
- Démonstrateur physique

Merci pour votre écoute !


Coron, J.-S., Holenstein, T., Künzler, R., Patarin, J., Seurin, Y. et Tessaro, S. (2016).
How to Build an Ideal Cipher : The Indifferentiability of the Feistel Construction.
Journal of Cryptology, 29(1):61–114.