

Secure Migration to Post-Quantum Cryptography on Smartcard

Aurélien Greuet

IDEMIA Secure Transactions - Crypto & Security

PHISIC - May 21, 2025



Context 1

Context

IDEMIA Secure Transactions

- › Payment → bank card, mobile
- › Telecoms → SIM/eSIM
- › Connectivity → connected cars, NFC car keys

Crypto & Security Team

Development + security evaluation of crypto libraries

- › For smartcard / secure element (≃ smartcard chip without card)
- › Secure against side-channel and faults attacks

PHISIC 2022

Talk on Post-Quantum Crypto deployment on smartcard

- › Focus on basic primitives
- › Main message: tricky because of **performance** and **security** constraints

→ this talk: achievements on basic primitives + next step = protocol implementations

Smartcard Constraints & Post-Quantum Cryptography

2

Smartcard Constraints for Cryptography



4 GHz + multi-core + AVX

> 8 GB

> 2.5 MB/s

Constant time implem



CPU

RAM

Data
Transmission

Security
Needs

100 MHz, single core

≤ 64 KB

≤ 100 kB/s

Masking (side-channel attacks)
Redundancy (fault attacks)

Post-Quantum Cryptography: Overview

Solution to resist quantum computers

- › New public key crypto \neq RSA, Elliptic Curves

Post-Quantum Cryptography: Overview

Solution to resist quantum computers

- › New public key crypto \neq RSA, Elliptic Curves

Key Exchange Mechanism (KEM)

Key pair sk, pk

Encaps Generate + encrypt a shared secret: $(ss, ct) = \text{Encaps}(pk)$

Decaps Decrypt ciphertext to recover shared secret: $ss = \text{Decaps}_{sk}(ct)$

Length versus ECC: $pk +, ct + \rightarrow$ execution $+$ / communication $+$ / RAM $+$

Post-Quantum Cryptography: Overview

Solution to resist quantum computers

› New public key crypto \neq RSA, Elliptic Curves

Key Exchange Mechanism (KEM)

Key pair sk, pk

Encaps Generate + encrypt a shared secret: $(ss, ct) = \text{Encaps}(pk)$

Decaps Decrypt ciphertext to recover shared secret: $ss = \text{Decaps}_{sk}(ct)$

Length versus ECC: $pk +, ct + \rightarrow$ execution $+$ / communication $+$ / RAM $+$

Digital Signature Scheme

Key pair sk, pk

Sign Given message m , signer computes $\sigma = \text{Sign}_{sk}(m)$

Verif Anyone can check that σ is valid: $\text{Verif}_{pk}(m, \sigma)$

Length versus ECC: $pk ++, \sigma +++ \rightarrow$ execution $++$ / communication $+++$ / RAM $+++$

Achievements

3

Quantum Safe Proof of Concepts

Secure Implementation of Standards

- › Implementation + practical security evaluation of current NIST standards
- › Substantial work on **masking** + **RAM optimization**

Quantum Safe Proof of Concepts

Secure Implementation of Standards

- › Implementation + practical security evaluation of current NIST standards
- › Substantial work on **masking** + **RAM optimization**

Proof of Concepts

Goal Adapt existing protocols

Constraints PQC algos are new and may evolve (new attacks, standard updates)

- › Avoid any security regression
 - **hybrid crypto** = combination classical + PQ
- › Ability to update post-quantum crypto
 - **crypto-agility**

Quantum Safe Proof of Concepts

Secure Implementation of Standards

- › Implementation + practical security evaluation of current NIST standards
- › Substantial work on **masking** + **RAM optimization**

Proof of Concepts

Goal Adapt existing protocols

Constraints PQC algos are new and may evolve (new attacks, standard updates)

- › Avoid any security regression
 - **hybrid crypto** = combination classical + PQ
- › Ability to update post-quantum crypto
 - **crypto-agility**

Use cases

- › Bank card transaction
- › 5G authentication
- › Passport reading
- › eSIM profile download
- › eSIM crypto-agility

Hybrid Authentication

4

Authentication with Classical Cryptography

Authentication with Signature



Bank card

sk_C, pk_C

$C_B^{pk_C}$ = authent material for pk_C



Terminal

Public key pk_B

Authentication with Classical Cryptography

Authentication with Signature



Bank card

sk_C, pk_C

$C_B^{pk_C}$ = authent material for pk_C

$pk_C, C_B^{pk_C}$



Terminal

Public key pk_B

Authentication with Classical Cryptography

Authentication with Signature



Bank card

sk_C, pk_C

$C_B^{pk_C}$ = authent material for pk_C

$pk_C, C_B^{pk_C}$



Terminal

Public key pk_B

Terminal Verify that pk_C is genuine with pk_B and $C_B^{pk_C} \rightarrow pk_C$ now trusted

Authentication with Classical Cryptography

Authentication with Signature



Bank card

sk_C, pk_C

$C_B^{pk_C}$ = authent material for pk_C

$pk_C, C_B^{pk_C}$

$rand_T$



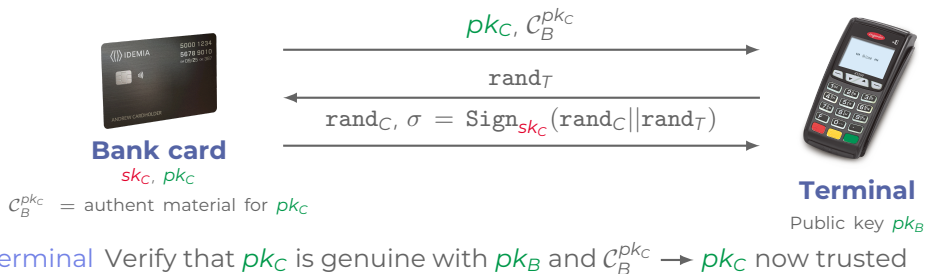
Terminal

Public key pk_B

Terminal Verify that pk_C is genuine with pk_B and $C_B^{pk_C} \rightarrow pk_C$ now trusted

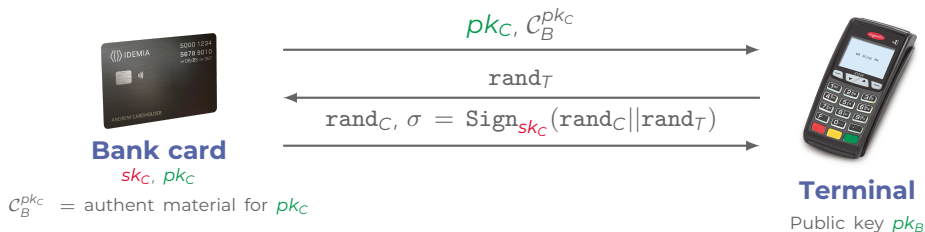
Authentication with Classical Cryptography

Authentication with Signature



Authentication with Classical Cryptography

Authentication with Signature



Terminal Verify that pk_C is genuine with pk_B and $C_B^{pk_C} \rightarrow pk_C$ now trusted

Terminal $\text{Verif}_{pk_C}(rand_C || rand_T, \sigma) \rightarrow$ card signature valid \rightarrow card authenticated!

Authentication with Post-Quantum Cryptography

Natural Idea

Same protocol with post-quantum signature

→ Execution time ++

→ Communication time +++

Authentication with Post-Quantum Cryptography

Natural Idea

Same protocol with post-quantum signature

→ Execution time ++

→ Communication time +++

Smarter Idea

Authentication scheme with KEM?

→ Execution time +

→ Communication time +

Post-Quantum Authentication with KEM



Bank card

sk_C^* , pk_C^*

$C_B^{pk_C^*}$ = authentic material for pk_C^*



Terminal

Public key pk_B^*

Post-Quantum Authentication with KEM



Bank card

sk_C^* , pk_C^*

$c_B^{pk_C^*}$ = authent material for pk_C^*

pk_C^* , $c_B^{pk_C^*}$



Terminal

Public key pk_B^*

Post-Quantum Authentication with KEM



Bank card

sk_C^* , pk_C^*

$c_B^{pk_C^*}$ = authentic material for pk_C^*

pk_C^* , $c_B^{pk_C^*}$



Terminal

Public key pk_B^*

Terminal Verify with pk_B^* and $c_B^{pk_C^*}$ that pk_C^* is genuine \rightarrow pk_C^* now trusted

Post-Quantum Authentication with KEM



Bank card

sk_C^* , pk_C^*

$c_B^{pk_C^*}$ = authentic material for pk_C^*

pk_C^* , $c_B^{pk_C^*}$

ct



Terminal

Public key pk_B^*

Terminal Verify with pk_B^* and $c_B^{pk_C^*}$ that pk_C^* is genuine \rightarrow pk_C^* now trusted

Terminal $(ss, ct) \leftarrow \text{Encaps}(pk_C^*)$ and sends ct

Post-Quantum Authentication with KEM



Bank card

sk_C^* , pk_C^*

$c_B^{pk_C^*}$ = authentic material for pk_C^*

pk_C^* , $c_B^{pk_C^*}$

ct



Terminal

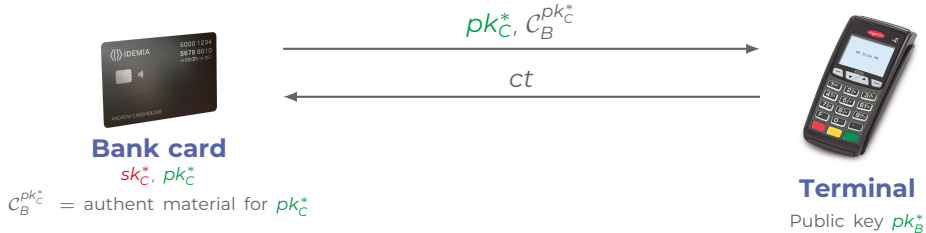
Public key pk_B^*

Terminal Verify with pk_B^* and $c_B^{pk_C^*}$ that pk_C^* is genuine \rightarrow pk_C^* now trusted

Terminal $(ss, ct) \leftarrow \text{Encaps}(pk_C^*)$ and sends ct

Card $ss = \text{Decaps}_{sk_C^*}(ct)$

Post-Quantum Authentication with KEM



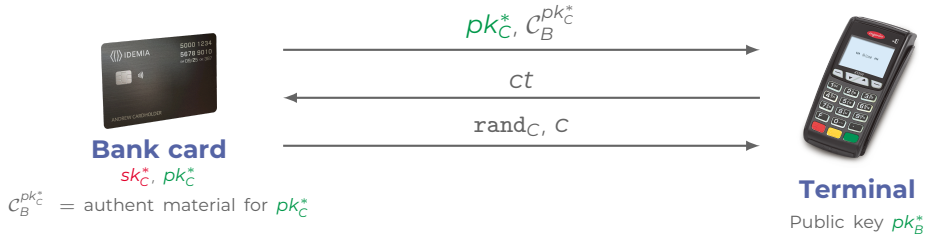
Terminal Verify with pk_B^* and $c_B^{pk_C^*}$ that pk_C^* is genuine \rightarrow pk_C^* now trusted

Terminal $(ss, ct) \leftarrow \text{Encaps}(pk_C^*)$ and sends ct

Card $ss = \text{Decaps}_{sk_C^*}(ct)$

Card Generates random rand_C , derive symmetric key $K = \text{KeyDeriv}(ss || \text{rand}_C)$

Post-Quantum Authentication with KEM



Terminal Verify with pk_B^* and $c_B^{pk_C^*}$ that pk_C^* is genuine \rightarrow pk_C^* now trusted

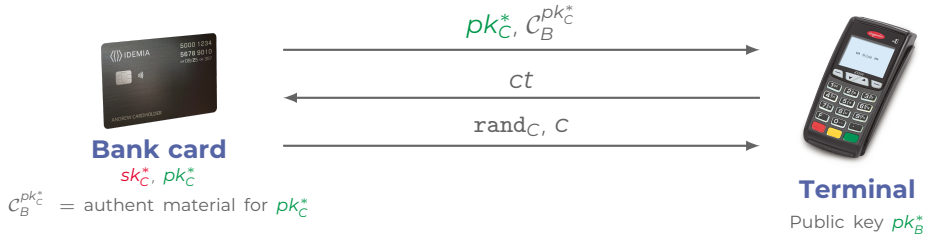
Terminal $(ss, ct) \leftarrow \text{Encaps}(pk_C^*)$ and sends ct

Card $ss = \text{Decaps}_{sk_C^*}(ct)$

Card Generates random $rand_C$, derive symmetric key $K = \text{KeyDeriv}(ss || rand_C)$

Card Computes $mac = \text{SymEnc}_K(ct)$, sends $rand_C$ and mac

Post-Quantum Authentication with KEM



Terminal Verify with pk_B^* and $c_B^{pk_C^*}$ that pk_C^* is genuine \rightarrow pk_C^* now trusted

Terminal $(ss, ct) \leftarrow \text{Encaps}(pk_C^*)$ and sends ct

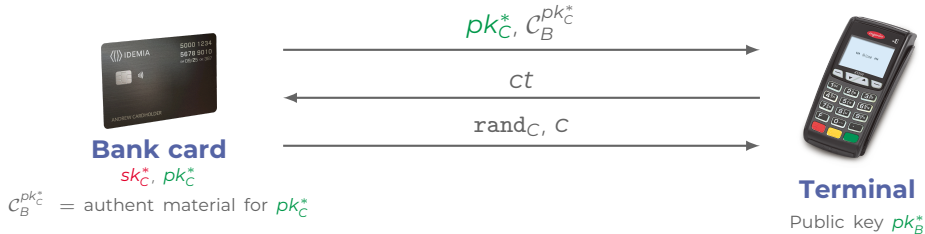
Card $ss = \text{Decaps}_{sk_C^*}(ct)$

Card Generates random $rand_C$, derive symmetric key $K = \text{KeyDeriv}(ss || rand_C)$

Card Computes $mac = \text{SymEnc}_K(ct)$, sends $rand_C$ and mac

Terminal Computes $K = \text{KeyDeriv}(ss || rand_C)$

Post-Quantum Authentication with KEM



Terminal Verify with pk_B^* and $c_B^{pk_C^*}$ that pk_C^* is genuine \rightarrow pk_C^* now trusted

Terminal $(ss, ct) \leftarrow \text{Encaps}(pk_C^*)$ and sends ct

Card $ss = \text{Decaps}_{sk_C^*}(ct)$

Card Generates random $rand_C$, derive symmetric key $K = \text{KeyDeriv}(ss || rand_C)$

Card Computes $mac = \text{SymEnc}_K(ct)$, sends $rand_C$ and mac

Terminal Computes $K = \text{KeyDeriv}(ss || rand_C)$

Terminal If $ct = \text{SymDec}_K(mac) \rightarrow$ card authenticated

Hybrid Authentication (with KEM)



Bank card

$sk_C, pk_C / sk_C^*, pk_C^*$

$C_B^{pk_C} / C_B^{pk_C^*} = \text{authent material for } pk_C / pk_C^*$



Terminal

Public keys pk_B / pk_B^*

Hybrid Authentication (with KEM)



Bank card

$sk_C, pk_C / sk_C^*, pk_C^*$

$C_B^{pk_C} / C_B^{pk_C^*} = \text{authent material for } pk_C / pk_C^*$

Classical authent: RSA / ECC, with $pk_C, C_B^{pk_C}$



Terminal

Public keys pk_B / pk_B^*

Hybrid Authentication (with KEM)



Bank card

$sk_C, pk_C / sk_C^*, pk_C^*$

$C_B^{pk_C} / C_B^{pk_C^*} = \text{authent material for } pk_C / pk_C^*$

Classical authent: RSA / ECC, with $pk_C, C_B^{pk_C}$

PQ authent: KEM with $pk_C^*, C_B^{pk_C^*}$



Terminal

Public keys pk_B / pk_B^*

Hybrid Authentication (with KEM)



Bank card

$sk_C, pk_C / sk_C^*, pk_C^*$

$C_B^{pk_C} / C_B^{pk_C^*} = \text{authent material for } pk_C / pk_C^*$

Classical authent: RSA / ECC, with $pk_C, C_B^{pk_C}$

PQ authent: KEM with $pk_C^*, C_B^{pk_C^*}$



Terminal

Public keys pk_B / pk_B^*

Terminal Authentication OK \iff Classical authent & PQ authent both OK

Hybrid Authentication (with KEM)



Bank card

$sk_C, pk_C / sk_C^*, pk_C^*$

$C_B^{pk_C} / C_B^{pk_C^*} = \text{authent material for } pk_C / pk_C^*$

Classical authent: RSA / ECC, with $pk_C, C_B^{pk_C}$

PQ authent: KEM with $pk_C^*, C_B^{pk_C^*}$



Terminal

Public keys pk_B / pk_B^*

Terminal Authentication OK \iff Classical authent & PQ authent both OK

Overhead RSA + ML-KEM versus RSA for a full transaction

Setup: ARM Cortex-M3 @100 MHz – NIST security level 1

[Bettale et al. – CARDIS'22]

PQ Certif Algo	Cert len	Card computation	Communication	Total Timing	Storage
Falcon	++	380ms	5.3 s	$\times 3.4$	$\times 1.39$
ML-DSA	++++	380ms	9.0 s	$\times 5.6$	$\times 1.55$

→ Most of time spent in communication!

Conclusion

5

Conclusion

Secure Post-Quantum Crypto on smartcard

- › Hybrid + crypto-agility not straightforward but feasible
- › Crypto execution time +
- › Communication time +++

Suggestions for improvement

- › More RAM, PQ accelerators → impact: +
- › Communication speed → impact: +++

Conclusion

Secure Post-Quantum Crypto on smartcard

- › Hybrid + crypto-agility not straightforward but feasible
- › Crypto execution time +
- › Communication time +++

Suggestions for improvement

- › More RAM, PQ accelerators → impact: +
- › Communication speed → impact: +++

› Future work

- › PoCs on wider ecosystems
 - Hyperform project
 - 3 years french program
 - Hybrid crypto + crypto-agility
 - Secure elements, PC & cloud



CRYPTONEXT
SECURITY



IDEMIA
SECURE
TRANSACTIONS



IRISA



SYNACKTIV

Thank you!

aurelien.greuet@idemia.com



www.idemia.com

Crypto-Agility

Problematic

- › Secure update of crypto algorithms = trusted source + integrity
- › For the worst case: quantum computer + security breach in PQ schemes

Crypto-Agility

Problematic

- › Secure update of crypto algorithms = trusted source + integrity
- › For the worst case: quantum computer + security breach in (non hashed-based) PQ

Crypto-Agility

Problematic

- › Secure update of crypto algorithms = trusted source + integrity
 - › For the worst case: quantum computer + security breach in (non hashed-based) PQ
- Rely on (stateful) hash-based signatures

Crypto-Agility

Problematic

- › Secure update of crypto algorithms = trusted source + integrity
 - › For the worst case: quantum computer + security breach in (non hashed-based) PQ
- Rely on (stateful) hash-based signatures

Crypto-Agility



Update server

Hash-based keypair sk , pk



SIM

Server hash-based public key pk

Crypto-Agility

Problematic

- › Secure update of crypto algorithms = trusted source + integrity
 - › For the worst case: quantum computer + security breach in (non hashed-based) PQ
- Rely on (stateful) hash-based signatures

Crypto-Agility



Update server

Hash-based keypair sk , pk



SIM

Server hash-based public key pk

Server Send $patch + \sigma = \text{HashBasedSign}_{sk}(patch)$

Crypto-Agility

Problematic

- › Secure update of crypto algorithms = trusted source + integrity
 - › For the worst case: quantum computer + security breach in (non hashed-based) PQ
- Rely on (stateful) hash-based signatures

Crypto-Agility



Update server

Hash-based keypair sk, pk



SIM

Server hash-based public key pk

Server Send $patch + \sigma = \text{HashBasedSign}_{sk}(patch)$

SIM If $\text{Verif}_{pk}(patch, \sigma)$, apply patch → update code + certs + generate keys