Smartcard and Post-Quantum Crypto

Aurélien Greuet – aurelien.greuet@idemia.com
IDEMIA - Crypto & Security Labs

IDEMIA

June 7-9, 2021

Context
1

Context

IDEMIA

Merge between Morpho and Oberthur Technologies

- Identity Management → 3B ID docs, 5M biometric terminals
- Payment → 800M payment products (2020)
- Telecoms → 900M SIM cards (2020)

Crypto & Security Labs

Development + practical evaluations of crypto libraries

- For smartcard / secure element (≃ smartcard chip without card)
- Secure against side-channel and faults attacks
- For the following products:
  - Electronic ID cards, electronic passports
  - Chip bank cards, mobile payment
  - SIM cards, eUICCs

Problematic: why and how to deploy PQC on today's smartcards

Smartcard Constraints, Impact for PQC Deployment

2

Computer vs Smartcard

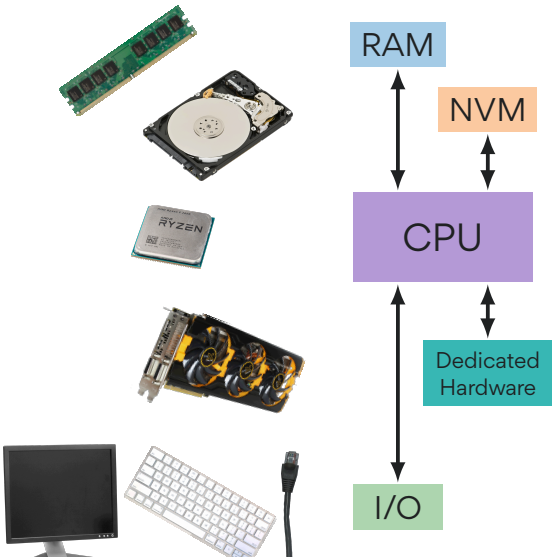

Image credits: Slide 15

Computer vs Smartcard

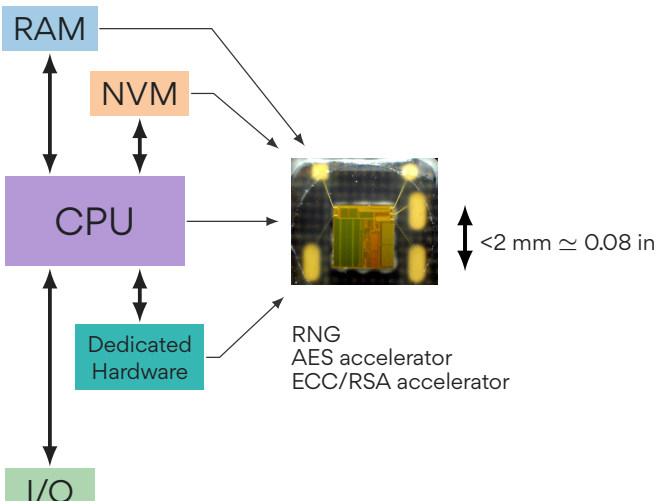

Image credits: Slide 15

Computer vs Smartcard

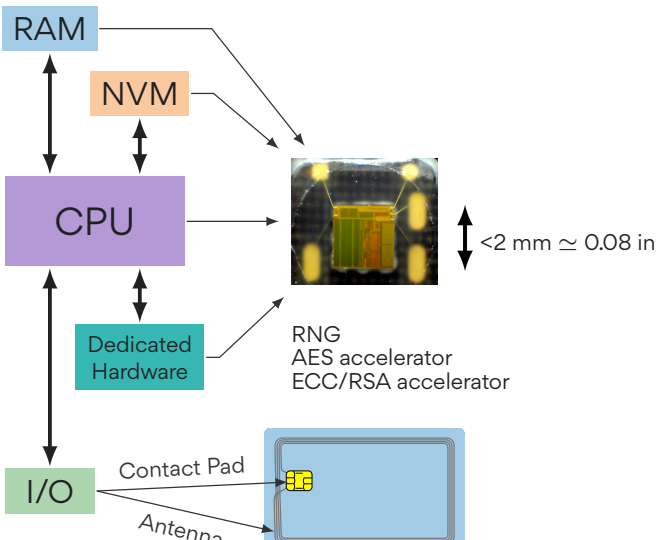

Image credits: Slide 15

Smartcard Specificities: Performances

Low Computing Capacity

	$400 PC	High-end Smartcard	
CPU	64-bit, 4 cores @4 GHz	32-bit, 1 core @100 MHz	→ > 40× slower
RAM	8 GB	48 kB	→ 170 000× less

	STM32F4	High-end Smartcard	
CPU	Cortex-M4 @168 MHz	Cortex-M3 @100 MHz	→ > 1.68× slower
RAM	192 kB	48 kB	→ 4× less

Communication rates

Pretty slow: < 100 kB/s

Performance Constraints: Examples

- Contactless banking transaction: < 300 ms
- Key Generation performed in factory: < 3-4 second

Satisfying Constraints

Dedicated Hardware

RNG 32-bit random in $50 - 100$ cycles, parallel execution

AES 1 block encryption: several hundred cycles, parallel execution

ECC/RSA 2048-bit modular mult: several thousand cycles, parallel execution, $> 10\times$ faster than software implem

→ but no dedicated hardware for Post-Quantum Crypto yet

Off-card computation

Example: signature of a several MB scanned document (Qualified eIDAS signature)

1. Hash all the document except the last block on terminal or computer
2. Send partial hash state + document last block to smartcard
3. Finalize hashing and compute signature on smartcard

→ not possible if computation of *Hash* (`rand || msg`)

Security Constraints

Into the wild

An issued smartcard is in uncontrolled hostile environment:

- Attacker = owner
- No monitoring, no remote action
- Hard to deploy security flaw patches
- Sensitive to specific side-channel/fault attacks

Security Certification

Hardware and software can be certified, e.g. Common Criteria Certification

→ ensures practical security level

→ long process: 6-18 months, **need to be anticipated**

Cost of Security

Protection against 1st order Side Channel Attacks

- SCA: Simple Power Analysis, Differential/Correlation Power Analysis
- Countermeasure = masking → at best, execution time $\times 2$, RAM $\times 2$
- Practically, much worse
  → Example: masked Dilithium execution time $\times 5.6$ with optimized modulus
  [Migliore et al. *Masking Dilithium*, ACNS 2019]

Protection against single fault attacks

Countermeasure = redundancy → execution time up to $\times 2$, small RAM overhead

Other attacks

- Safe error attacks → additional checks
- Template/machine learning attacks → shuffling

→ **Security against physical attacks is expensive**

Impact on PQ Crypto Deployment

Performance & security constraints eliminate some finalists:

McEliece: too much RAM, too slow

- RAM: OS McEliece > 70 kB
- Time: KG > 1 224 Mcycles

[Roth et al. *Classic McEliece Implementation with Low Memory Footprint*, CARDIS 2020]

Impact on PQ Crypto Deployment

Performance & security constraints eliminate some finalists:

McEliece: too much RAM, too slow

- RAM: 

- Time: KG > 1 224 Mcycles + communication to output 260 kB pubkey → > 14 s

  [Roth et al. *Classic McEliece Implementation with Low Memory Footprint*, CARDIS 2020]

Impact on PQ Crypto Deployment

Performance & security constraints eliminate some finalists:

McEliece: too much RAM, too slow

- RAM: OS | McEliece > 70 kB
- Time: KG > 1 224 Mcycles + communication to output 260 kB pubkey → > 14 s

  [Roth et al. *Classic McEliece Implementation with Low Memory Footprint*, CARDIS 2020]

Falcon: too much RAM, too slow with countermeasures

- RAM: OS | Falcon > 25 kB
- Time: KG > 171 Mcycles

  [Pornin. *New Efficient, Constant-Time Implementations of Falcon*, ePrint 2019]

Impact on PQ Crypto Deployment

Performance & security constraints eliminate some finalists:

McEliece: too much RAM, too slow

- RAM: OS | McEliece > 70 kB
- Time: KG > 1 224 Mcycles + communication to output 260 kB pubkey → > 14 s

  [Roth et al. *Classic McEliece Implementation with Low Memory Footprint*, CARDIS 2020]

Falcon: too much RAM, too slow with countermeasures

- RAM: OS | Falcon > 25 kB | + masking
- Time: KG > 171 Mcycles + countermeasures overhead → > 3.5 s

  [Pornin. *New Efficient, Constant-Time Implementations of Falcon*, ePrint 2019]

Impact on PQ Crypto Deployment

Performance & security constraints eliminate some finalists:

McEliece: too much RAM, too slow

- RAM: OS | McEliece > 70 kB
- Time: KG > 1 224 Mcycles + communication to output 260 kB pubkey → > 14 s

  [Roth et al. *Classic McEliece Implementation with Low Memory Footprint*, CARDIS 2020]

Falcon: too much RAM, too slow with countermeasures

- RAM: OS | Falcon > 25 kB | + masking
- Time: KG > 171 Mcycles + countermeasures overhead → > 3.5 s

  [Pornin. *New Efficient, Constant-Time Implementations of Falcon*, ePrint 2019]

Rainbow-Classic: too slow with countermeasures

- Time: KG > 115 Mcycles

  [Moya Riera. *Performance Analysis of Rainbow on ARM Cortex-M4*, Bachelor Thesis]

Impact on PQ Crypto Deployment

Performance & security constraints eliminate some finalists:

McEliece: too much RAM, too slow

- RAM: OS | McEliece > 70 kB
- Time: KG > 1 224 Mcycles + communication to output 260 kB pubkey → > 14 s

  [Roth et al. *Classic McEliece Implementation with Low Memory Footprint*, CARDIS 2020]

Falcon: too much RAM, too slow with countermeasures

- RAM: OS | Falcon > 25 kB | + masking
- Time: KG > 171 Mcycles + countermeasures overhead → > 3.5 s

  [Pornin. *New Efficient, Constant-Time Implementations of Falcon*, ePrint 2019]

Rainbow-Classic: too slow with countermeasures

- Time: KG > 115 Mcycles + comm. for 150 kB pubkey + countermeasures → > 4 s

  [Moya Riera. *Performance Analysis of Rainbow on ARM Cortex-M4*, Bachelor Thesis]

Impact on PQ Crypto Deployment

Performance & security constraints eliminate some finalists:

McEliece: too much RAM, too slow

- RAM: OS | McEliece > 70 kB
- Time: KG > 1 224 Mcycles + communication to output 260 kB pubkey → > 14 s

  [Roth et al. *Classic McEliece Implementation with Low Memory Footprint*, CARDIS 2020]

Falcon: too much RAM, too slow with countermeasures

- RAM: OS | Falcon > 25 kB | + masking
- Time: KG > 171 Mcycles + countermeasures overhead → > 3.5 s

  [Pornin. *New Efficient, Constant-Time Implementations of Falcon*, ePrint 2019]

Rainbow-Classic: too slow with countermeasures

- Time: KG > 115 Mcycles + comm. for 150 kB pubkey + countermeasures → > 4 s

  [Moya Riera. *Performance Analysis of Rainbow on ARM Cortex-M4*, Bachelor Thesis]

→ **only lattice-based finalists are practical on current smartcard**

Some Ideas for PQC Deployment on Smartcard

3

Some Ideas for PQC Deployment on Smartcard 1/2

Memory issues

- Standardization of at least 1 KEM and 1 signature fitting in smartcard
- Consider low memory devices (< 50 kB) in addition to Cortex-M4

Dedicated Hardware

- Keccak co-processor: secure, running in parallel
  → Many schemes spend 40-70% on hashing
  [Kannwischer et al. *pqm4: Testing and Benchmarking NIST PQC on ARM Cortex-M4*, ePrint 2019]

Off-card hash for qualified signature

- Avoid computations of *Hash* (`rand` || `msg`)?
  → *Hash* (`msg` || $\cdots$) instead would allow off-card hash computation

Some Ideas for PQC Deployment on Smartcard 2/2

Specifications and Parameters

- ”NTT schemes” with randoms not necessarily in NTT domain?
  → would slow down software implementations but allows to:
  - Use generic polynomial multiplication hardware
    [Roy, Basso. *High-speed Instruction-set Co-processor for Lattice-based KEM: Saber in Hardware*, TCHES 2020]
  - Re-use RSA accelerator for polynomial multiplication
    [Albrecht et al. *Implementing RLWE-based Schemes Using an RSA Co-Processor*, TCHES 2019]
    [Bos et al. *Post-Quantum Cryptography with Contemporary Co-Processors*, ePrint 2020]

- Investigate trade-offs on parameters
  → Example: masked Dilithium with power of 2 modulus much faster
  [Migliore et al. *Masking Dilithium*, ACNS 2019]

Thank you for your attention!

aurelien.greuet@idemia.com

IDEMIA

Join us on

www.idemia.com

Image Credits

From Wikimedia Commons

`Swissbit_2GB_PC2-5300U-555_remix_crop_transparent.png`, CC BY-SA 3.0

`Laptop-hard-drive-exposed_remix_transparent.png`, CC BY-SA 3.0

`AMD_Ryzen_5_2600_(39851733273)_remix_transparent.png`, CC0

`Sapphire Radeon R9 290X-front oblique PNr°0437_remix_transparent.png`, CC BY 3.0

`Computer_monitor_remix_transparent.png` by Zzubnik, Public domain

`Apple_Magic_Keyboard_-_US_remix_transparent.png`, CC BY 4.0

`Ethernet_cable_remix_crop_rotate_transparent.png`, CC BY-SA 3.0

`Sim_Chip.jpg` by Janke, CC BY-SA 4.0

`Dual_smart_card.svg`, CC BY-SA 4.0