h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES

in collaboration with

Bundesamt
für Sicherheit in der
Informationstechnik

Implementation of a QR PACE protocol

Project System Development (M.Sc.)
WS 2022/23

Ilyes Ben Dlala | Hans Geißner | Jean Kanellakopoulos

Agenda

- Motivation and Setting
- Rectified PACE Protocol
- Contributions
  - Programming language migration
  - Fix of communication bug
  - Provision of benchmark capabilities
- Live Demonstration
- Outlook

Motivation and Setting

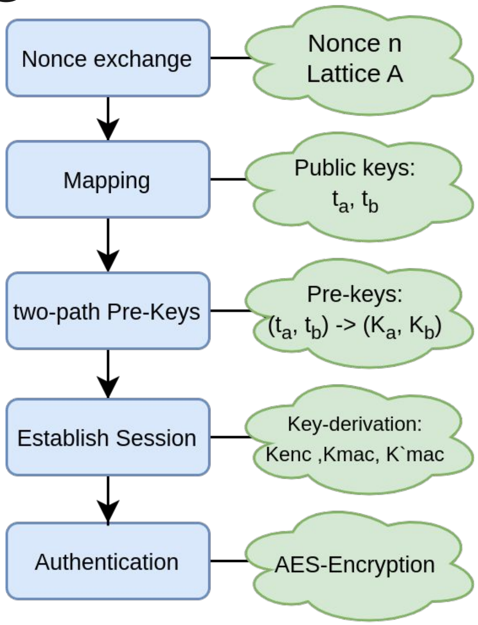
- Establishment of quantum computers (QCs) will pose challenges to cryptography and eIDs
  - Widespread cryptographic primitives become obsolete (e.g. DH, RSA)
  - Threat to currently used eID protocols (such as PACE and EAC)
- Implementation of PACE based on Kyber
  - Considered safe against future QC attacks
  - Implementation of necessary KEMs already exist (e.g. PQ-Crystals[1], PQM4[2])
- Development on STM32 Nucleo Board[3] (with ARM Cortex-M4 processor)
  - Sufficiently similar to the computation capabilities of an eID
  - Card terminal simulated by ordinary computer

1 https://github.com/pq-crystals/kyber
2 https://github.com/mupq/pqm4
3 STM32L4R5ZIT6

Rectified PACE Protocol: Kyber-Ding PACE

- Generate nonce
- Encrypt nonce with PIN-derived key
- Send KEM public base A with encrypted nonce
- Recover nonce using PIN-derived key
- Compute ephemeral KEM PKs
- Mask PK with hashed nonce
- Exchange public keys
- Unmask PK_A with decrypted nonce
- Encapsulate pre-keys using ephemeral KEM PKs
- Decapsulate pre-keys K_a K_b
- Derive master key K
- Create session keys (K_{enc}, K_{MAC}, etc.)
- Create session tokens T_A T_B
- Exchange encrypted messages

► eID ► Terminal ► Both



Contributions

Programming language migration: C++ → C

Why?

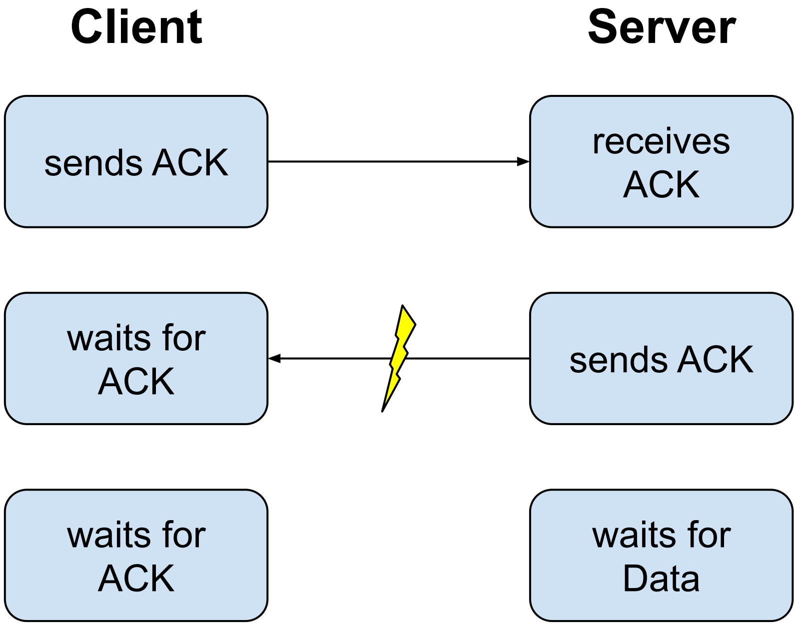
- Performance: C provides low-level control and a fast, efficient way to perform computations required in cryptographic algorithms.
- Portability: Cryptographic algorithms must be implemented consistently across different platforms to ensure their security. C is widely supported and known for its portability, making it a better choice for cryptographic implementations.
- Standardization: C is an established standard and widely adopted, making it easier to compare, audit, and review cryptographic implementations.
- Security: C has a straightforward programming model with fewer ways to write incorrect code, reducing the risk of security vulnerabilities. It also provides low-level control over the hardware, which is important for implementing secure cryptographic algorithms.

How?

- Replaced C++ Classes with Structs
- Replaced C++ and third party libraries with C ones
- Replaced automatic C++ memory management with explicit memory allocations (const char*, malloc() ...)
- Code refactor to make use of C features (less casting), i.a ...

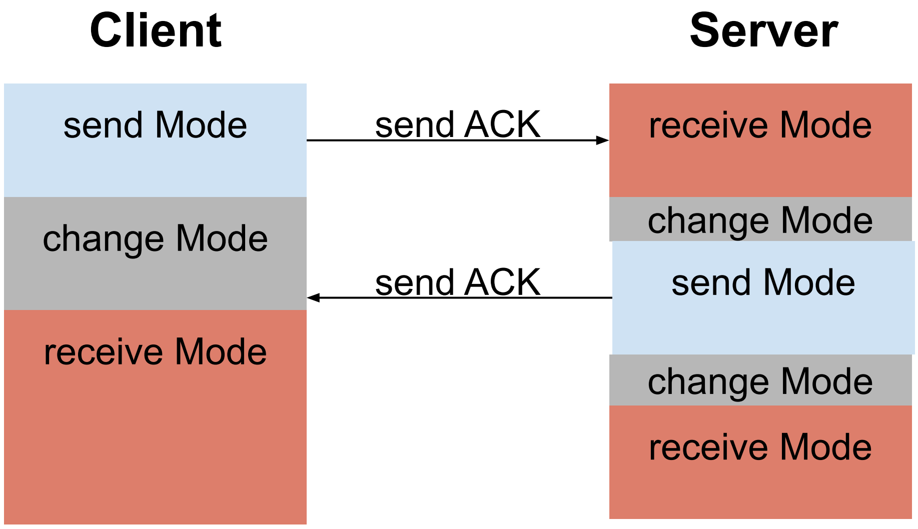
Fix of Communication Bug

- Communication from Server to Client fails
- Server and Client go into a Deadlock where they are waiting on each other



Slow Hardware as an Error Source

- One possible problem is that the client needs to actively be in receive mode
- Since the server is faster than the client, this may cause issues
- This can be solved by proper interrupt handling



Live Demo

Protocol Output of Terminal

Waiting for data
reading 16 bytes
Done
[+] Connected!
Sending 16 bytes
Done
Waiting for data
reading 32 bytes
Done
[+] Nonce Generation
[+] Nonce:
f19032b357f8c90aad…
Waiting for data
reading 32 bytes
Done
[+] Generation of Asymmetric Key
Sending 1184 bytes
Done
Waiting for data
reading 1184 bytes
Done

[+] Public Seed:
f0498e9387a1884de0f4…
[+] Public Key a:
8e875962557f7851…
[+] Public Key b:
7f997c7805acadb6…

[+] PreKey Generation
Sending 1088 bytes
Done
Waiting for data
reading 1088 bytes
Done
[+] shared secret a:
56fa138a8f3c30dc793d8b35263…
[+]shared secret b:
887eabd8953d098b876e01643a…
[+] cipher a:
57e70289bfc0f4b71b…
[+] cipher b:
074a2bcb059ee8700…
[+] Token Verification
Sending 16 bytes
Done

Waiting for data
reading 16 bytes
Done
Verification successful.

[+] SessionKey:
f571056572ef6aea3c94…
[+] EncryptionKey:
62e6471ec6597b3c…
[+] MacKey:
5f0bed4be342e093…
[+] Mac'Key:
c88d0d9f391d2a31…
[+] SessionKey
[+] Key:
62e6471ec6597b3c…
[+] SID:
8e875962557f785118…
Waiting for data
reading 4 bytes
Done

Pre-Communication lasted 3669 ms
Waiting for data
reading 96 bytes
Done
48656c6c6f2c206…

Hello, my name is Alice and I study Computer Science at the Hochschule Darmstadt.
Sending 96 bytes
Done

Benchmark Capabilities

- Time benchmarks for the µ-controller to evaluate the implementation
- Processor on board has various count/time capabilities
  - 64-bit register TIM5 selected by us for benchmarks
    - CLK = 120 MHz
    - Pre-Scaler for TIM5 = 60,000

    → One tick ~ 0.5 ms
    - Timing precision could be further increased
- Setup and usage are described in the repository's wiki[1]
- No benchmarks have been collected so far

1 https://code.fbi.h-da.de/aw/prj/athenepqc/mpse-eid-implementation/-/wikis/Experimental-setup

Future Work

- Run benchmarks with different configurations on the board
- Deploy Kyber implementation from the *pqm4* library
- Take a look at *libopencm3* as a possible HAL replacement

Thank you for your attention!