PV181 Laboratory of security and applied cryptography

Seminar 10: Post Quantum Cryptography in Practice

Łukasz Chmielewski
chmiel@fi.muni.cz

CROCS
Centre for Research on Cryptography and Security

Goals

- Why do we need post-quantum crypto?
  - Why is it not called quantum crypto?
- Context + Main Schemes + Efficiency
- How to use it in Python / Java
- Homework

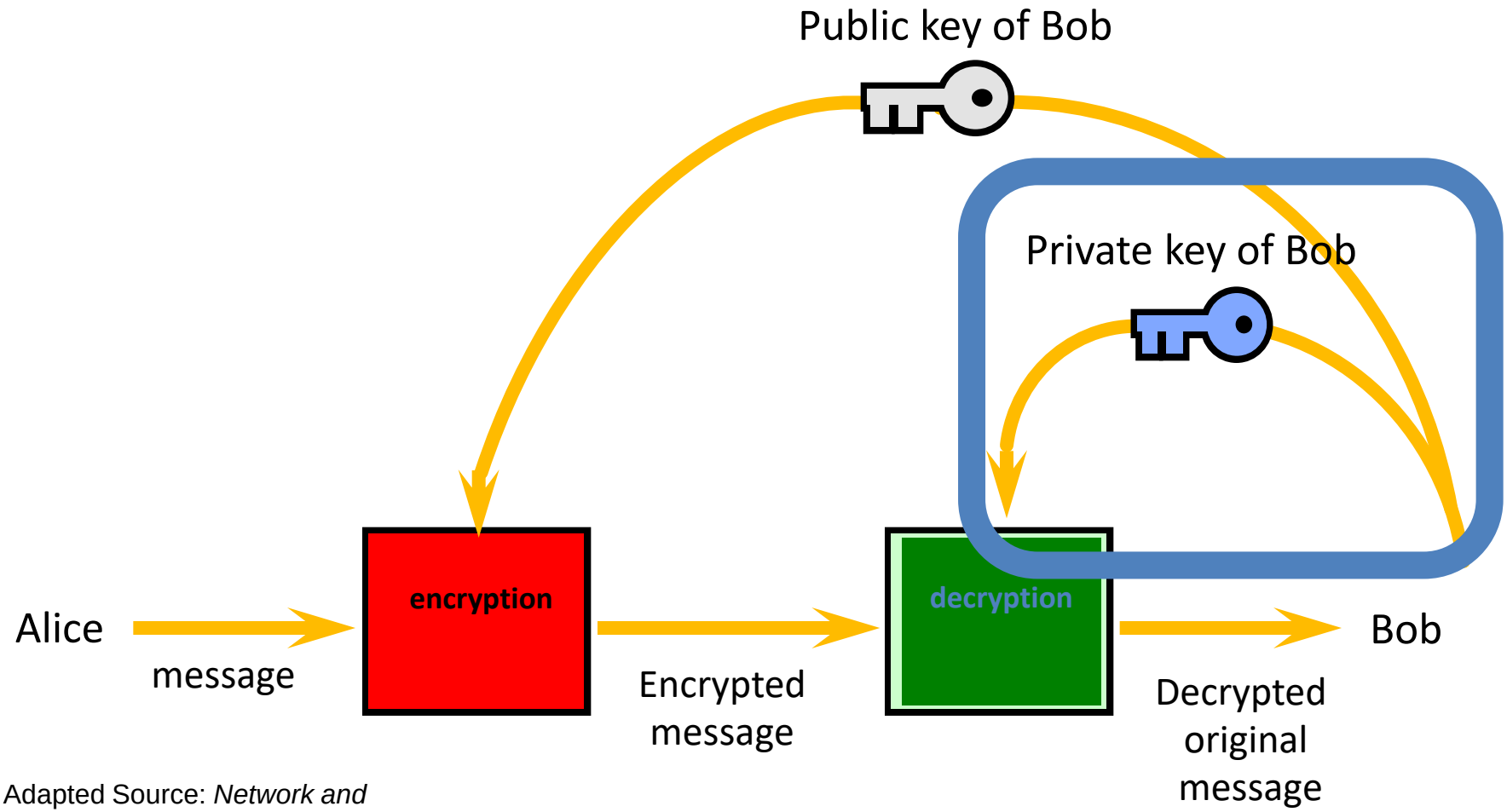
- In the first part I expect some discussions
- The intro is inspired by the work of Douglas Stebila, an Associate Professor of cryptography from the University of Waterloo, Canada.

Outline

- Part 1: Discussion
  - Introduction (also classic crypto reminder)
  - Why do we need post-quantum crypto?
    - Why is it not called quantum crypto?
    - Security vs. Efficiency
- Part 2: Schemes
- Part 3: Libraries experimentation
- Homework

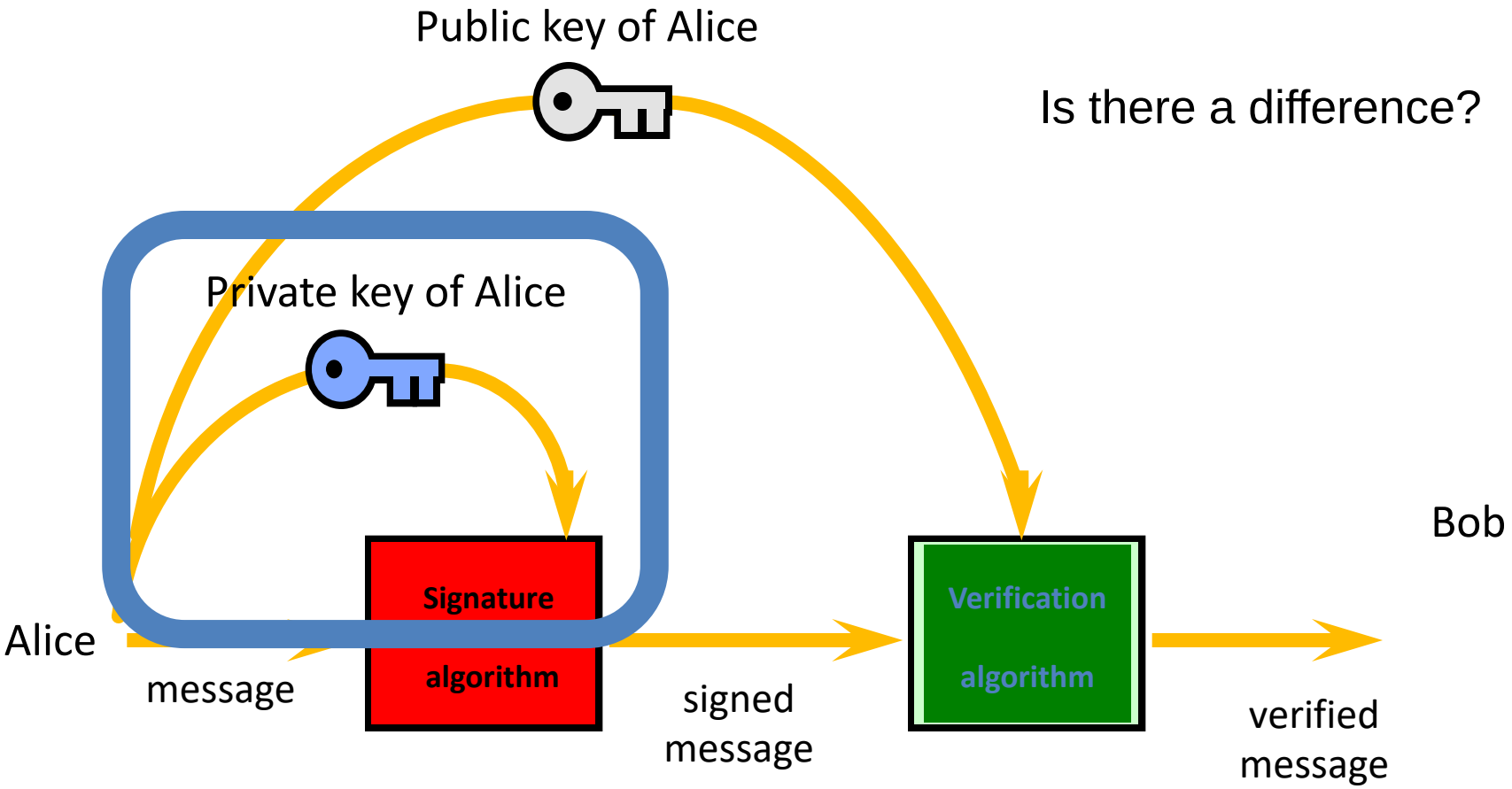
Intro

Recall: Asymmetric cryptosystem



Adapted Source: *Network and Internetwork Security* (Stallings)

Recall: Digital signature scheme



Source: *Network and Internetwork Security* (Stallings)

Classic Crypto

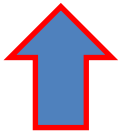
- We experimented with classical crypto
  - Also called “Pre-Post-Quantum” ☺
- Symmetric Crypto: AES, DES, ASCON, SHA-2
- Asymmetric Crypto: RSA, ECC, ECDSA, DSA
- Both kinds of primitives are constructed using varying degrees of mathematical structure.
- The structure should imply that an adversary trying to break the primitive needs to solve some hard mathematical problem.

RSA: reminder

1. Secret primes p, q: $n = p \cdot q$
2. Public exponent e:
$$\gcd(e, (p-1)) = \gcd(e, (q-1)) = 1$$
3. Private exponent d: $d \cdot e \equiv 1 \; mod \; \varphi(n)$

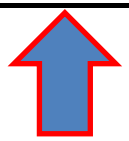
Encryption (public n, e): $E(m) = m^e \; mod \; n = c$

Decryption (private n, d): $D(c) = c^d \; mod \; n = m$



ECC: reminder

- Generating key pair
  - Select a random integer **d** from [1,n − 1]
  - Compute **P** = [d]G = d*G;



- Private key: **d**
- Public key: **P**,
  - also: **G**, and curve details are also public

- Signature Algorithm: ECC

RSA vs. ECC: what is easier?

- Which key is bigger: ECC or RSA?
- Why?
- Let $N = p \cdot q$ for random p, q such that $\log p \approx \log q$.
- It takes at most $2^{(\log N)/2} \approx 2^{\log p}$ division attempts
  - ECC is similar
- But there exist much faster attacks, such as the (general number field sieve, GNFS) that takes the following number of operations:

$$\exp\left(\left(\sqrt[3]{\frac{64}{9}} + o(1)\right)(\ln N)^{\frac{1}{3}}(\ln \ln N)^{\frac{2}{3}}\right)$$

- Can it be done even faster?

PQC Intro

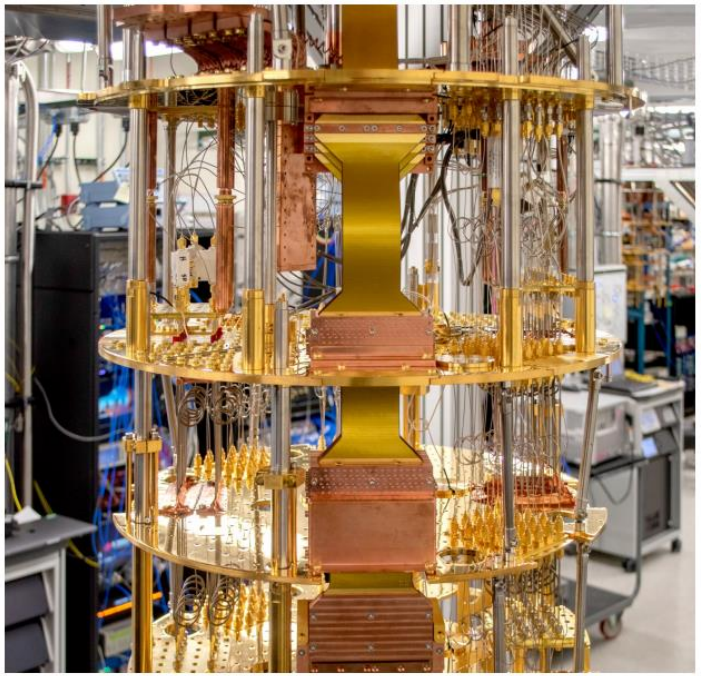
Task 0

- Why “I know Kung-fu”?
- One single interesting point from the preparation
- Please be fast ☺

- You were asked to look at:
  – https://en.wikipedia.org/wiki/Post-quantum_cryptography

Quantum computing

- Processing using quantum mechanics
- Processing information in superposition can dramatically speed up some computations
- But not everything (quantum computers aren't magic)
- Troubles building above prototype.
- Quantum Cryptography?
  - Exists but we do not discuss that today.



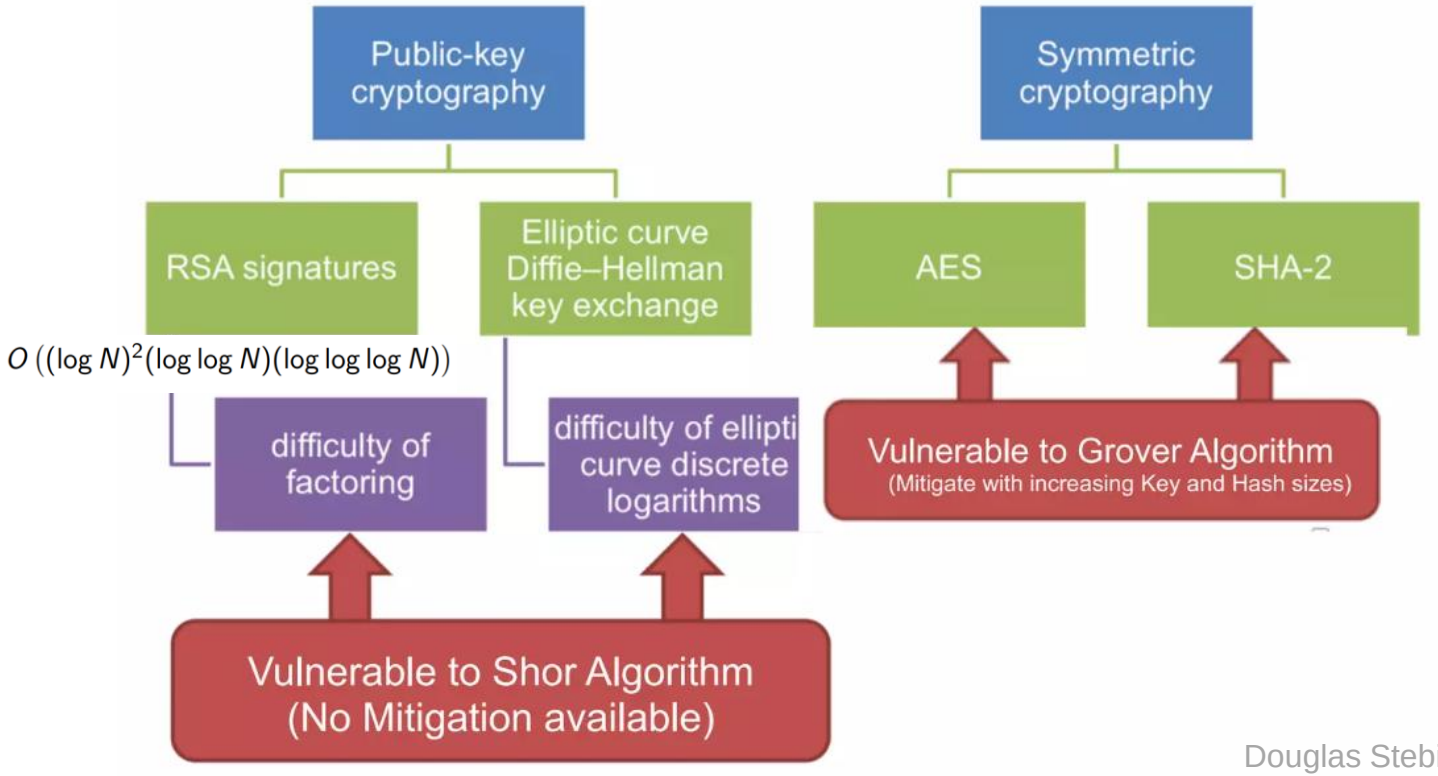
Douglas Stebila

Quantum supremacy?

- Quantum supremacy or "quantum advantage" is the potential ability of quantum computing devices to solve problems that classical computers practically cannot.
- Superpolynomial speedup over the best known or possible classical algorithm.
- Was quantum supremacy achived?
- Yes, in a way (2022-)
- Wiki: In March of 2024, D-Wave Systems reported on an experiment using a quantum annealing based processor that out-performed classical methods including tensor networks and neural networks. They argued that no known classical approach could yield the same results as the quantum simulation within a reasonable time-frame and claimed quantum supremacy. The task performed was the simulation of the non-equilibrium dynamics of a magnetic spin system quenched through a quantum phase transition.

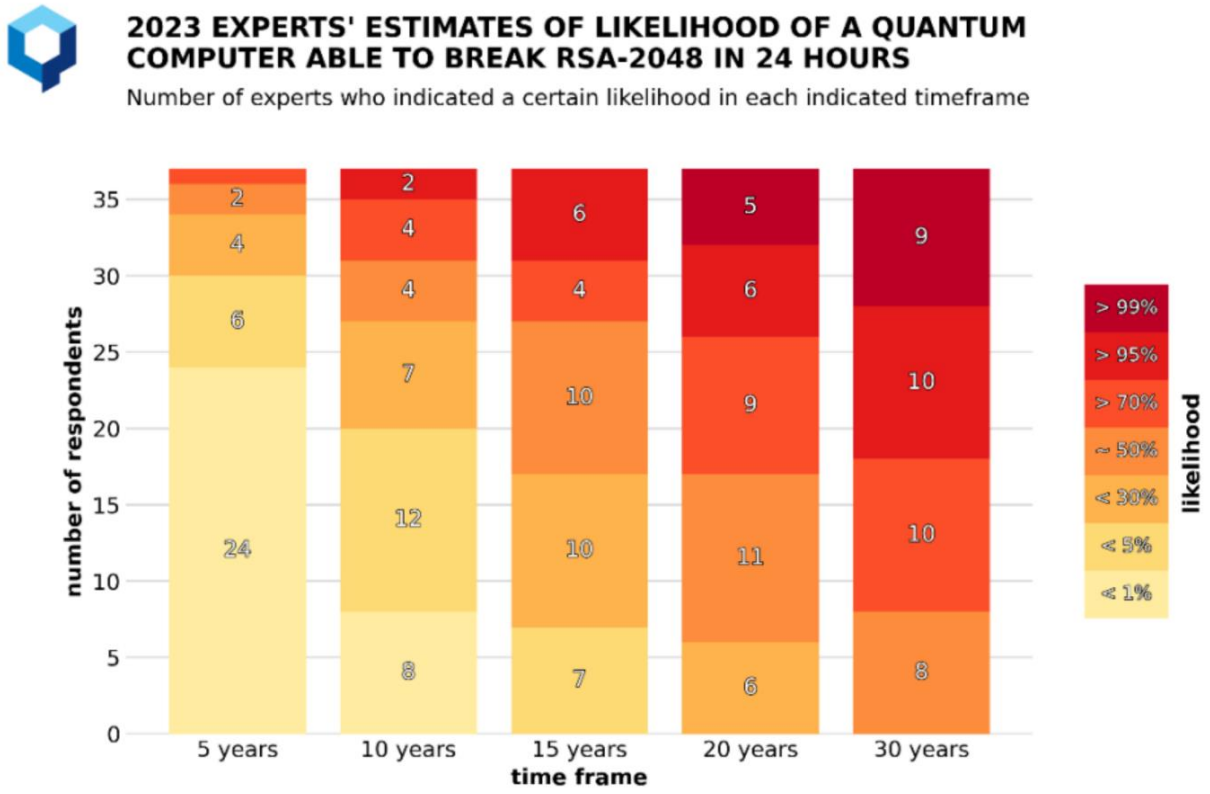
Quantum supremacy?

Quantum Known Vulnerabilities
(one slide summary)



Douglas Stebila

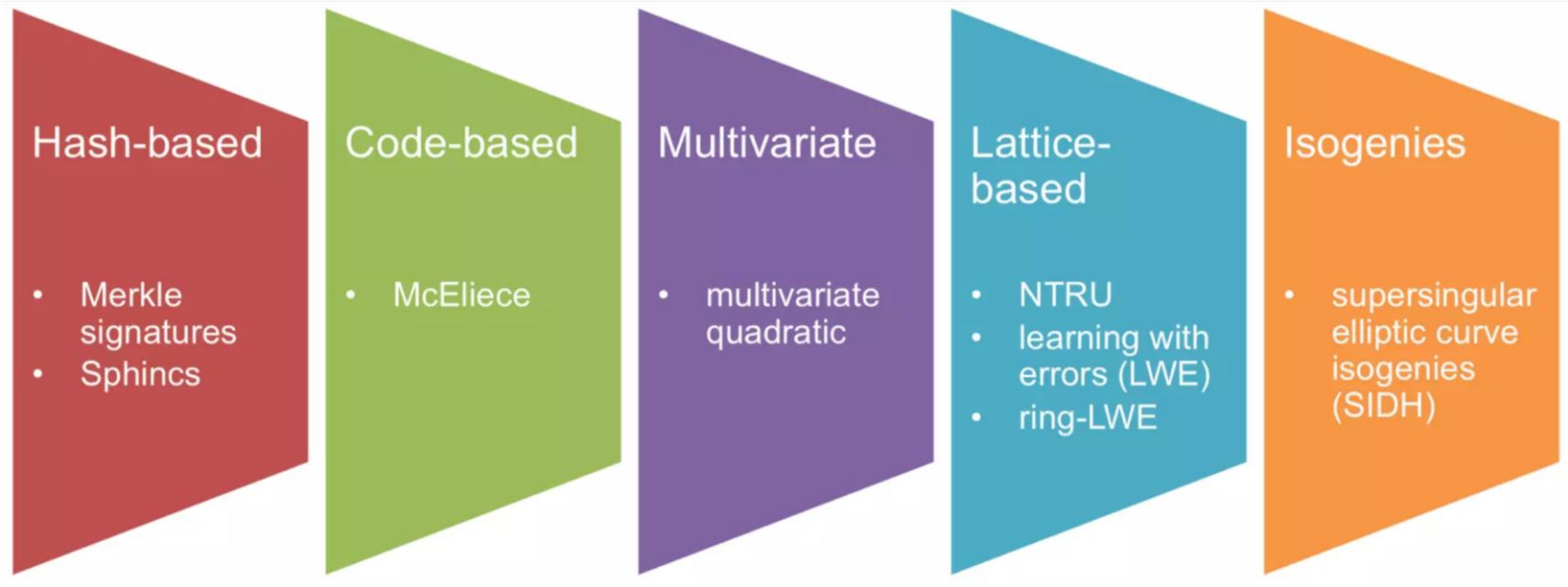
When will a cryptographically relevant quantum computer be built?



https://globalriskinstitute.org/publication/2023-quantum-threat-timeline-report/

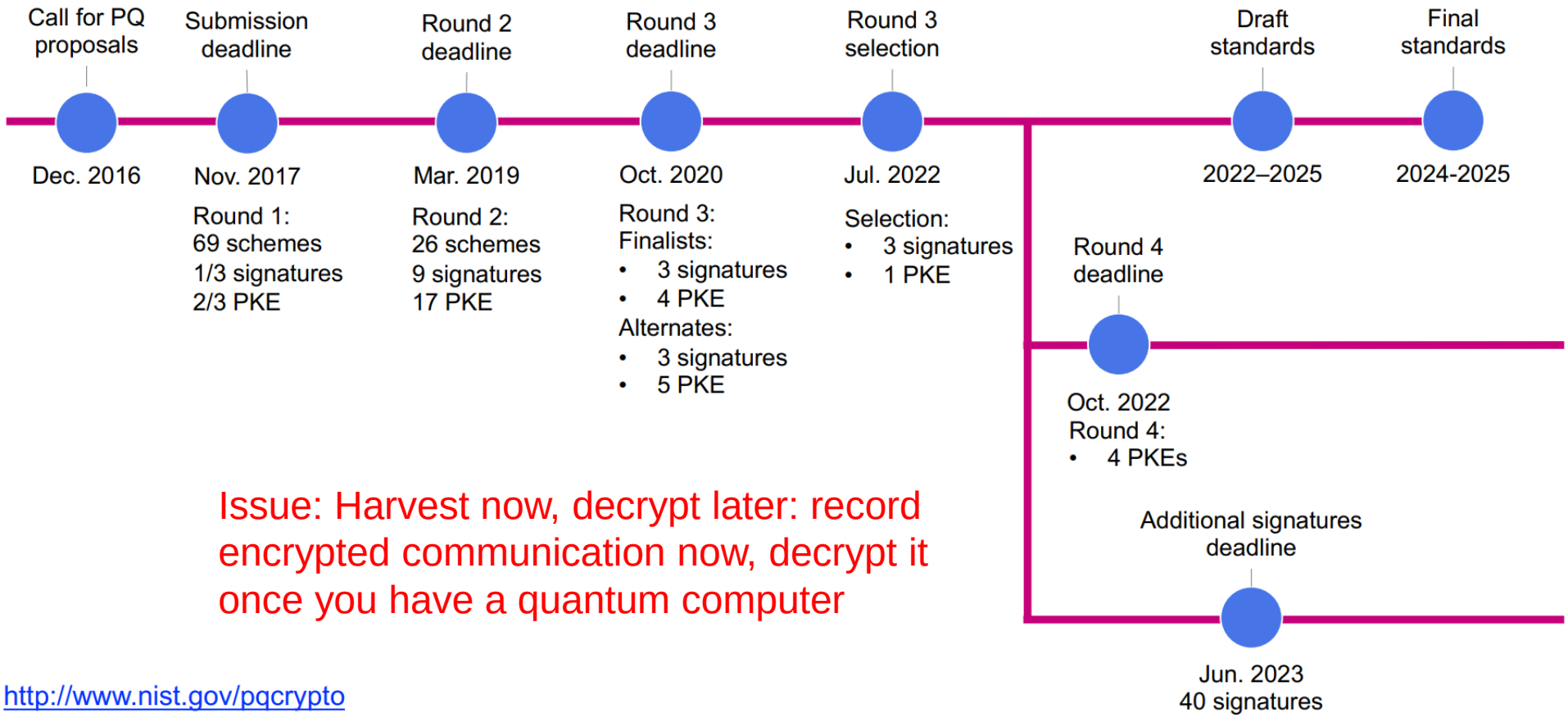
PQC Schemes

Post-Quantum Cryptography
(Quantum Safe Against Grover and Schorr Algorithms)



Douglas Stebila

Standardization of PQ cryptography



Issue: Harvest now, decrypt later: record encrypted communication now, decrypt it once you have a quantum computer

http://www.nist.gov/pqcrypto

Post-Quantum (PQ) Schemes

- NIST runs a standardization for PQ schemes:
  - https://csrc.nist.gov/projects/post-quantum-cryptography
- What is scandalized?
  - Digital Signatures
  - Key Encapsulation Mechanisms
- PQ aspects:
  - Lack of confidence in security
  - Slow computation
  - Large communication (big keys)
- There are many libraries, but often non-trivial to install:
  - https://libpqcrypto.org/index.html
    (not only standard installation but dependencies need to be installed separately)

Task 1

- Discuss in pairs what Key Encapsulation Mechanism is?
- What components could there be?
- Write pros of that solution?
- Write cons of that approach?
- What is the difference between PQC and ECC/RSA in this context?

Post-Quantum Cryptography
(Quantum Safe Against Grover and Schorr Algorithms)

Hash- & symmetric-based
- Can only be used to make signatures, not public key encryption
- Very high confidence in hash-based signatures, but large signatures required for many signature-systems

Code-based
- Long-studied cryptosystems with moderately high confidence for some code families
- Challenges in communication sizes

Multivariate quadratic
- Variety of systems with various levels of confidence and trade-offs
- Substantial break of Rainbow algorithm in Round 3
- Also crypt-currency based on Rainbow broken

Lattice-based
- High level of academic interest in this field, flexible constructions
- Can achieve reasonable communication sizes
- Cons: some quantum concerns, patent concerns

Elliptic curve isogenies
- Newest mathematical construction
- Small communication, slower computation
- Full break of SIKE in Round 4

Based on Douglas Stebila's presentation

NIST PQC standards

- Key encapsulation mechanisms
  - ML-KEM (FIPS 203)
    - a.k.a. Kyber
    - Lattice-based
- Digital signatures
  - ML-DSA (FIPS 204)
    - a.k.a. Dilithium
    - Lattice-based
  - SLH-DSA (FIPS 205)
    - a.k.a. SPHINCS+
    - Stateless hash-based
  - FN-DSA (draft pending)
    - a.k.a. Falcon
    - Lattice-based

PQ vs Classic algorithm sizes

Public key encryption scheme	Public key size (bytes)	Ciphertext overhead (bytes)
RSA-2048	256	256
ECDH (NISTp256, X25519)	32	32
ML-KEM-512	800	768
ML-KEM-768	1184	1088

Signature scheme	Public key size (bytes)	Signature size (bytes)
RSA-2048	256	256
ECDSA (NISTp256, Ed25519)	32	64
ML-DSA-44	1312	2420
SLH-DSA-SHA2-128s	32	7856
Falcon-512	897	752
XMSS / LMS	48–128	1600–25000+

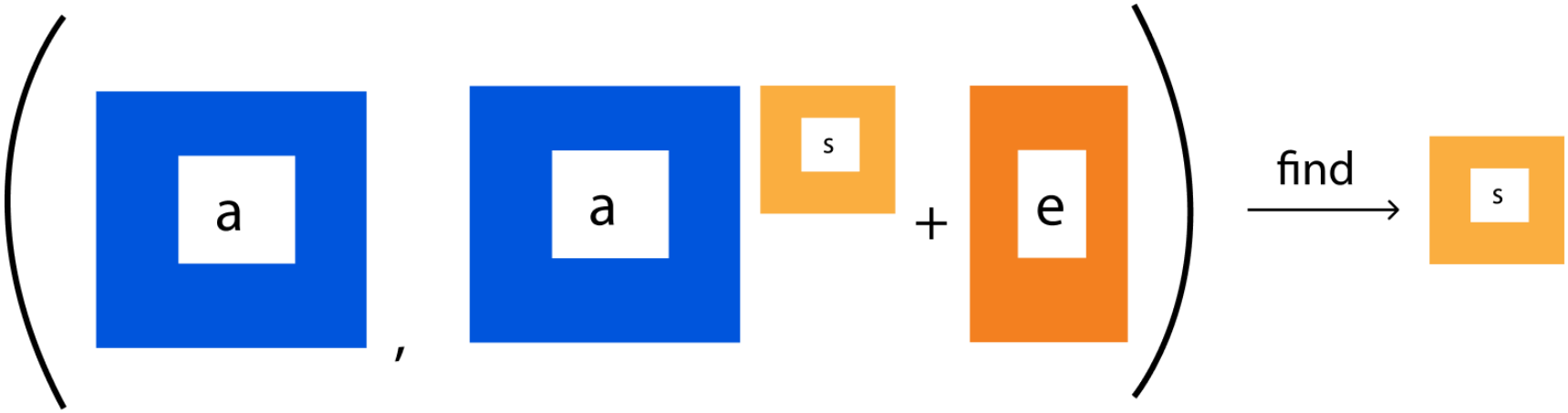
Douglas Stebila

Task 2

- Discuss in pairs how to approach “Lack of confidence in PQ security”?
- After 5 minutes: propose the main approach.
- We select one approach and then for 10 min discuss how would you implement it and:
  - List 3 pros
  - List 3 cons

The LWE problem: search and decision

- The Learning With Errors (LWE) problem asks to recover a secret vector $s=(s_1,\ldots s_n)$, where each s_i is in Z_q, given a sequence of random, "approximate" linear equations on s.



Determine if (a, b) is of the form above or random

https://blog.cloudflare.com/post-quantum-key-encapsulation/

Task 3 (Python)

- Check the code for ECC (ecc.py) and run it.
  - What do you see? What is the speed?
  - Add some averaging of multiple executions.
  - Modify it so the slowest curve is used.
  - Also, try the slow hash function.
- QuantCrypt: https://github.com/aabmets/quantcrypt
- Measure the time of two of the PQC schemes: one KEM and one Digital Signature.
- An example (without measurements) is in IS: pq1.py and pq2.py.

Task 4 (Optional, Java)

- Have a look at MLDSAExample.java
- What does this code do?
- https://www.bouncycastle.org/

- Implement time measurements and check the size of the keys, signatures etc.

Extra Materials

- QuantCrypt code examples:
  - https://github.com/aabmets/quantcrypt/wiki/Code-Examples
- Bouncy Castle
  - PQC Almanac:
    - https://downloads.bouncycastle.org/java/docs/PQC-Almanac.pdf
  - Test Code Examples (that code you need to simplify for homework):
    - https://github.com/bcgit/bc-java/tree/main/core/src/test/java/org/bouncycastle/pqc/crypto/test

Assignment 8 – Efficiency of Signature Generation Algorithms

- This is a programming assignment. Please upload your scripts/code and the required analysis via the course webpage.
- The deadline for submission is Dec. 6, 2024, 8:00.
  - -3 points for each started 24h after the deadline.
- Your code should be contained in one .py and one .java file. Please name the submission file as <uco_number>_hw8.zip. Put there both the python code, the java project folder, the analysis document, and all data produced during analysis (as long as the size is reasonable).
- The code must contain comments so that it is reasonably easy to understand how to run the script for evaluating each answer.

Assignment 8 - Tasks

1. Using quantcrypt implement the following signature schemes: Dilithium, Falcon, FastSphincs, SmallSphincs. In particular, use the library to run key generation, signature generation, and signature verification. **[1 points]**
2. Perform a performance time efficiency comparison analysis for these schemes. Analyze key generation, signature generation, and signature verification separately. Write a summary of your results, which primitive seems to be the best, and for which use case. Attach such a summary to your exercise submission. To have reliable results, perform operations a number of times and average results. Compare the results for ECDSA (using the same file) using the curve used during the seminar (SECP521R1, SHA3_512). **[4 points]**
   Remarks: (1) For the sake of computational time, use a small message (i.e., the alice.txt file from IS) to be signed. The same message should be used for all comparisons. (2) By “use case”, I mean a scheme, for example, some algorithms are slow but have fast verification, which might make them suitable for some applications. 3) The page limit on the attached analysis is 3 pages in total.
3. Perform a similar analysis with respect to the public key size, private key size, and signature size. Compare the results ECDSA (using the same file) using the curve used during the seminar (SECP521R1, SHA3_512) **[1.5 points]**
4. Implement a hybrid scheme and comment on its efficiency with respect to both execution time and sizes like in point 3 **[1 point]**
5. Implement Dilithium (ML-DSA) and Falcon (see the last link in extra materials) in Java and compare the efficiency results to quantcrypt. Can you comment on the results? The functionality should be analogical. **[2.5 points]**

Good luck!!!

Questions