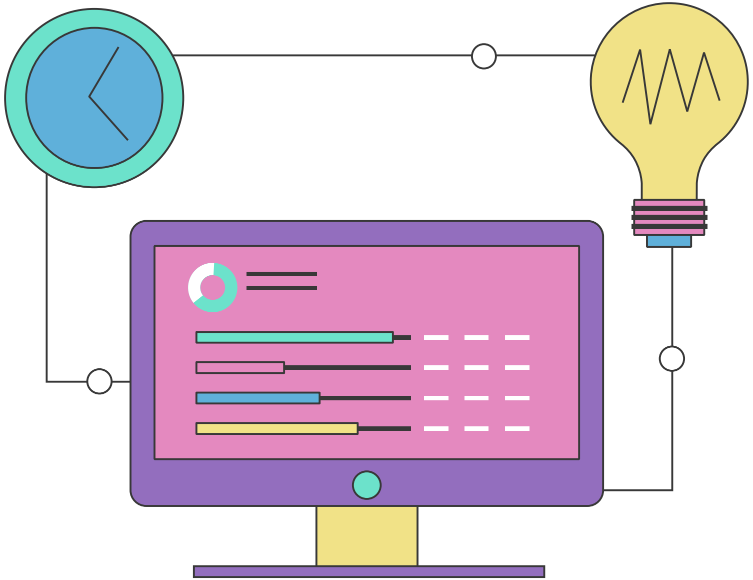
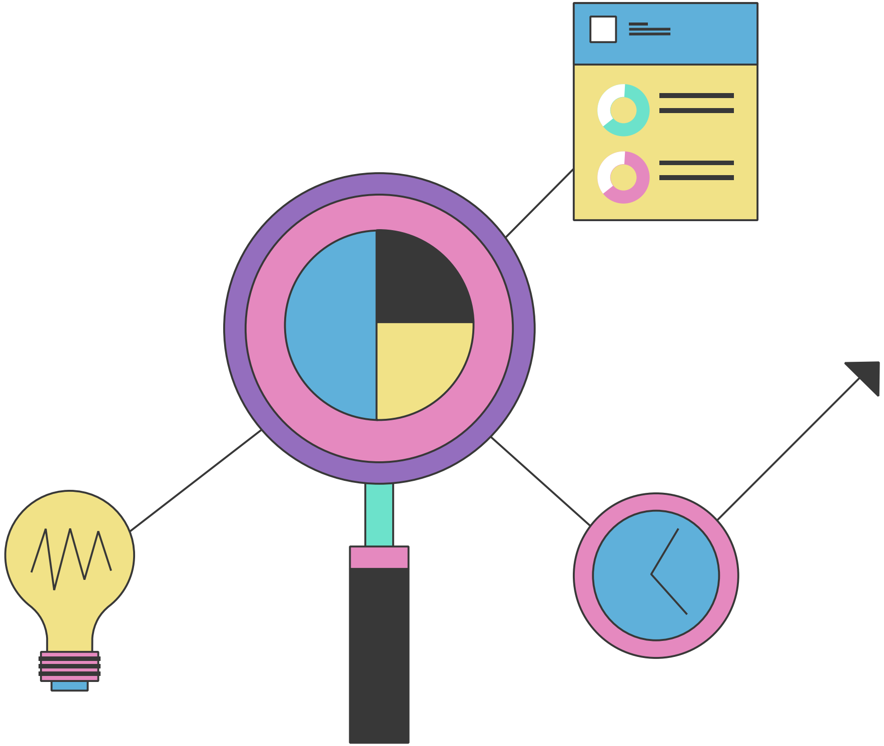


Making existing software quantum safe: a case study on IBM Db2

Sina Farahani

Helia Akbari

What is this research about?

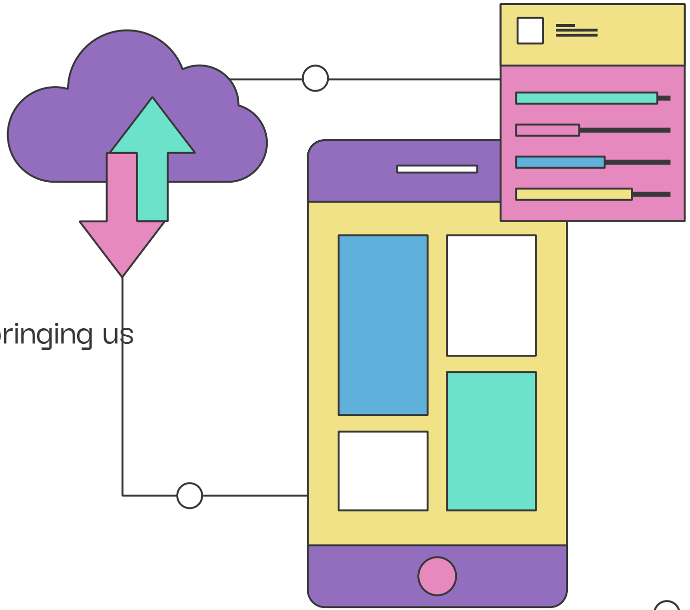


1. Introduction

1. Overview
    - QCs can break encryption algorithms

2. The timeline of QCs
    - 1982: idea from Richard Feynman
    - 1998: 2-qubit
    - 2017: 50-qubit, cloud-based QCs
    - 2019: Amazon Web Services
    - Many competitors are scaling up various QC architectures, bringing us closer to the day when QCs can solve practical problems.

3. Quantum advantage
    - Large QC can solve problems CC can't
    - 2019, 2020: no practical application
    - 20M qubits to hack 2048 RSA



4. Quantum advantage: impact on cybersecurity

- Integer factorization (polynomial), Solving discrete logarithmic problem
- Search in a set (O(n) → O($\sqrt{n}$))
- break RSA2048 with 20M qubits in less than a day
- A QC with 4099 perfectly stable, qubits can break RSA 2048 in 10 seconds

5. Quantum advantage: call to action

- **X**: time needed to deploy quantum-safe cryptographic solutions
- **Y:** time required to maintain the security of your encrypted data
- **Z**: time when a quantum computer capable of breaking existing encryption
- **X** > **Z:** 20 years to build FTQC. 20 years to build infrastructure
- **X** < **Z:** updating encryption proactively: topic of this paper
- **Y** > **Z:** Malicious entities can harvest sensitive data that must remain confidential for many decades
- **Y** < **Z:** data becomes stale and non-sensitive before the FTQCs arrive.

6. Quantum-safe: existing solutions

- **Quantum cryptography:** distributing entangled qubits (hard)
- **Post-Quantum Cryptography:** classical algorithms that are secure against both QCs and CCs (this paper)

2. The impact of quantum computing on existing systems

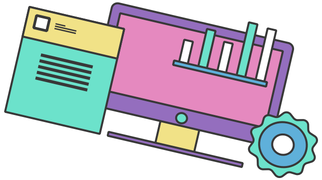
Encryption algorithm	Key size (bits)	Effective security level on CCs (bits)	Effective security level on QCs (bits)
RSA 1024	1024	80	0
RSA 2048	2048	112	0
ECC 256	256	128	0
ECC 384	384	256	0
AES 128	128	128	64
AES 256	256	256	128

1. **Asymmetric encryption**
   - Examples: RSA, DH, ECC
   - Weak: Shor's algorithm can perform integer factorization in polynomial time
   - Need to use quantum safe alternatives

2. **Symmetric encryption**
   - needs to perform a brute-force attack to break it
   - Key generation: (O(n) → O($\sqrt{n}$))
   - double key size to support the same level of protection.

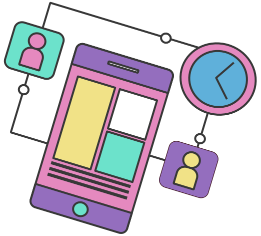
3. Challenges on existing systems

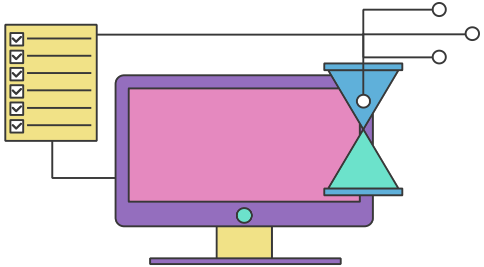
- **Action**: replace an existing asymmetric algorithm with a new one (or increase a key size of a symmetric one)
- Altering legacy systems is **challenging**
- Legacy systems lack adequate information or support to be maintained.
- the encryption-related code may be **spread** among **multiple** software **components**



4. Threats to the existing data

- Vulnerable to harvest-then-decrypt attack
- For the symmetric encryptions, we can increase the length of the key
- Encrypt archived data (stored on backup devices) with a quantum-safe algorithm.





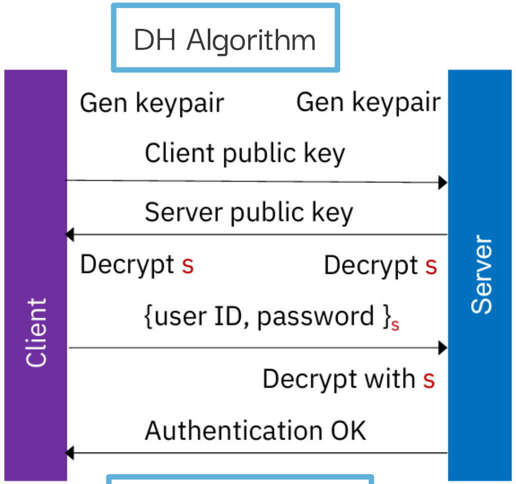
3. Industrial Setting

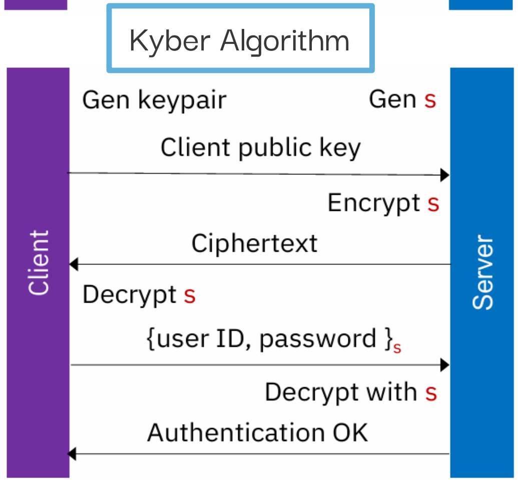
1. What is Db2?

- **mature** product (initially released in 1987, 34 years ago) with **Active** development
- codebase consists of **tens of millions** of lines of C/C++ code
- Db2 consists of a Db2 **server (**relational database management system) and data server **clients (**runs Db2 and SQL commands against the server).

2. Key exchange algorithm

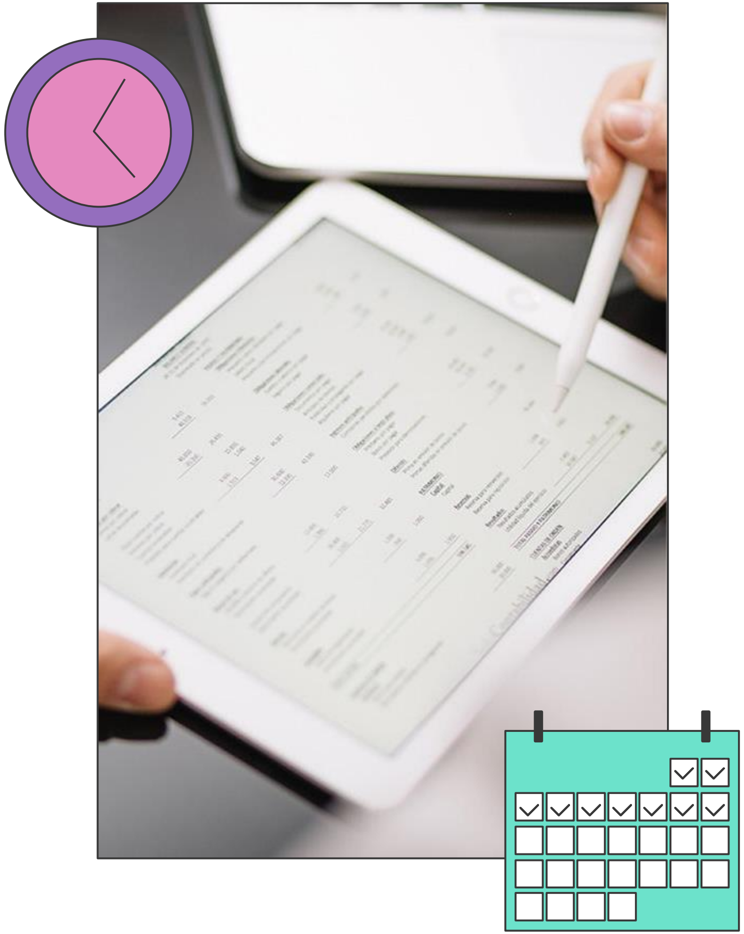
- Username and password
- Uses TLS protocol
- TLS uses both asymmetric cryptography (e.g., DH) and symmetric cryptography (e.g., AES) for encryption
- **DH key exchange** is vulnerable to quantum attacks.
- Use **Kyber** instead
- Kyber is a key encapsulation algorithm





3. Digital signature and TLS

- DH key exchange does not provide **authentication**: man-in-the-middle attacks.
- **RSA** digital signature is implemented alongside DH digital for authentication in TLS
- RSA is vulnerable → **Dilithium** as new digital signature scheme
- Upgrade TLS 1.2 to **TLS 1.3**
- **TLS 1.3** supports post-quantum authentication within IBM Global Security Kit (GSKit).



4. Experiments

1. **GSKit**
   - IBM GSKit provides libraries and utilities for both general purpose cryptography and Secure Sockets Layer (SSL) or TLS communication

2. **Implementation of Kyber**
   - We can use Kyber shared secret as a symmetric key (e.g., an AES 256 bit key) to protect channel between client and server.

1. **Dilithium and TLS 1.3**
   - The identity of the communicating parities are authenticated using asymmetric cryptography, i.e., RSA.
   - We need to replace RSA with quantum-safe cryptography, i.e., Dilithium.

1. **Performance evaluation**
   - The average response time even decreases by 0.635% after we migrate from DH to Kyber.

Algorithm	Avg. response time (ms)	St. Dev. (ms)
Kyber	162.514	15.281
DH	163.552	12.222

5. The 7E Roadmap

1) **Engage** executives and senior management

2) **Examine** existing products

3) **Evolve**: design a new software

4) **Educate** the programmers and designers

5) **Estimate** the impact of potential problems and the cost

6) **Execute** the new cybersecurity policy

7) **Essay** the new cybersecurity policy

1) Engage executives and senior management

- they can sponsor the initiative.
- can assess security concerns from a broader perspective.
- They will sponsor the allocation of the human.
- To educate the management, use formal presentations or reports and incorporate their feedback later.
- Engage multiple experts from different domains, because the evolution of the cybersecurity component often involves other parts of the system.

2) Educate the programmers and designers

- Ensure that everyone is on the same page because the security-related component is coupled with the remaining software components.
- the whole development organization needs to be aware of the challenges of quantum attacks.
- focus on training technical staff.
- We transfer knowledge and brainstorm within and outside this research team through work sessions, conferences, seminars, publications....

3) Examine existing products and their cybersecurity components

- identify and locate the issues
- review the document and programs
- assess the problems: for legacy systems, there
- exist di cult scenarios, such as lack of documentation, source code, or build infrastructure
- Identify existing data that may require protection.
- Note that the evolution of PQC may also a effect other components of the existing system.

4) Evolve: design a new software with crypto-agility

- quantum-safe algorithms may be added to the software later on.
- design systems in such a way that an existing encryption scheme can be easily replaced with a new one.
- the systems should be able to recognize and translate multiple encryption schemes.
- This will save costs in the future when the standards of PQC are finalized.
- Achieving crypto-agility requires that all business partners update hardware and software promptly. Moreover, all the partners should disclose crypto-related information.

5) Estimate the impact of potential problems

- prioritize the problems.
- The findings from Steps 2 and 3 should help to estimate the cost.
- Rate the cost of potential solutions in terms of human and time resources.
- prepare some buffer time in your project management.
- Upgrade of the cryptographic schemes involving symmetric encryption will typically be cheaper than the asymmetric one.

6) Execute the new cybersecurity policy

- Select and adopt appropriate solutions based on requirements, budgets, and priorities.
- For newly-built systems, PQC may be adopted.
- For legacy systems, the software and associated hardware may have to be altered
- For existing data, an intermediate solution e.g., re-encrypting the existing data with a quantum-safe cryptographic algorithm may be applied.

7) Essay the new cybersecurity policy

- Keep monitoring the performance and the robustness of your new cybersecurity policy in production
- make sure that the challenges associated with quantum advantage were addressed.
- adjust the policy if needed.
- Experiences and lessons learned from one project may also apply to another one.
- These lessons could serve as a building block to a general theory of making PQC evolution agile and smooth.

6. Challenges

Lack of documentation

- **Ambiguous, obsolete, erroneous, outdated** documentation
- Error in **API document**: socket handle parameter should have been an environment handle.
- the instructions were written for **32-bit** operating systems, but we were running on the 64-bit system.

Legacy/fragile development environment

Db2 project management requirements often lead to a limited number of unified legacy solutions. Many of them are **command-line tools**. Compared to graphical user interface, complex command-line interface has a steeper learning curve.

Large codebase

- **lack of comments:** challenging code evolution
- **complex structures:** project becomes less readable and understandable, making it more complicated to apply new changes to the system
- **long compilation time:** a complete build takes about two to three hours.

Distributed teams

- Upgrading a complex system often involves collaborations among multiple teams or organizations.
- A breakdown or lag in communication could cause delays in the development.
- During the PQC implementation, communication and collaboration among multiple geo-distributed teams with different backgrounds are required.

Technical debt

- Technical debt is a concept in software engineering that reflects the extra work caused by previous work when choosing an easy solution instead of applying the best overall solution
- Hard coding is one of the most common decisions that lead to technical debt.
- For example: hard coding key length

Underestimation of sizing

- As a cutting-edge and ever-changing technology, the application of PQC is still in its infancy, and developers have a steep learning curve for the application of PQC.
- We recommend that the readers err on the side of caution when estimating the amount of e ort and resources required for PQC evolution.

7. Take-away messages

Prepare ahead

- have a clear roadmap and timeline.
- follow the 7E steps to update your existing cryptography
- prepare for additional time for the development.

get support of management

PQC is a new emerging technology, it is important to educate your management and colleagues so that they are aware of the potential risk of data breach because of quantum attacks.

Collaborate with multiple departments

- the upgrade of cryptography can affect other components of the system or even cause failure of the system.
- we suggest that all stakeholders should be engaged in the upgrade as soon as possible to lower any risk to the system.

Document the development

- Document the development for future maintenance and evolution.
- NIST is still working on the standardization of PQC. This implies that PQC will evolve.
- It is a good idea to keep all the records of your development to simplify future changes.

Plan for crypto-agility

- The reason behind this is the evolution of PQC.
- we need to design the new cryptography with crypto-agility (such as dynamic key sizes).
- Paying of technical debt at this stage will improve the productivity in future development.

Measure performance impact

- we need to assess the new cryptographic systems performance and robustness.
- our finding shows that a PQC upgrade does not increase the time required to exchange cryptographic keys (and, on average, may reduce the timing slightly).

Thank you