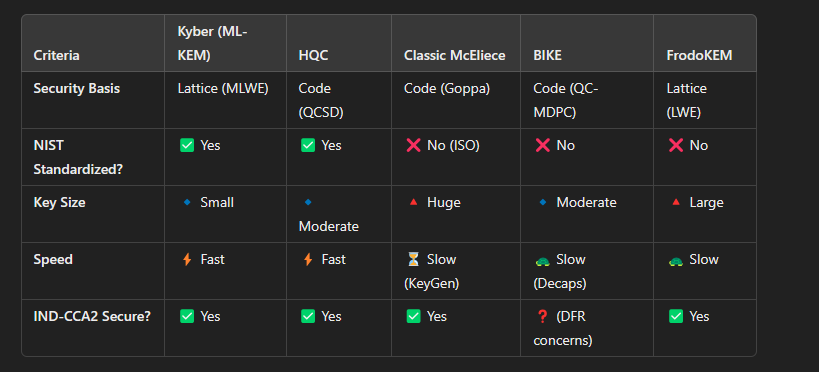
Algoritmos a comprobar:



1. NIST.IR.8413

Tercera ronda, lo anterior a los nuevos, importante tambien

1. NIST.IR.8545

4 Candidatos elegidos: BIKE, Classic McEliece, HQC, and SIKE. 🡪 Estandarizan HQC como alternativa a retículas (como backup), pero son mas lentos. Este reporte resume los criterios de evaluación de los candidatos, sus diseños básicos, sus ventajas y desventajas. Estandarizar HQC será el segundo después del kyber KEM.

1. NIST.SP.800-227.ipd

A key-encapsulation mechanism (KEM) is a set of algorithms that can be used by two par-ties under certain conditions to securely establish a shared secret key over a public channel. A shared secret key that is established using a KEM can then be used with symmetric-key cryptographic algorithms to perform essential tasks in secure communications, such as encryption and authentication. This document describes the basic definitions, properties, and applications of KEMs. It also provides recommendations for implementing and using KEMs in a secure manner.

1. Links algorithms <https://csrc.nist.gov/projects/post-quantum-cryptography/round-4-submissions>
2. 011318\_1\_5.0179566

Quantum key distribution provides secure keys with information-theoretic security ensured by the principle of quantum mechanics. The

continuous-variable version of quantum key distribution using coherent states offers the advantages of its compatibility with telecom industry,

e.g., using commercial laser and homodyne detector, is now going through a booming period. In this review article, we describe the principle

of continuous-variable quantum key distribution system; focus on protocols based on coherent states, whose systems are gradually

moving from proof-of-principle lab demonstrations to in-field implementations and technological prototypes. We start by reviewing the theoretical

protocols and the current security status of these protocols. Then, we discuss the system structure, the key module, and the mainstream

system implementations. The advanced progresses for future applications are discussed, including the digital techniques, system on

chip, and point-to-multipoint system. Finally, we discuss the practical security of the system and conclude with promising perspectives in this

research field.

1. s11042-024-20535-x

Development in Quantum computing paves the path to Quantum key distribution (QKD) by using the principles of quantum physics. QKD enables two remote parties to produce and share secure keys while removing all computing constraints on an adversary. The basic physics laws are used to identify any outside parties eavesdropping on the key exchange. In recent years many revolutionary developments in the field of QKD have been developed to overcome security and networking constraints. This survey provides an overview of the QKD protocol’s evolution and quantum network architecture. The paper also demonstrates QKD deployment techniques and elements of the QKD network. It also highlights ongoing design challenges by considering security and error estimation and correction, in contrast to studies concentrating on optical channels and equipment. Finally, this paper examines the possible directions for future research and offers design principles to guide the development of QKD and its related area.

We have surveyed the basic features of QKD and QKDN, both of which hold great promise for boosting the efficiency and security of communication systems. A few of the real-world obstacles that prevent them from being scaled up or used effectively have also been emphasized, including secret KR, distance, cost, compatibility, and security.

1. A COMPARATIVE REVIEW OF DATA ENCRYPTION METHODS IN THE USA AND EUROPE

Pa estado del arte, básicamente el titulo

1. An\_Overview\_and\_Analysis\_of\_Hybrid\_Encryption\_The\_Combination\_of\_Symmetric\_Encryption\_and\_Asymmetric\_Encryption

Ejemplo de implementación

1. Comparative\_Analysis\_of\_Energy\_Costs\_of\_Asymmetric\_vs\_Symmetric\_Encryption-Based\_Security\_Applications

Public key algorithms are heavily used in many digital applications including key establishment schemes, secure messaging apps, and digital signature schemes in cryptocurrencies. Recent developments in the field of quantum computation have placed these algorithms at risk as they enable the implementation of more effective attacks to derive the secret key. Most notably Shor's algorithm exponentially speeds up solving the factoring, discrete logarithm (DLP), and elliptic-curve discrete logarithm (ECDLP) problems. To address this challenge, NIST has initiated a process to develop and standardize a new quantum-resistant publickey cryptographic algorithm. However, asymmetric encryption schemes are known to be computationally intensive, hence energy demanding. The proliferation of energy-constrained internet of things devices, combined with the need to adopt higher complexity quantum resilient cryptographic algorithms, makes it more challenging to continue to use public-key algorithms for all applications. One approach to address these challenges is to adopt symmetric key systems, which are known to be more energy-efcient and more resilient to quantum computers-based attacks. This work performs a comprehensive comparison of energy costs between asymmetric and symmetric key schemes. This comparison is performed using two methods.

1. First-Order Masked Kyber on ARM Cortex-M4

Pos eso un kyber en un M4 to flama, latice based seguro. In this work, we presented a first-order masked Kyber specifically for the ARM Cortex- M4. We combined different approaches from previous works and practically verified thenfirst-order resistance with the ChipWhisperer Lite. As previous attacks, i.e. [NDGJ21]nhave shown, first-order masking is not enough to achieve practical side-channel resistance. Higher-order implementations should be combined with different countermeasures as shuffling or hiding. Additionally, the computational overhead of some operations in higher-order solutions still seems quite high and it might be part of future work to reduce it.

1. 2022-414

Algoritmo mixto: In this paper, we present the first quantum-resistant implementation of HPKE to address concerns that quantum computers bring to asymmetric algorithms. We propose PQ-only and PQ-hybrid HPKE variants and analyze their performance for two postquantum key encapsulation mechanisms and various plaintext sizes. We compare these variants with RSA and classical HPKE and show that the additional post-quantum overhead is amortized over the plaintext size.

1. An\_overview\_of\_Quantum\_Cryptography\_and

The paper aims to examine the mechanisms of quantum cryptography and review the relationship between quantum and classical encryption schemes. A brief introduction of quantum computation is provided, with a simplified explanation of Shor’s Algorithm showcasing the potentials of quantum computation. Related literature such as books, journals, proceedings, lecture notes and webpages on quantum cryptography were reviewed and were sourced from prominent databases like IEEE Xplore, ScienceDirect, and JSTOR. This gave a clearer picture on the mechanisms of quantum cryptography and Shor's algorithm. The authors were able to succinctly describe quantum cryptography, show how encryption is achieved by exploiting the properties of quantum particles, and demonstrated with examples the intricacies with Shor's algorithm. It is expected that interested researchers will be more informed on current research in quantum cryptography and influence potential cryptography scholars to explore further the mechanisms of quantum cryptography, quantum computation, and other principles of the quantum theory

1. An\_Efficient\_Quantum\_Computing\_technique\_for\_cracking\_RSA\_using\_Shors\_Algorithm

In nutshell, paper will discuss about various QC algorithms and will illustrate how shor’s algorithm is able to crack RSA.

1. 9508027v2

Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer. Basicamente el paper del algoritmo de shore

1. 0301141v2

Shor’s discrete logarithm quantum algorithm for elliptic curves. We show in some detail how to implement Shor’s efficient quantum algorithm for discrete logarithms for the particular case of elliptic curve groups. It turns out that for this problem a smaller quantum computer can solve problems further beyond current computing than for integer

factorisation. A 160 bit elliptic curve cryptographic key could be broken on a quantum computer using around 1000 qubits while factoring the security-wise equivalent 1024 bit RSA modulus would require about 2000 qubits

1. RESERCH FINAL

Implementation and Analysis of Shor’s Algorithm to Break RSA Cryptosystem Security. The goal of this paper is to assess the present state of Shor’s algorithm implementation and efficiency, specifically its application in the context of compromising public key cryptosystems like RSA.

1. 2502.12252v1

Roadmap to fault tolerant quantum computation using topological qubit arrays. We describe a concrete device roadmap towards a fault-tolerant quantum computing architecture based on noise-resilient, topologically protected Majorana-based qubits. Our roadmap encompasses four generations of devices: a single-qubit device that enables a measurement-based qubit benchmarking protocol; a two-qubit device that uses measurement-based braiding to perform single-qubit Clifford operations; an eight-qubit device that can be used to show an improvement of a two-qubit operation when performed on logical qubits rather than directly on physical qubits; and a topological qubit array supporting lattice surgery demonstrations on two logical qubits. Devices that enable this path require a superconductor-semiconductor heterostructure that supports a topological phase, quantum dots and coupling between those quantum dots that can create the appropriate loops for interferometric measurements, and a microwave readout system that can perform fast, low-error single-shot measurements. We describe the key design components of these qubit devices, along with the associated protocols for demonstrations of single-qubit benchmarking, Clifford gate execution, quantum error detection, and quantum error correction, which differ greatly from those in more conventional qubits. Finally, we comment on implications and advantages of this architecture for utility-scale quantum computation.

1. Quantum\_Resistance\_Saber-Based\_Group\_Key\_Exchange\_Protocol\_for\_IoT (1)

Interesante pa micros

1. Fips 202 estandar federal: funciones de hash
2. Fips 203 estandar federal iontercambio claves
3. BIke\_spec
4. HQC\_submision
5. Kyber
   1. Ideal-lwe: fundamentals of algebra?
   2. Fuyasaki transform: why the algorithms change for INDCCA2 security
   3. NTT
   4. Delta failure: <https://github.com/pq-crystals/security-estimates>

Noise selection, fundamentos

Different noise values Δ1\Delta\_1Δ1​ and Δ2\Delta\_2Δ2​. Notice that in Algorithm 5, there is additional implicit noise created via Compressq\text{Compress}\_qCompressq​. This has the effect of adding some (deterministic) noise to the ciphertext, which can be interpreted as increasing the noise of the error polynomials e1e\_1e1​ and e2e\_2e2​. If the decryption error probability is low enough, then it makes sense to also increase the noise of the other secret terms (i.e. sss; eee; rrr) to be at a similar level as e1e\_1e1​ plus the deterministic noise.

We utilize this idea (exclusively) for the Kyber512 parameter set. Relying on the rounding noise from Compressq\text{Compress}\_qCompressq​ to add error is akin to the LWR assumption. But unlike in (Ring/Module)-LWR schemes, where the security completely relies on the noise that’s deterministically generated by rounding, our dependence on the deterministic noise is much smaller. First, it only adds 6 bits of Core-SVP hardness, and second, we are adding noise and rounding together, which presumably has less algebraic structure than just rounding.

In short, without the LWR assumption, our parameter set for Kyber512 has 112 bits of Core-SVP hardness – more specifically, the public keys are protected with 118 bits, and the ciphertexts with 112; with a weak version of the LWR assumption, it has 118-bit security everywhere.

Chapter 4 security for later:

1. Saber

Procedimiento:

Step 1: Controller generates its keypair

-----------------------------------------

[ Controller (PSoC) ]

kem\_keygen(public\_key, private\_key)

Step 2: Controller sends public key to PC

-----------------------------------------

[ Controller ] --(public\_key)--> [ PC ]

Step 3: PC encapsulates a secret and sends ciphertext

-----------------------------------------------------

[ PC ]

kem\_encaps(ciphertext, shared\_secret\_PC, public\_key)

[ PC ] --(ciphertext)--> [ Controller ]

Step 4: Controller decapsulates to recover same shared secret

-------------------------------------------------------------

[ Controller ]

kem\_decaps(shared\_secret\_CTRL, ciphertext, private\_key)

Step 5: Both sides derive AES-256 key from shared secret

---------------------------------------------------------

[ PC ]

AES\_key\_PC = SHA256(shared\_secret\_PC)

[ Controller ]

AES\_key\_CTRL = SHA256(shared\_secret\_CTRL)

Step 6: Clean up sensitive data from RAM

-----------------------------------------

[ PC ]

secure\_memzero(shared\_secret\_PC)

(Optionally) secure\_memzero(ciphertext)

[ Controller ]

secure\_memzero(shared\_secret\_CTRL)

secure\_memzero(private\_key)

(Optionally) secure\_memzero(ciphertext)

✅ Only AES key remains in memory/flash (for future use)

✅ Now:

AES\_key\_PC == AES\_key\_CTRL → 🎉 Ready to use AES-256

**OPTION 1: Use Windows DPAPI (Data Protection API)**

This is built into Windows and super easy to use — it encrypts the key so only your user account or machine can decrypt it.

**🔧 How it works:**

1. You generate the AES key from the PQC shared secret.
2. You use CryptProtectData() to encrypt it before storing.
3. Later, use CryptUnprotectData() to load it.

**✅ Pros:**

* Easy and built-in
* No need to manage master passwords
* Protects against other users on same machine

**⚠️ Cons:**

* Not portable across machines (unless you export keys with extra flags)

**✅ 1. Keep the key in memory only as long as needed**

* Don’t keep it longer than necessary.
* For example, use it to decrypt/encrypt, then wipe it.

**✅ 2. Use secure memory clearing**

* Overwrite it with zeros (secure\_memzero()) or random junk when done.
* Use VirtualLock() on Windows to prevent paging (advanced but good).

**✅ 3. Consider using “protected memory” or secure enclaves**

* Windows: CryptProtectMemory() for memory encryption
* Use with secure APIs that avoid exposing raw key to your code

**✅ 4. Use low-level libraries that support secure key management**

* Libraries like libsodium or OpenSSL can store keys in special memory regions

Algorithms:

Bike: <https://bikesuite.org/>

HQC: <https://pqc-hqc.org/>

Pagina ppal: <https://csrc.nist.gov/Projects/post-quantum-cryptography/round-4-submissions>

Ronda 3: <https://csrc.nist.gov/Projects/post-quantum-cryptography/post-quantum-cryptography-standardization/round-3-submissions>

Kyuber: <https://pq-crystals.org/>

Saber: <https://www.esat.kuleuven.be/cosic/pqcrypto/saber/>