**CSE Department – Faculty of Engineering - MSA Spring 2025**

[**Computer Security CSE446**](https://e-learning.msa.edu.eg/course/view.php?id=1760)

**Course Project**

**Course Instructor: Dr. Ali Somua**

Quantum-Based Encryption System Using AES and RSA

Youssef Amr 220863

**Table of Contents**

[**Project Overview** 3](#_gjdgxs)

**Introduction** [7](#_3dy6vkm)

**System Architecture** [11](#_3znysh7)

**Implementation**

**Cryptoanalysis**

**Conclusion**

**References** [13](#_2et92p0)

**1.Project Overview**

This project presents a secure encryption model that integrates quantum-generated random numbers to create high-entropy keys for AES encryption. The AES key is more secure due to the usage of RSA, simulating a hybrid cryptographic system.

**2.Introduction**

**2.1 Motivation**

Randomness is one of the most important concepts in cryptography. the more unpredictable the system is the more strong and secure it is

**2.2 Problem statement**

Classical RNG (random-number-generators) are predictable and that would be a crucial problem for any cryptographic system that is main reliance is on True randomness

**2.2.1 How it is predictable**

RNG’S uses algorithms that are mostly deterministic which will lead to a case called pseudo-randomness.

Most PRNG’s are predictable due to the algorithm nature which will eventually lead to a weakness to your cryptographic system this predictability will affect key generation algorithms security (RSA,AES,etc)

and security of any virtual asset that rely on RNG

**2.3 Goal**

the goal of this project is to use quantum-randomness a principle that is based on superposition which will lead to the generation of truly random numbers which will overcome the limitations of PRNG

**2.3.1 Benefits of using QRNG in key generation**

1-Improved Entropy: AES generated keys are more random

2-True unpredictability: QRNGs usually cannot be reversed engineered

3-Higher Security: The combination of unpredictability and uncertainty will eventually lead to higher security system

**3.Solution outline**

This project present a secure encryption model that integrate quantum-generated random numbers the solution follows a hybrid layered encryption approach

**3.1.Quantum Random Number Generation (QRNG):**

A quantum circuit is built using Qiskit to generate random binary strings by applying Hadamard gates to qubits. The outcome provides high-entropy, truly random bit sequences.

**3.2.Key Derivation with SHA-256:**

A quantum circuit is built by using Qiskit to generate random binary strings by applying Hadamard gates on qubits. The result is high-entropy, And truly random bit sequences.

**3.3.AES Encryption (Symmetric Layer):**

The plaintext message is encrypted by using AES in EAX mode. This provides both confidentiality and integrity by generating a tag alongside the ciphertext and nonce.

**3.4.RSA Encryption (Asymmetric Layer):**

The AES key is encrypted using RSA with a 2048-bit key pair. This ensures secure key exchange, allowing the AES key to be transmitted or stored securely.

**3.5.Decryption Process:**

A quantum circuit is built using Qiskit to generate random binary strings by applying Hadamard gates to qubits. The outcome is high-entropy, truly random bit sequences.

**3.6.Validation and Analysis:**

The randomness quality of the QRNG output is verified by using entropy metrics, chi-square tests, and avalanche effect analysis to ensure cryptographic stability.

**System Architecture**

**QRNG(Quantum Random Number Generator)**

Generating random numbers by using a hardmard simulated circuit which puts qubits in superposition state. These gates place qubits into a superposition state, the output is bits in uncertain state and this will will lead to truly random bit output which is very suitable for cryptographic applications

**SHA-256 hashing of QRNG output → AES Key**

Using hashing function which will work by converting the raw qrng bits into suitable format for using in AES and also will provide more security to the bits because hashing will increase the variance between each bit sequence entropy wise which will lead to a more secure key

**AES Encryption of message**

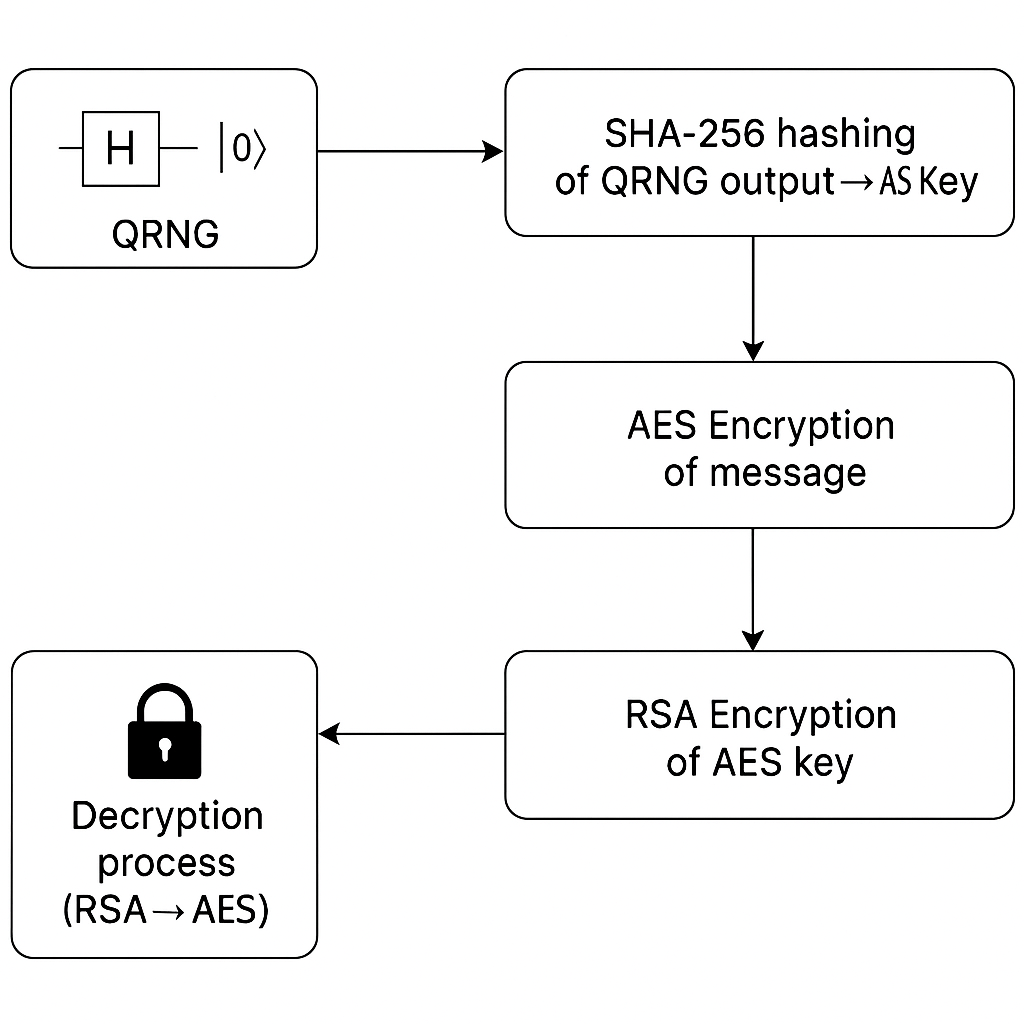
Taking the hashed Qrng output and applying aes algorithm to it, and this will form the first layer

**RSA Encryption of AES key**

The AES key is then encrypted using an RSA public key. This forms the second encryption layer in a hybrid model.

**Decryption process (RSA → AES)**

The model decrypts first the AES layer using the Rsa Public key and then the ciphertext is decrypted using AES with the recovered key to retrieve the original message.



**5.Implementation**

**5.1.QRNG**

**A Quantum circuit is constructed using Qiskit where each qubit is initialized into superposition via hadamard gate. when measuring it produce a truly random bitstring**

**code snippet example:**

**n\_bits = 8 # Choose the number of bits you want**

**n\_bits = 16 # -new modification in bit num to fit AES function req**

**qc = QuantumCircuit(n\_bits, n\_bits)**

**# Apply Hadamard to all qubits to create superposition**

**for i in range(n\_bits):**

**qc.h(i)**

**qc.measure(i, i)**

**A Hadamard gate is applied to each qubit to initiate superposition, then measured to generate 16 bit random value**

**5.2. Hashing to Derive AES Key**

**The raw QRNG is passed through SHA-256 hashing algorithm. The resulting 256-bit digest is used to derive the AES key**

****

**The first 16 bytes of the SHA-256 digest are used as the AES-128 key.**

**5.3. AES Encryption**

**The AES algorithm is applied to the plain text by using the hashed key EAX mode, which provide integrity by generating**

**Ciphertext**

**Nonce (used once per session)**

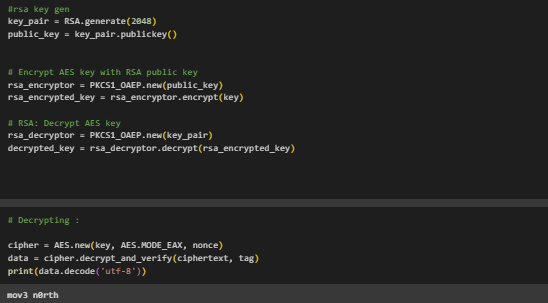
**Tag (authentication token)**

****

**AES is in EAX mode that encrypts the message and generates an authentication tag**

**5.4. RSA Key Generation, Encryption, Decryption.**

**The Rsa key bit is generated. The AES key is encrypted using the RSA public key for secure sharing.**

****

## 6. Cryptoanalysis

To assess the security and strength of the encryption system, multiple cryptographic analysis techniques were applied to both the key generation process and the output of the cryptographic functions.

### 6.1. Avalanche Test for SHA-256

The avalanche effect ensures that a small change in input (such as flipping a single bit) results in a significant, unpredictable change in the output hash.

Method: Two QRNG-generated bitstrings with 1-bit difference were hashed using SHA-256.

Result: More than 50% of the output bits changed, confirming a strong avalanche effect.

Conclusion: SHA-256 demonstrates excellent diffusion, making reverse-engineering the input infeasible.

### 6.2. Bit Distribution Test

This test checks the balance of 0s and 1s in the QRNG output to evaluate bias.

Method: Generated 10,000 bits using the QRNG circuit.

Result: Frequencies of 0s and 1s were approximately equal, with a ratio near 1:1.

Conclusion: The bitstream is statistically unbiased, validating QRNG randomness.

### 6.3. Chi-Square Test on QRNG Output

The chi-square test evaluates whether the distribution of 0s and 1s deviates significantly from a uniform distribution.

Method: Applied chi-square test to the observed frequency of bits.

Result:

Chi-square statistic: *≈ small value*p-value: *> 0.05*

**Conclusion: The p-value indicates that any variation is statistically insignificant. The distribution is considered uniform.**

### 6.4. Entropy of Generated Data

Entropy measures the unpredictability or information density of a dataset. Ideal binary entropy is 1 bit per bit.

Method: Calculated entropy of the 10,000-bit QRNG output.

Result:

Entropy ≈ 7.80–7.99 bits for 8-bit blocks (max = 8.0)

Conclusion: High entropy confirms the high-quality randomness of QRNG output.

### 6.5. Interpretation of Results

All tests confirmed that:

The QRNG generates truly random and unbiased bits

The hashing algorithm (SHA-256) produces highly sensitive, irreversible output

The resulting AES key has sufficient entropy and variability

The encryption system is resistant to statistical attacks

**Overall, the system demonstrates strong cryptographic properties suitable for secure communication and key generation.**

## 7. Conclusion

**This project successfully demonstrates the practical application of quantum randomness in modern cryptographic systems. By integrating a simulated Quantum Random Number Generator (QRNG) with classical encryption techniques (AES and RSA), the system enhances entropy and key unpredictability—core requirements for secure data encryption.**

### Key Findings

**The use of Hadamard-based QRNG produced unbiased, high-entropy binary sequences.**

**SHA-256 effectively increased key uniformity and avalanche diffusion.**

**AES encryption in EAX mode provided confidentiality and integrity.**

**RSA secured the AES key in a hybrid cryptographic scheme.**

**Cryptoanalysis tests (entropy, chi-square, avalanche) validated the system’s randomness and security.**

### Limitations

**The project currently lacks an active transmission layer (e.g., networking or device-to-device messaging).**

**Quantum randomness was simulated, not generated from hardware-based quantum devices.**

### Future Work

**Implement end-to-end encrypted transmission using Python socket programming (sender/receiver model).**

**Explore blockchain-based randomness verification using Chainlink VRF.**

**Test the system against real-world attack models and expand to hardware QRNGs or IBM Quantum backends.**

## 

## 8. References

1. **Qiskit Documentation – Quantum Circuit Simulation and Measurement  
    https://qiskit.org/documentation/**
2. **PyCryptodome Documentation – AES and RSA Implementation in Python** [**https://pycryptodome.readthedocs.io/**](https://pycryptodome.readthedocs.io/)
3. **NIST Cryptographic Standards – AES, RSA, and SHA** [**https://csrc.nist.gov/publications/**](https://csrc.nist.gov/publications/)
4. **William Stallings, *Cryptography and Network Security*, 7th Edition – Pearson, 2016.**
5. **M. Nielsen & I. Chuang, *Quantum Computation and Quantum Information*, Cambridge University Press.**
6. **Chainlink VRF – Verifiable Random Function for Smart Contracts  
    https://docs.chain.link/vrf**