Quantum Random Number Generator for One Time Password Generation

Vito Cucinelli

Gian Luca Schiano

Alexandre Gautier

Mirko Ciardo

Random Number Generators

Definition

RNGs are algorithms or devices used to produce a sequence that lack any pattern

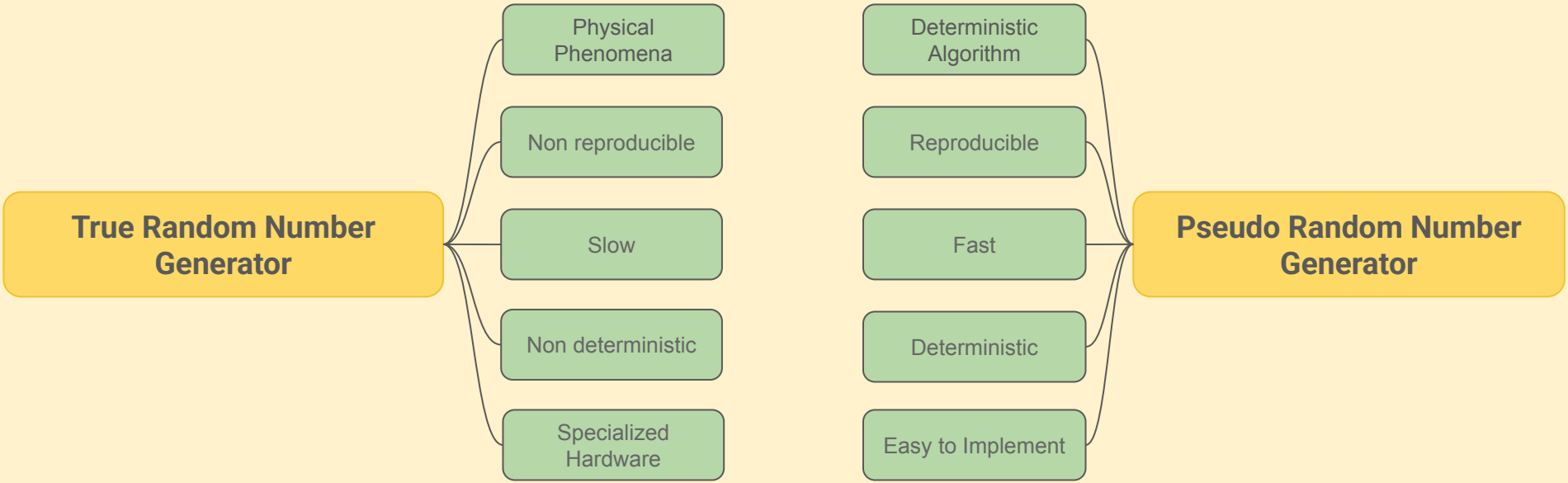
Applications

- Cryptography
- Gaming
- Simulations
- Statistical Sampling

Key Components

- **Seed**: Initial value to start the process
- **Algorithm:** How numbers are generated
- **Output:** Random sequence

TRNG vs PRNG


True Random Number Generator
Physical Phenomena
Non reproducible
Slow
Non deterministic
Specialized Hardware
Deterministic Algorithm
Reproducible
Fast
Deterministic
Easy to Implement
Pseudo Random Number Generator

Quantum Random Number Generator

Quantum Mechanics

Superposition

$$|\psi\rangle = \frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle$$

Measurement

50% → $|0\rangle$

50% → $|1\rangle$

True Random Number Generator!

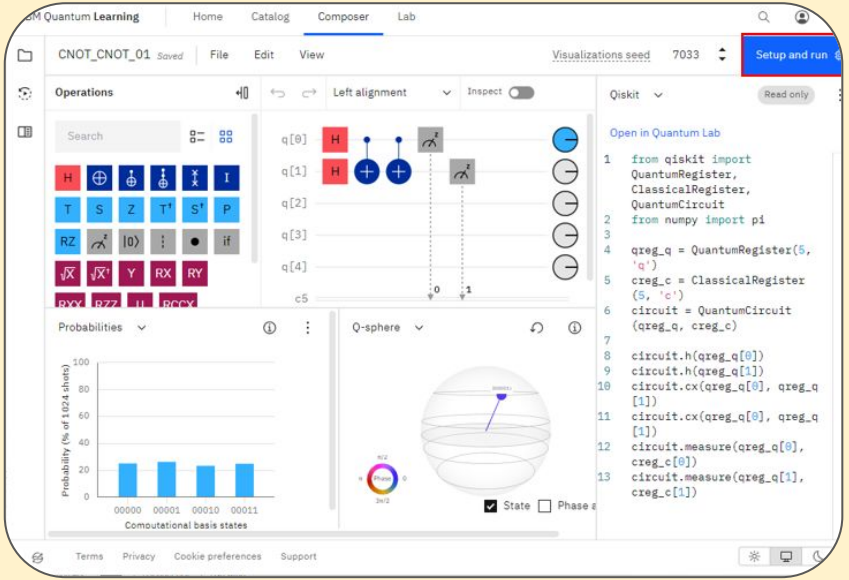
Fundamental Tools for Quantum

IBM Quantum Platform

- Access to a real Quantum computer
- Circuit simulation

Qiskit

- Python framework for quantum computing simulation

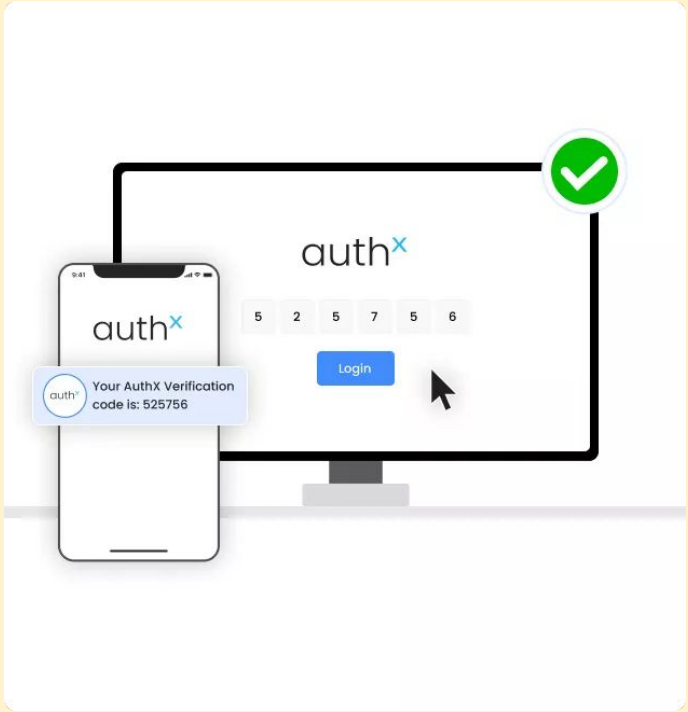


One-time Passwords

A **One-Time Password (OTP)** is a code valid for only one login session or transaction. Unlike static passwords, OTPs are dynamic and expire after use or after a short time period.

- Single Use
- Time/event based
- Flexibility in delivery methods,using a different channel

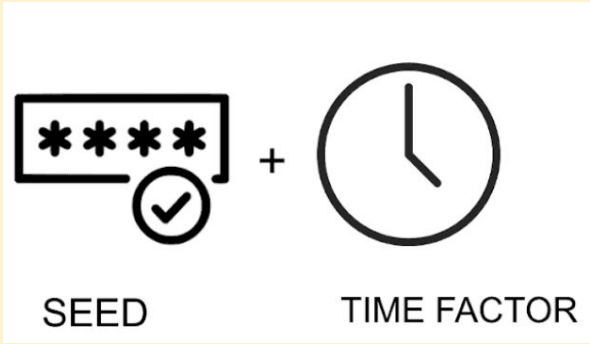
The main purpose of OTP is strengthen authentication by adding a unique, temporary code that can only be used once



Types of OTP

Event-based OTP (HOTP) generates a new code each time a specific action or event occurs, such as a button press or login attempt.

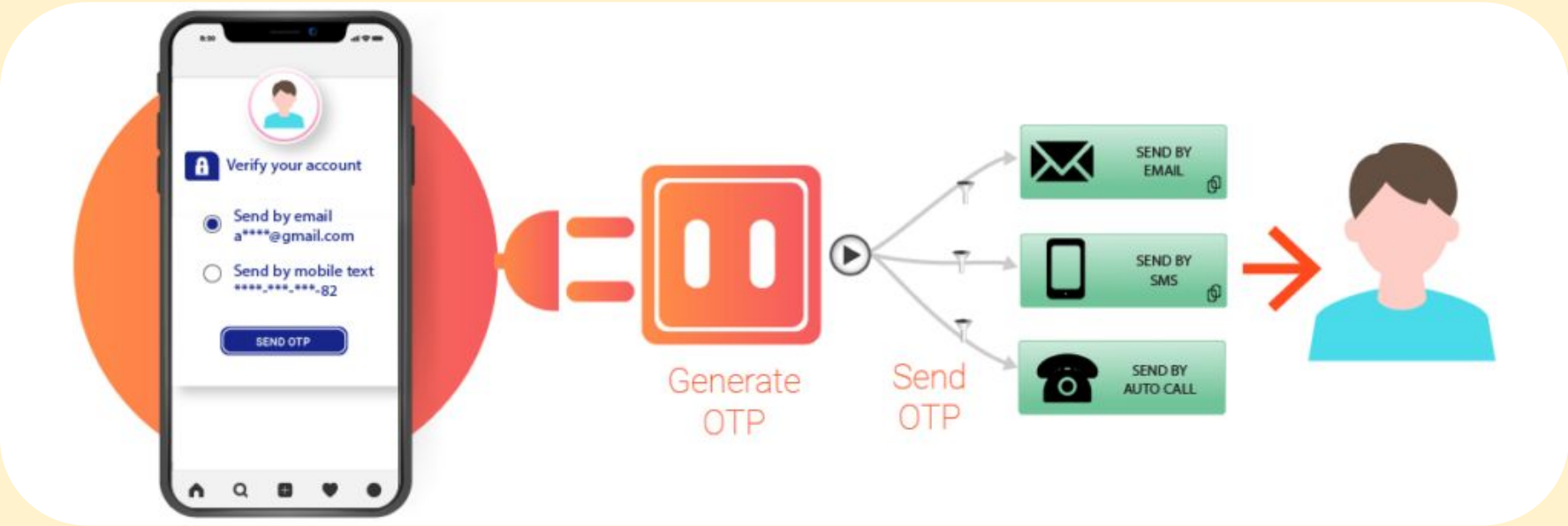
1. **Hardware token devices** used in corporate environments for secure login.
2. **Banking systems** requiring OTP generation for each transaction or login.
3. **Smart card authentication** where codes are generated on demand.



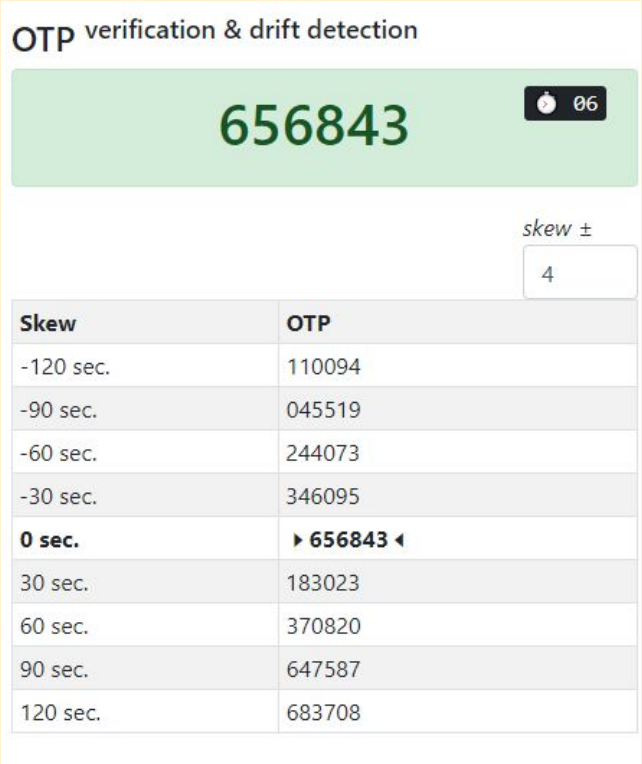
Time-based OTP generates new codes valid for a fixed time interval depending on the application

1. **Two-Factor Authentication (2FA)** apps like Google Authenticator or Authy.
2. **Cloud services login**, e.g., Gmail, AWS, or GitHub with time-sensitive codes.
3. **Mobile banking apps**

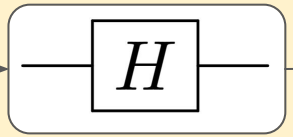
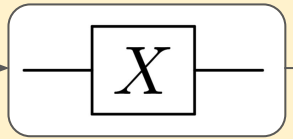
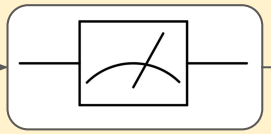
Out of band


Verify your account
Send by email
a****@gmail.com
Send by mobile text
****-***-***-82
SEND OTP
Generate OTP
Send OTP
SEND BY EMAIL
SEND BY SMS
SEND BY AUTO CALL

Seed

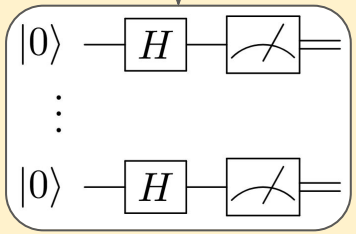

OTP verification & drift detection
656843
06
skew ±
4
Skew
OTP
-120 sec.
110094
-90 sec.
045519
-60 sec.
244073
-30 sec.
346095
0 sec.
656843
30 sec.
183023
60 sec.
370820
90 sec.
647587
120 sec.
683708

Quantum Gates

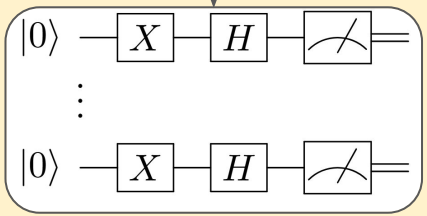
Hadamard gate		Superposition
X gate		Not operation
Measurement		State collapse

Quantum Circuits for QRNG

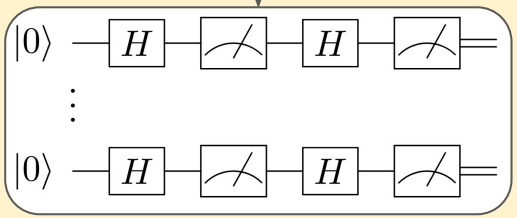
Basic QRNG



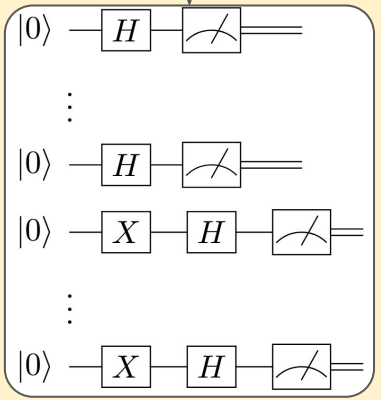
Type 1



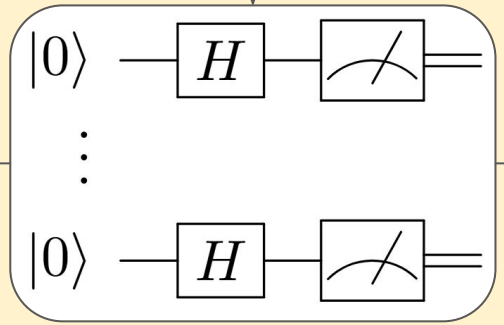
Type 2



Type 3



Basic QRNG



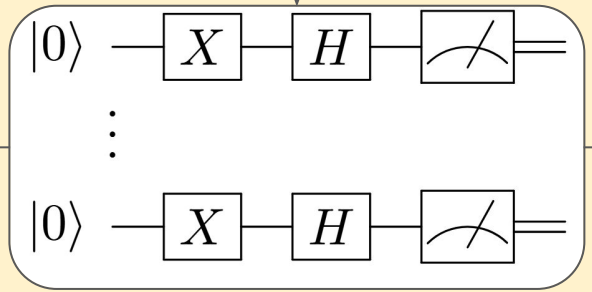
Advantage

Simple design for easy implementation

Drawback

No bias characterization capability

Type 1 QRNG



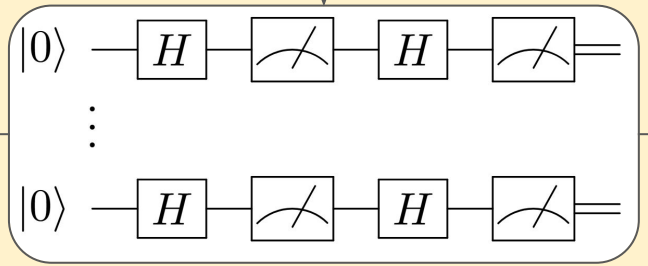
Advantage

It explores how bit-flip symmetry affects the output distribution

Drawback

Additional gate introduces further potential sources of error

Type 2 QRNG



Advantage

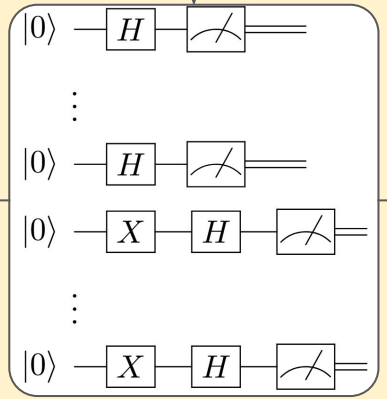
Evaluates gate-measurement correlation

Drawback

Extremely sensitive to decoherence and measurement-induced errors

CHOSEN FOR THE EXPERIMENT!

Type 3 QRNG

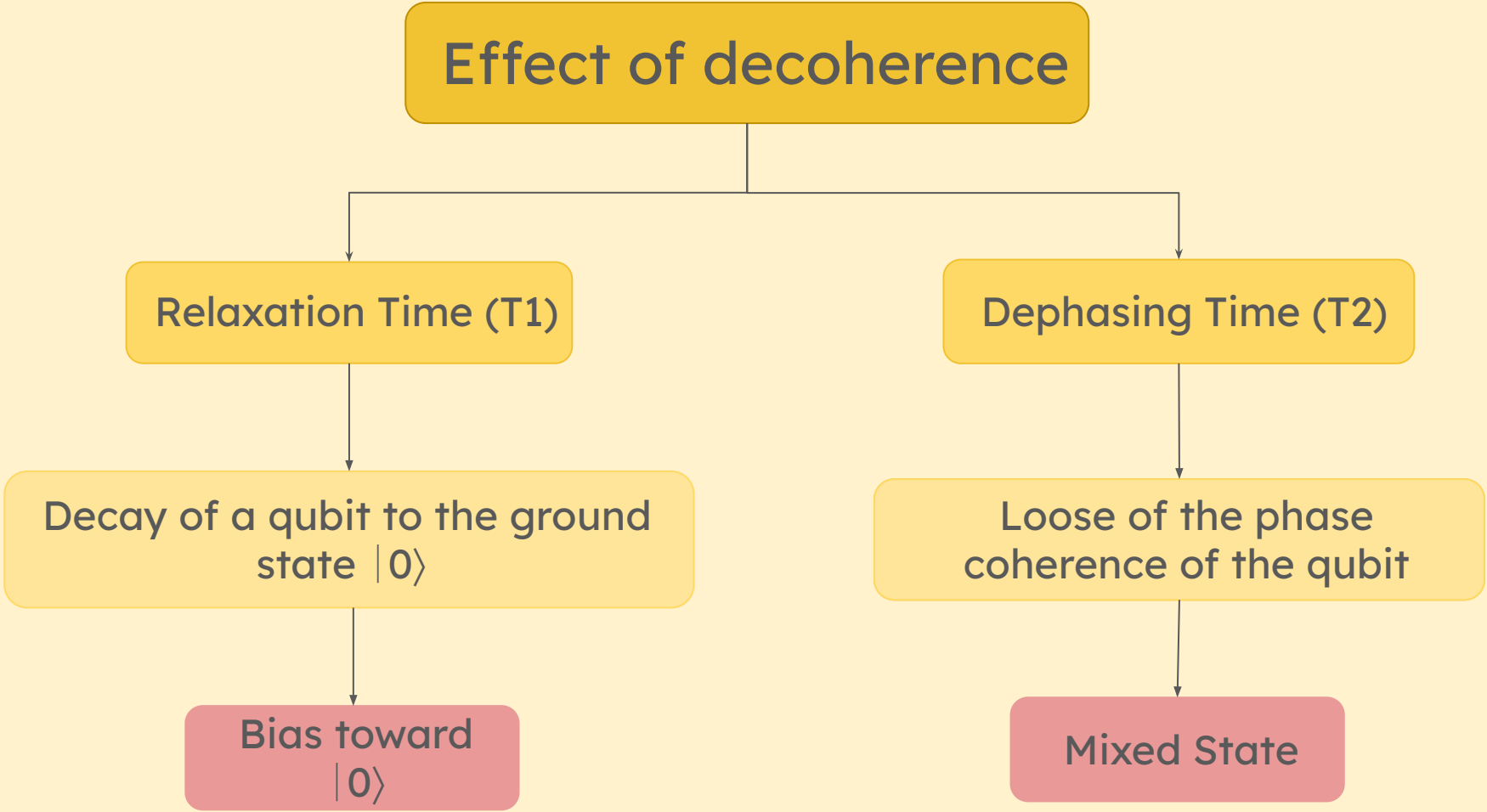


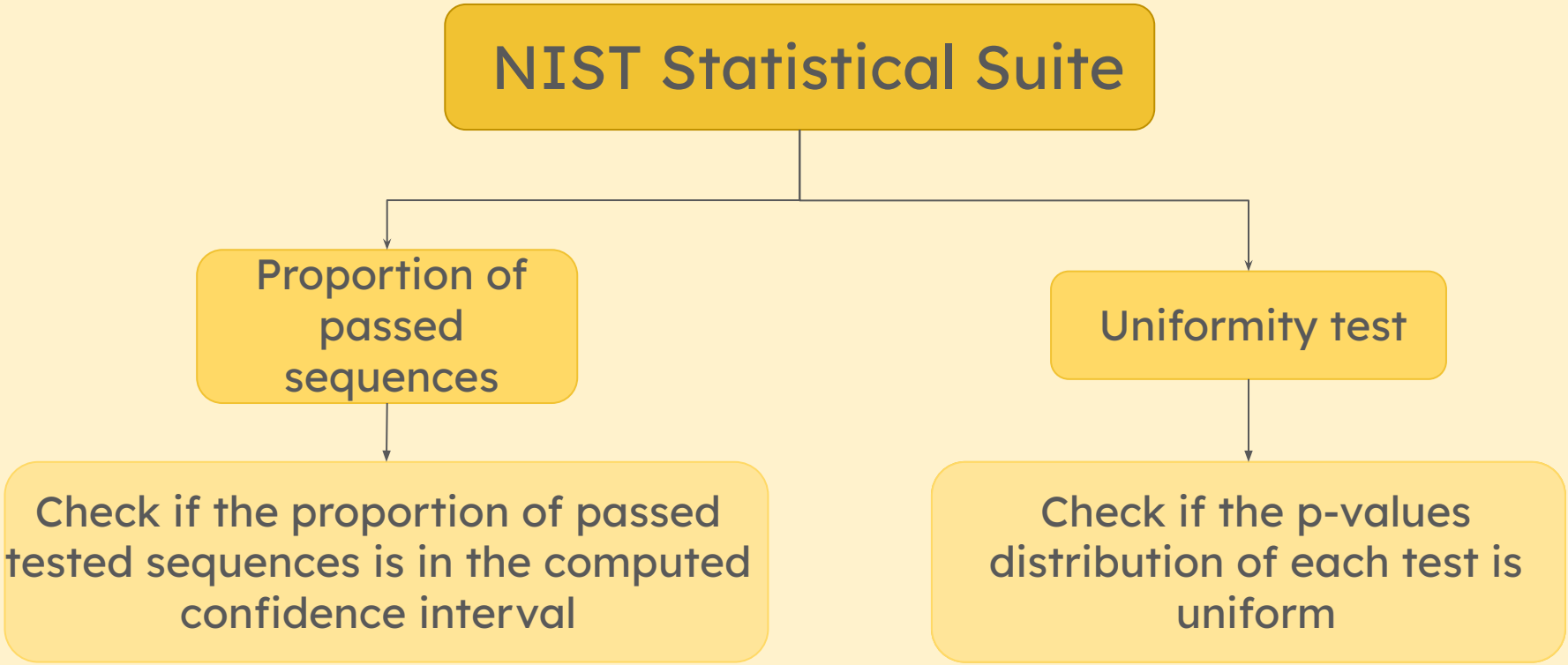
Advantage

Mitigates initialization biases through complementary setups

Drawback

Increased complexity


Effect of decoherence
Relaxation Time (T1)
Dephasing Time (T2)
Decay of a qubit to the ground state |0⟩
Loose of the phase coherence of the qubit
Bias toward |0⟩
Mixed State


NIST Statistical Suite
Proportion of passed sequences
Uniformity test
Check if the proportion of passed tested sequences is in the computed confidence interval
Check if the p-values distribution of each test is uniform

Experiment

Length = 1,040 million
Block size = 19
Sample size = 139
Quantum computer = ibm_sheerbroke

- As input a sequence of binary digits

In this code snippet we create the quantum random number directly from an account, specifying the token

```
# 65 mila shots to have a 1 milion bit number
# "447ab388994f0e83d45c52c41eb003beb31c6932a2e941e60b3b3a708a0995449d76517def2e1d141287186156dbfb2bef29095442da9d1e62ed9e329ff843e3"
g = QRNG()
g.chooseCircuit(QRNG_type=3, qubits_number=16)
M_number = g.generate_Numbers(num_qrn = 50, token ="" ,quantum_computer = "ibm_sheerbroke", num_shots = 65000, verbose = False, simulation = False)
```

+ Code + Text

In this code snippet, we use the function retrieve_from_IBM to retrieve the jobs from the different account(since we have only 10 minutes available for each one) and the function will collapse the results in only one output. The txt file should formatted to have the token in the first row and in the following all the job_ids

```
lista_job = [
    "job_alex_1.txt",
    "job_mirko_2cav.txt",
    "job_alex_2.txt",
    "job_mirko_3hotmail.txt",
    "job_alex_3.txt",
    "job_vito1_pallavolo.txt",
    "job_gianlu1.txt",
    "job_vito2_s333996.txt",
    "job_gianlu2gianluca.schiano@yahoo.txt",
    "job_vito_3vitocucinelli05.txt",
    "job_gianlu3schianog399.txt",
    "job_vito_4vitov2.txt",
     "job_mirko_1.txt"

]
qrns_from_IBM = g.retrieve_from_IBM(files = lista_job)
```

Experiment

- Apply a series of different statistical tests (currently 15 main tests in SP 800-22 Rev 1a, some with sub-tests). For each test applied to a sequence, the suite calculates a p-value.
- A significance level denoted by α is chosen (α = 0.01).
  - If the calculated p-value ≥ α, the sequence is considered to have passed that specific test,
  - If the calculated p-value < α, the sequence is considered to have failed that test.

We choose the parameter alpha to use to run statistical test. The parameter indicates the number of numbers that will fail the test(in this case only 1% of the sample size)

```
g.statistical_test(alpha = 0.01,list_qrns=M_numbers)
```

```
def statistical_test(self, alpha,list_qrns = None):
  self.NIST_tests(verbose=True,list_qrns=list_qrns)
  self.proportion_passed_sequences_test(alpha = alpha)
  self.uniformity_test(alpha = alpha)
```

Proportion of passed sequences

- Frequency (Monobit) test, Cusum test and Runs test are the 3 most important tests and fails
- Frequency within Block test, derives from Frequency test
- All these test fails, but Frequency within Block test has an higher p-value than Frequency Test

Systematic Bias

Table 1: Proportion of passed sequences

Test Name	Pass Rate	Verdict
Frequency (Monobit)	0.292 857	FAIL
Frequency within Block	0.592 857	FAIL
Runs	0.278 571	FAIL
Longest Runs in Block	0.728 571	FAIL
Matrix Rank	0.992 857	PASS
Discrete Fourier Transform	0.971 429	PASS
Overlapping Template Matching	0.964 286	FAIL
Maurer's Universal	1.000 000	PASS
Linear Complexity	0.978 571	PASS
Serial	0.996 429	PASS
Approximate Entropy	1.000 000	PASS
Cumulative Sums (Cusum)	0.257 143	FAIL
Random Excursions	0.894 643	FAIL
Random Excursions Variant	0.963 492	FAIL

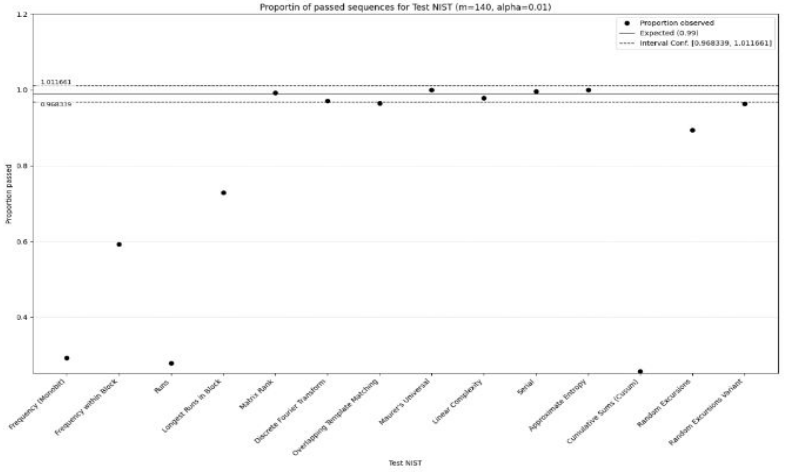
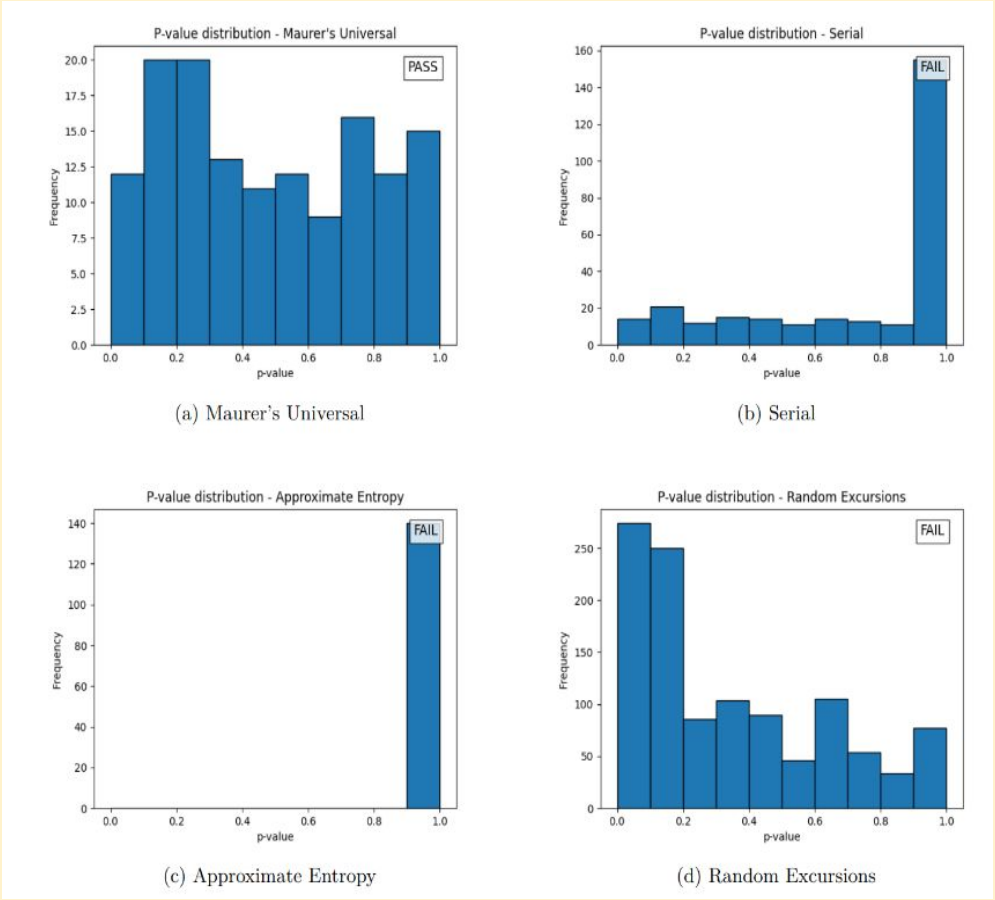


Figure 2: Proportion of passed sequences

Uniformity Test

- Serial and Approximate entropy test pass the first test, but the frequency of p-value is not uniformly distributed.
- For Serial and Approximate entropy(and not only) the uniformity test is failed
- If the random number generator is good, should produce numbers that have uniformly distributed p-values

Systematic Bias



(a) Maurer's Universal (b) Serial

(c) Approximate Entropy (d) Random Excursions

- https://github.com/Vitocuc/QRNG
- Honno. sts-pylib: Python interface to the NIST statistical tests for randomness. Accessed: 2025-05-04.2020. url:https://github.com/honno/sts-pylib.
- Lawrence E. Bassham III et al. A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications. NIST Special Publication 800-22 Rev 1a. Gaithersburg, MD:National Institute of Standards and Technology, Apr. 2010. doi: 10.6028/NIST.SP.800-22r1a. url:https://doi.org/10.6028/NIST.SP.800-22r1a.
- R. B. Prajapati and S. D. Panchal. “Enhanced Approach To Generate One Time Password (OTP) Using Quantum True Random Number Generator (QTRNG)”. In: International Journal of Computing and Digital Systems 15.1 (2024), pp. 279–292. doi: 10.12785/ijcds/150122. url: https://doi.org/10. 12785/ijcds/150122.
- Owen Root and Maria Becker. “Does True Randomness Exist? Efficacy Testing IBM Quantum Computers via Statistical Randomness”. In: arXiv preprint arXiv:2401.12250 (Jan. 2024). 12 pages, 6 figures.doi: 10.48550/arXiv.2401.12250. url: https://arxiv.org/abs/2401.12250.