1. RSA and Elliptic curve cryptography (ECC).

RSA relies on: The difficulty of factoring large integers. No known classical algorithm can do this efficiently.

ECC relies on: The difficulty of the Elliptic curve Discrete logarithm Problem (ECDLP) - finding the scalar k given P and kP on an elliptic curve.

- RSA and ECC are considered computationally secure: solving their underlying math problems would take thousands of years.
- Potential impacts:

1. HTTPS; VPN's; Email - encryption

2. Quantum key distribution (QKD) plays a fundamental role in the vision of future proof, quantum-safe cryptographic systems.

- confidentiality: No third party can learn the key without being detected.
- security is guarantee by quantum mechanics, not by the hardness of math
- Quantum channel: Alice sends qubits encoded with random bit values in randomly chosen bases.
- measurement: bob measureme the incoming qubits using randomly chosen bases.
- classical channel: They communication publicly to compare which measurem-

bases matched

- key sifting: From the matching measurement, they extract a raw shared key.
- Trusted-node networks: QKD can link via "trusted" nodes in a secure network, but lacks the long-distance efficiency of public-key systems.
- Hybrid-crypto systems: Ideal for defense, national security and critical infrastructure requiring long-term confidentiality
- Distance: current QKD systems are limited to ~100-200 km without repeaters.
- Hardware requirements: Needs specialized quantum communica-

2. Quantum Resistance:

- **RSA:**
  - Not quantum-resistance.
  - shor's algorithm: efficiently factors large integers on a quantum computers, breaking RSA.
- **Lattice-Based crypto:**
  - currently believed to be resistant to quantum attacks.
  - No known quantum algorithm efficiently solves lattic problems, like SVP or LWE.
  - They are Prime candidates for post-quantum cryptography.

Security foundations: RSA security depends on the difficulty of factoring integers, a single specific

mathmaticall problem.

- Lattic - based schemes:
- security reduces to worst-case hardness of lattic problems, meaning breaking the scheme is as hard as solving the hard instances of certain lattic problems.
- often have stronger theoretical security guarentees.
- NTRU: practical lattic-based encryption. RSA and other classical crypto systems are broken by powerful quantum computers.

1) RSA and Elliptic curve cryptography (ECC).

RSA relies on: the difficulty of factoring large integers. No known classical algorithm can do this efficiently.

ECC relies on: the difficulty of the Elliptic curve Discrete logarithm Problem (ECDLP) - finding the scalar k given P and kP on an elliptic curve.

- RSA and ECC are considered computationally secure: solving their underlying math problems would take thousands of years.
- Potential impacts:

1. HTTPS; VPN's; Email - encryption

and most secure messaging apps rely on RSA or Ecc.

- A quantum adversary could encrypt intercepted traffic.

- Post Quantum cryptography:
cryptography alogrithms designed to be secure aguinst quantum attacks.

- Hybrid cryptographic systems:
- combine classical and post-quantum algorithms to ensure compatibility and future proofing.

- Quantum key Distribution:
uses principles of quantum mechan of secure key exchange.

2. Quantum key distribution (QKD) plays a fundamental role in the vision of future proof, quantum-safe cryptographic systems,

- confidentiality: No third party can learn the key without being detected.
- security is guarantee by quantum mechanics, not by the hardness of math
- Quantum channel: Alice sends qubits encoded with random bit values in randomly chosen bases
- Measurement: bob measureme the incoming qubits using randomly chosen bases.
- classical channel: They communication publicly to compare which measureme

bases matched.

- key sifting: From the matching measurement, they extract a raw shared key.
- Trusted-node networks: QKD can link via "trusted" nodes in a secure network, but lacks the long-distance efficiency of public-key systems.
- Hybrid-crypto systems: Ideal for defense, national security and critical infrastructure requiring long-team confidentiality.
- Distance: current QKD systems are limited to ~100-200 km without repeaters.
- Hardware requirements: Needs specialized quantum communica-

2. Quantum Resistance:

- **RSA:**
  - Not quantum-resistance.
  - Shor's algorithm efficiently factors large integers on a quantum computers, breaking RSA.
- **Lattice-Based crypto:**
  - currently believed to be resistant to quantum attacks.
  - No known quantum algorithm efficiently solves lattic problems like SVP or LWE.
  - They are Prime candidates for post quantum cryptography.

Security foundations: RSA security depends on the difficulty of factoring integers, a single specific

mathmatical problem.

- Lattic-based schemes:

- security reduces to worst-case hardness of lattic problems, meaning breaking the scheme is as hard as solving the hard instances of certain lattic problems,
- often have stronger theoretical security guarentees,

- NTRU: practical lattic-based encryption. RSA and other classical crypto systems are broken by powerful quantum computers.

Step-1: Designa a custom PRNG in my_prng.py

```
class myPRNG
def __init__(self, seed: int=123456789):
    self.state=seed
    self.a=1664525
    self.c=1013904223
    self.m=2**32
def next(self)-> float:
    self.state=(self.a * self.state + self.c) % self.m
    x=self.state
    x^= (x>>11)
    x^= (x<<7) & 0x9D2C5680
    x^= (x<<15) & 0xEFC60000
    x^= (x>>18)
    return(x % self.m)/self.m
def generate_n(self, n: int):
```

return [self.next() for _ in range(n)]

Step:2 Analyze the PRNG-Uniformity

Dependency:

```
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
from my-prng import MyPRNG
def analyze_prng():
    Prng = MyPRNG(seed=42)
    nums = Prng.generate_n(10000)
    plt.figure(figsize=(10,4))
    sns.histplot(nums, bins=50, kde=True,
    stat="density")
    plt.title("PRNG Output Distribution (uniformity check)")
    plt.ylabel("Density")
```

return [self.next() for _ in range(n)]

Step:2 Analyze the PRNG-Uniformity & Dependency:

```
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
from my-prng import MyPRNG
def analyze_prng():
    Prng = MyPRNG(seed = 42)
    nums = Prng.generate_n(10000)
    plt.figure(figsize = (10, 4))
    sns.histplot(nums, bins = 50, kde = True,
    stat = "density")
    plt.title("PRNG Output Distribution (uniformity check)")
    plt.ylabel("Density")
```

```
plt.savefig("uniformity.png")
plt.show()
plt.figure(figsize=(6,6))
plt.plot(nums[:-1], nums[1:],
alpha=0.5)
plt.title("Lag plot (dependency check")
plt.xlabel("xn")
plt.savefig('dependency.png")
plt.show()
if __name__ == "__main__":
    analyze_prng()
```

Step-3: Create the GitHub Repo structure

```
mkdir my-prng
cd my-prng
touch my_prng.py analyze_prng.py
README.md requirments.txt
```