How Shor's Algorithm can potentially break RSA generated pins on a quantum system :

RSA is a widely used encryption algorithm that secures data online. It's working goes as follows :

1. Key Generation:
   - Choose two large prime numbers p and q.
   - Compute $N = p \times q$ → this is part of the public key.
   - Choose e (public exponent) and compute d (private key) such that:

     $$e \cdot d \equiv 1(\mathrm{mod}(p-1)(q-1))$$

   - Public key = (N, e), Private key = d
2. Encryption:
   - Given message M, the ciphertext is:

     $$C = M^e \bmod N$$

3. Decryption:
   - Given ciphertext C, the message is recovered by:

     $$M = C^d \bmod N$$

RSA relies on the fact that factoring large numbers is hard for classical computers.

Given $N = p \times q$, it's easy to multiply p and q, but very hard to factor N back into p and q, especially when p and q are hundreds of digits long.

Without knowing p and q, you can't compute (p-1)(q-1) and therefore can't compute d.

How Shor's Algorithm Breaks RSA :

Shor's Algorithm can factor N efficiently using a quantum computer.

Here's the connection:

- RSA security : based on factoring being hard
- Shor's Algorithm : factors N in polynomial time

Breakdown:

1. Given public key N, Shor's Algorithm factors it into p and q.
2. Compute $\phi(N) = (p-1)(q-1)$
3. Compute private key d from public exponent e using the modular inverse:

   $$D \equiv e^{-1} \bmod \phi(N)$$

4. Now decrypt any RSA ciphertext intended for that keypair.

Example :

Say :

- p = 7, q = 13 , thus N = 91
- e = 5

From Shor's Algorithm, you get:

- $\phi(91) = (7-1)(13-1) = 6 \times 12 = 72$
- Compute d = 29 such that $5 \times 29 \equiv 1 \mod 72$

Using Shor's Algorithm, we can recover 7 and 13, and compute d, thus breaking RSA.

Code:

```
import numpy as np

from math import gcd

from fractions import Fraction

from random import randint


def euclidGCD(n, m):

   if m == 0:

      return n

   return euclidGCD(m, n % m)


def classical_period_finding(a, N):

   r = 1

   while pow(a, r, N) != 1:

      r += 1

      if r > N:

         return None

   return r


def shors_classical(N):
```

```
    if N % 2 == 0:
        return 2
    attempt = 0
    while True:
        attempt += 1
        a = randint(2, N - 1)
        print(f"Attempt {attempt}: Trying a = {a}")
        if gcd(a, N) != 1:
            print(f"GCD({a}, {N}) = {gcd(a, N)}. Found factor early!")
            return gcd(a, N)

        r = classical_period_finding(a, N)
        if r is None or r % 2 != 0:
            print(f"Invalid period r = {r}")
            continue

        if pow(a, r, N) != 1:
            continue

        plus = gcd(pow(a, r // 2) + 1, N)
        minus = gcd(pow(a, r // 2) - 1, N)

        for factor in [plus, minus]:
            if factor != 1 and factor != N and N % factor == 0:
                print(f"Success! Found non-trivial factor: {factor}")
                return factor

N = 65
factor = shors_classical(N)
print(f"One factor of {N} is {factor}")
```