Factoring with a Quantum Computer: The State of the Art

Seyoon Ragavan (MIT)

Based on joint works with Gregory D. Kahanamoku-Meyer*, Vinod Vaikuntanathan*, and Katherine Van Kirk†

*MIT, †Harvard

Integer Factorisation

Given an n-bit integer $N < 2^n$, find its prime factorisation in $\mathsf{poly}(n)$ time.

Classical Factoring: A Crash Course

Given an n-bit integer $N < 2^n$, find its prime factorisation in poly(n) time.

Factoring general integers

- Brute force: $\exp(O(n))$

Classical Factoring: A Crash Course

Given an n-bit integer $N < 2^n$, find its prime factorisation in poly(n) time.

Factoring general integers

- Brute force: $\exp(O(n))$
- Quadratic sieve (many works from Fermat to Pomerance '82): $\exp(\tilde{O}(n^{1/2}))$

Classical Factoring: A Crash Course

Given an n-bit integer $N < 2^n$, find its prime factorisation in poly(n) time.

Factoring general integers

- Brute force: $\exp(O(n))$
- Quadratic sieve (many works from Fermat to Pomerance '82): $\exp(\tilde{O}(n^{1/2}))$
- General number field sieve (Pollard '88; Buhler-Lenstra-Pomerance '93): $\exp(\tilde{O}(n^{1/3}))$

Classical Factoring: A Crash Course

Given an n-bit integer $N < 2^n$, find its prime factorisation in poly(n) time.

Factoring general integers

- Brute force: $\exp(O(n))$
- Quadratic sieve (many works from Fermat to Pomerance '82): $\exp(\tilde{O}(n^{1/2}))$
- General number field sieve (Pollard '88; Buhler-Lenstra-Pomerance '93): $\exp(\tilde{O}(n^{1/3}))$

Factoring special-form integers

Classical Factoring: A Crash Course

Given an n-bit integer $N < 2^n$, find its prime factorisation in poly(n) time.

Factoring general integers

- Brute force: $\exp(O(n))$
- Quadratic sieve (many works from Fermat to Pomerance '82): $\exp(\tilde{O}(n^{1/2}))$
- General number field sieve (Pollard '88; Buhler-Lenstra-Pomerance '93): $\exp(\tilde{O}(n^{1/3}))$

Factoring special-form integers

- Lenstra ECM ('87): $\exp\left(\tilde{O}((\log P)^{1/2})\right)$ where P is a factor of N

Classical Factoring: A Crash Course

Given an n-bit integer $N < 2^n$, find its prime factorisation in $\mathsf{poly}(n)$ time.

Factoring general integers

- Brute force: $\exp(O(n))$
- Quadratic sieve (many works from Fermat to Pomerance '82): $\exp(\tilde{O}(n^{1/2}))$
- General number field sieve (Pollard '88; Buhler-Lenstra-Pomerance '93): $\exp(\tilde{O}(n^{1/3}))$

Factoring special-form integers

- Lenstra ECM ('87): $\exp\left(\tilde{O}((\log P)^{1/2})\right)$ where P is a factor of N
- Boneh, Durfee, Howgrave-Graham ('99): if $N = P^r Q$ and $r \geq \log P$, can factor in $\mathsf{poly}(n)$ time

Classical Factoring: A Crash Course

Given an n-bit integer $N < 2^n$, find its prime factorisation in $\mathsf{poly}(n)$ time.

Factoring general integers

- Brute force: $\exp(O(n))$
- Quadratic sieve (many works from Fermat to Pomerance '82): $\exp(\tilde{O}(n^{1/2}))$
- General number field sieve (Pollard '88; Buhler-Lenstra-Pomerance '93): $\exp(\tilde{O}(n^{1/3}))$

Factoring special-form integers

- Lenstra ECM ('87): $\exp\left(\tilde{O}((\log P)^{1/2})\right)$ where P is a factor of N
- Boneh, Durfee, Howgrave-Graham ('99): if $N = P^r Q$ and $r \geq \log P$, can factor in $\mathsf{poly}(n)$ time

For more: see Pomerance's survey "A Tale of Two Sieves"!

Integer Factorisation

Given an n-bit integer $N < 2^n$, find its prime factorisation in poly(n) time.

- Fastest classical algorithm for general N: $\exp\left(\tilde{O}(n^{1/3})\right)$ time

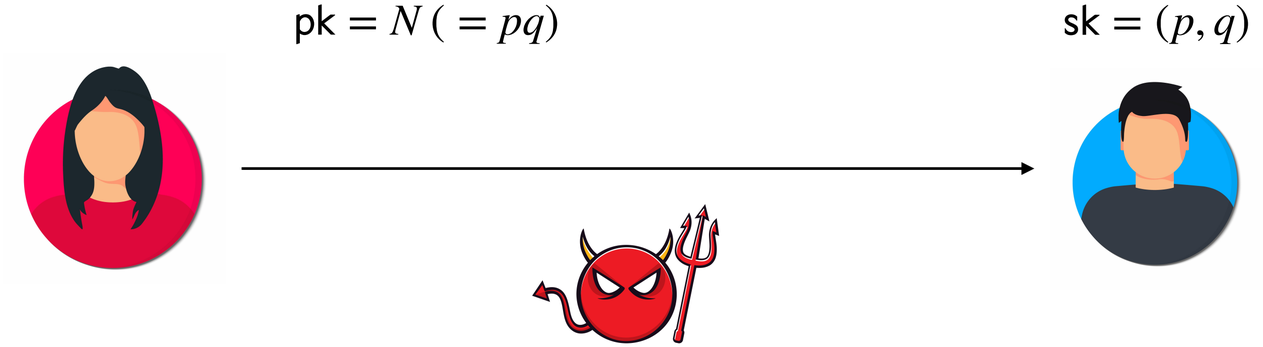
Integer Factorisation

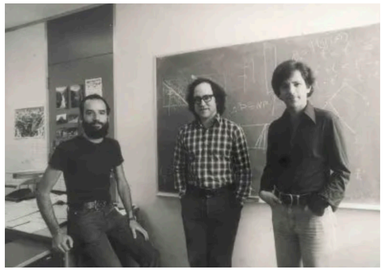
Given an n-bit integer $N < 2^n$, find its prime factorisation in poly(n) time.

- Fastest classical algorithm for general N: $\exp\left(\tilde{O}(n^{1/3})\right)$ time
- **Quantum algorithms:** poly(n) **time!** (Shor 1994)

Implication 1: Breaking Cryptography

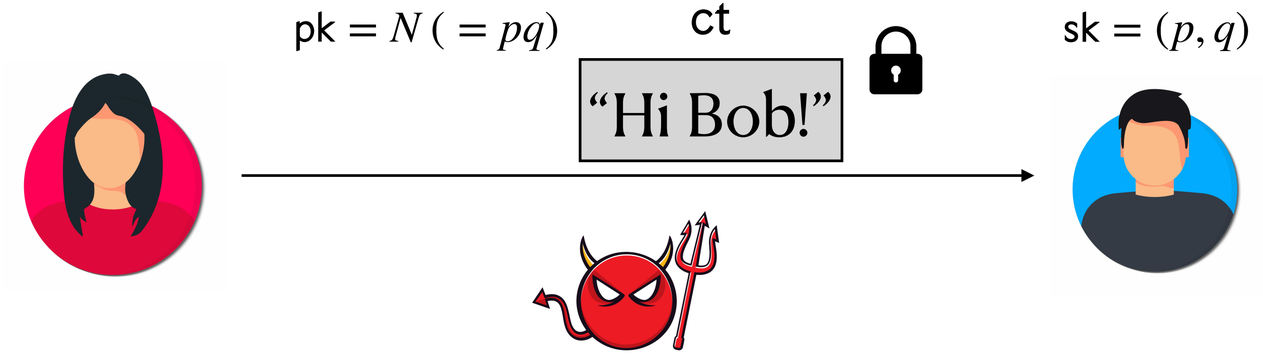
- RSA public-key cryptography:

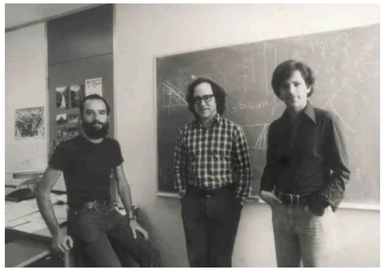




Implication 1: Breaking Cryptography

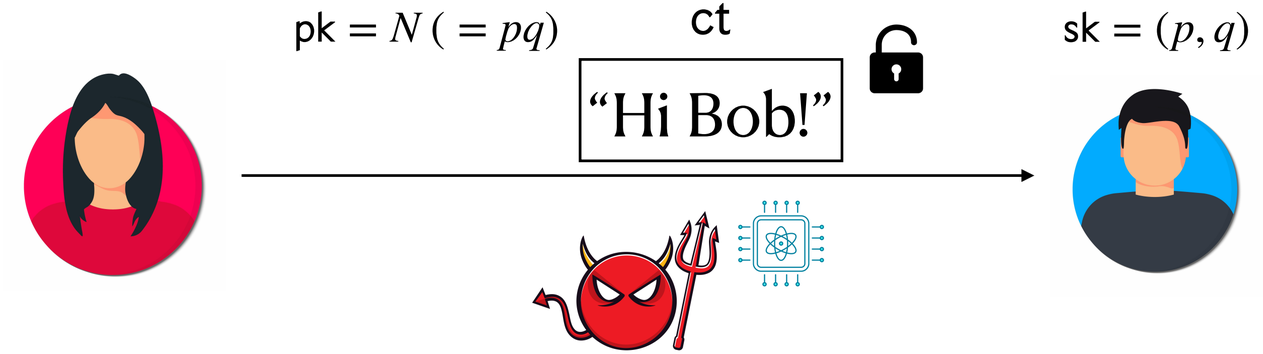
- RSA public-key cryptography:

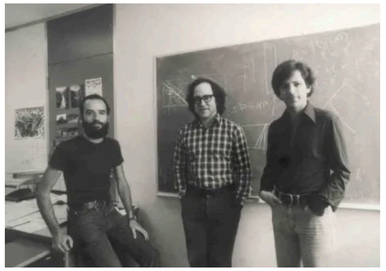




Implication 1: Breaking Cryptography

- RSA public-key cryptography:



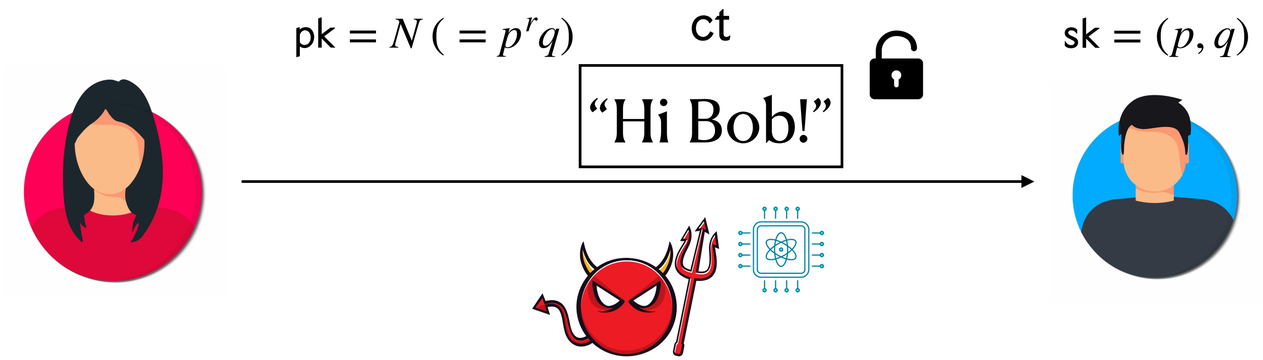


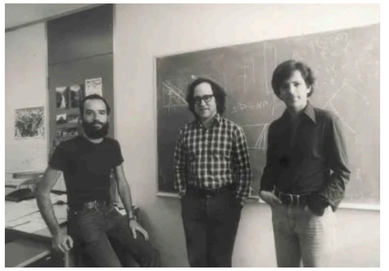
- **Completely broken if the eavesdropper has a large quantum computer (learn Bob's secret key by factoring $N = pq$)**

Implication 1: Breaking Cryptography

- Okamoto-Uchiyama ($r = 2$) or Takagi ($r > 2$) cryptography:

Goal: faster decryption than RSA



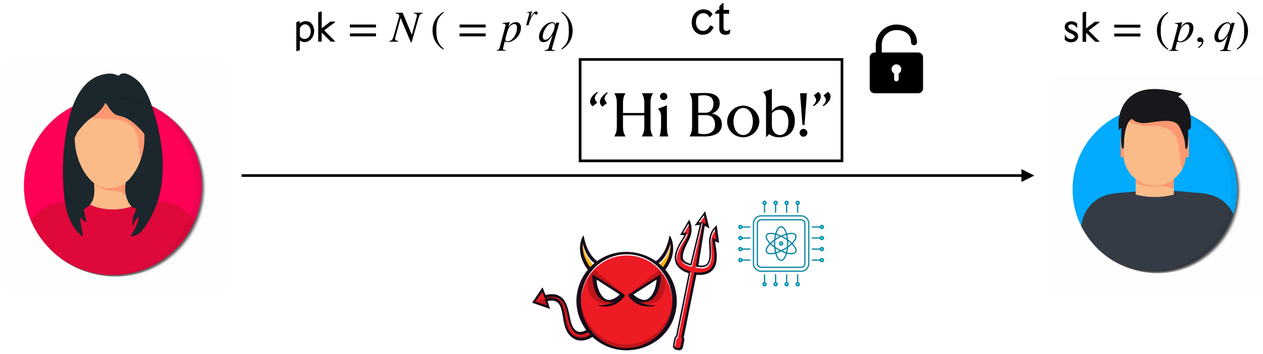


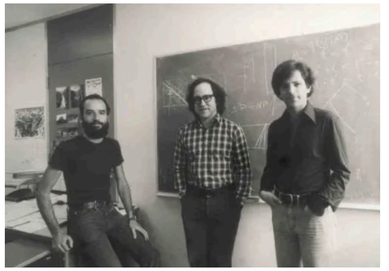
- **Completely broken if the eavesdropper has a large quantum computer (learn Bob's secret key by factoring $N = p^r q$)**

Implication 1: Breaking Cryptography

- Okamoto-Uchiyama ($r = 2$) or Takagi ($r > 2$) cryptography:

Goal: faster decryption than RSA

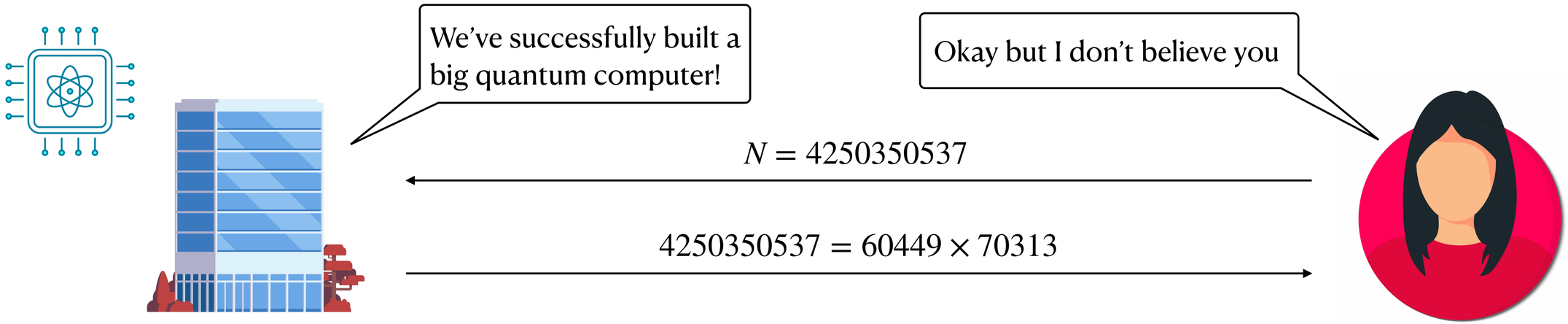




- **Completely broken if the eavesdropper has a large quantum computer (learn Bob's secret key by factoring $N = p^r q$)**

Coming up: even better quantum circuits for factoring $p^r q$ ($r > 1$)

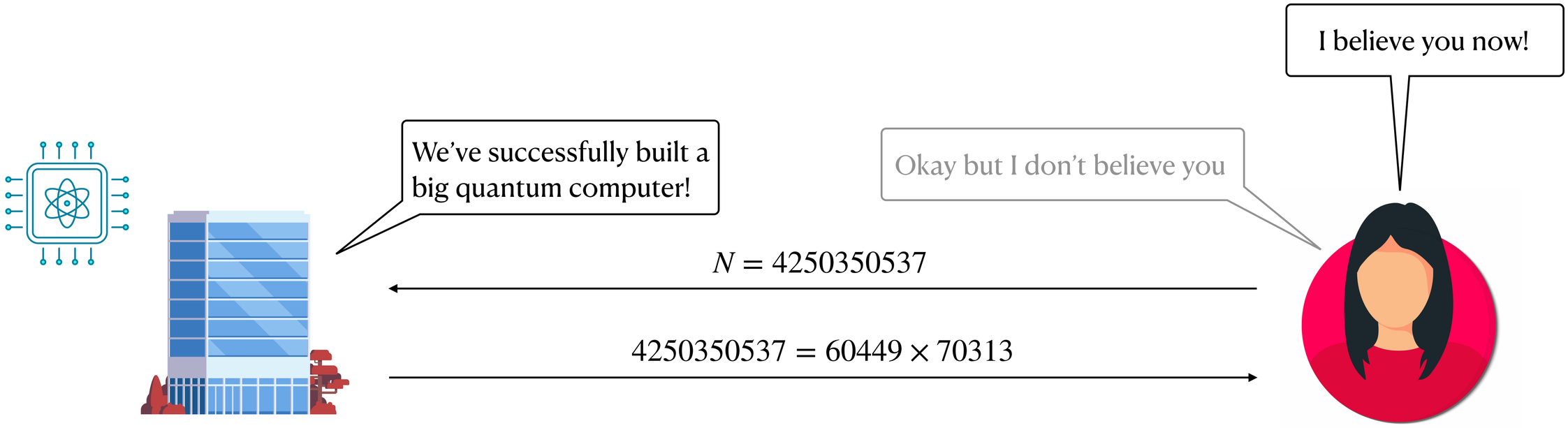
Implication 2: Proofs of Quantumness



Q: How can XYZABC Labs convince Alice that they really do have a large quantum computer?

One answer: By factoring a large integer of Alice's choice!

Implication 2: Proofs of Quantumness



Q: How can XYZABC Labs convince Alice that they really do have a large quantum computer?

One answer: By factoring a large integer of Alice's choice!

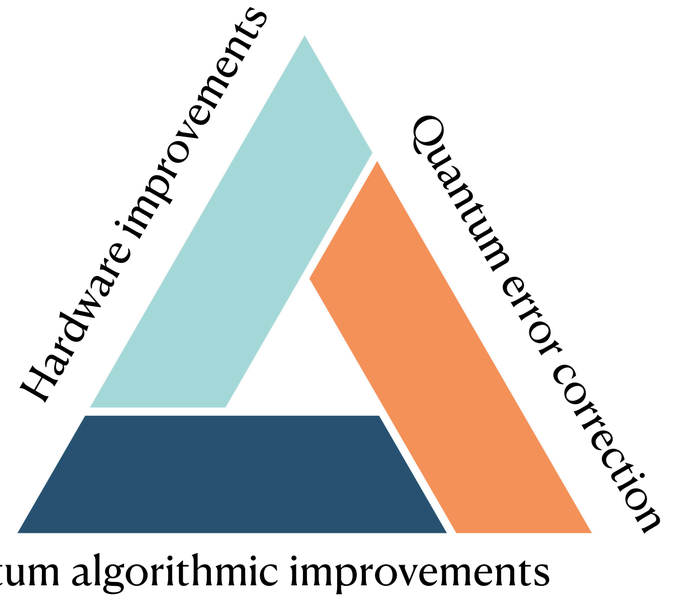
The Current State of Affairs

- Known for 30 years: if we had a large-scale quantum computer, we could verifiably demonstrate quantum advantage (by factoring large integers)
- **Slight catch: no-one has managed to build a large-scale quantum computer yet**

The Current State of Affairs

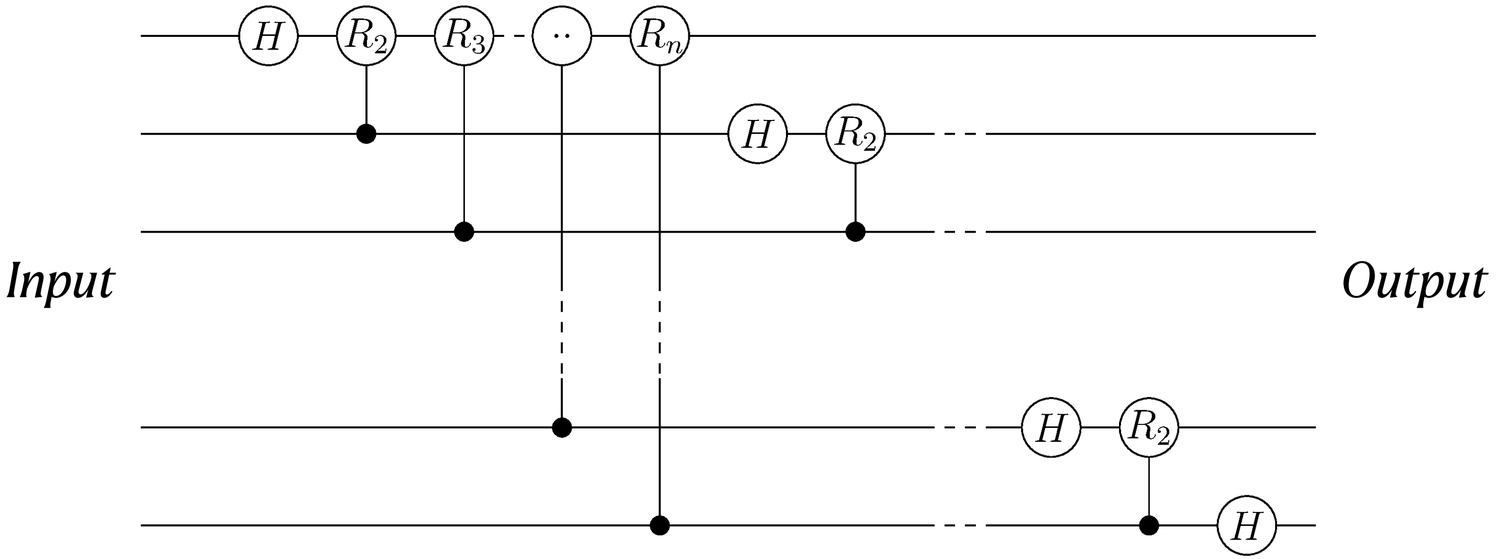
- Known for 30 years: if we had a large-scale quantum computer, we could verifiably demonstrate quantum advantage (by factoring large integers)
- **Slight catch: no-one has managed to build a large-scale quantum computer yet**

"Our qubits are constantly trying to fall apart… It's like you're trying to write in the sand and the wind is blowing it away."



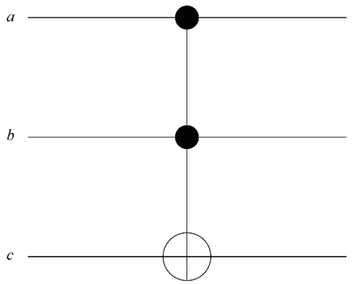
Efficiency Metrics for Quantum Circuits

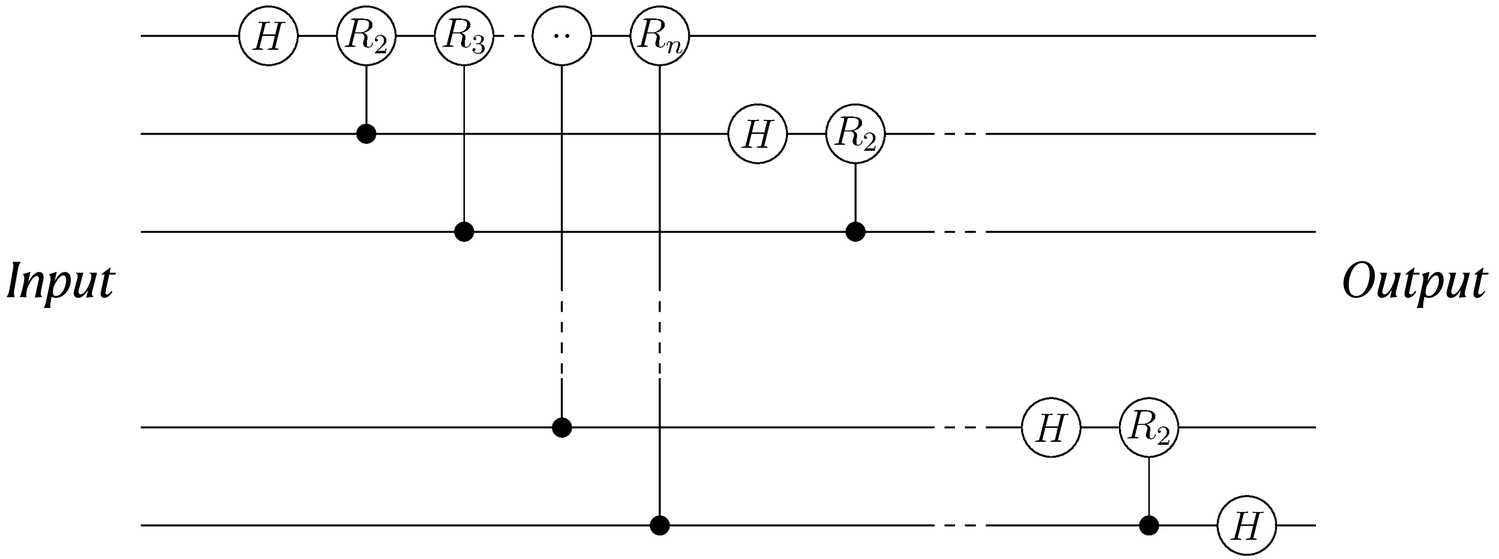
- Similar to a classical circuit, but uses reversible logic gates on a fixed number of wires



Efficiency Metrics for Quantum Circuits

- Similar to a classical circuit, but uses reversible logic gates on a fixed number of wires

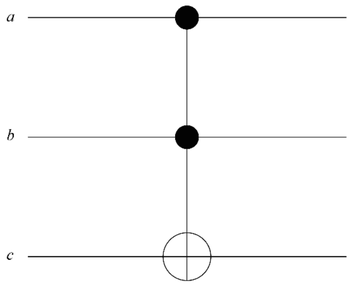


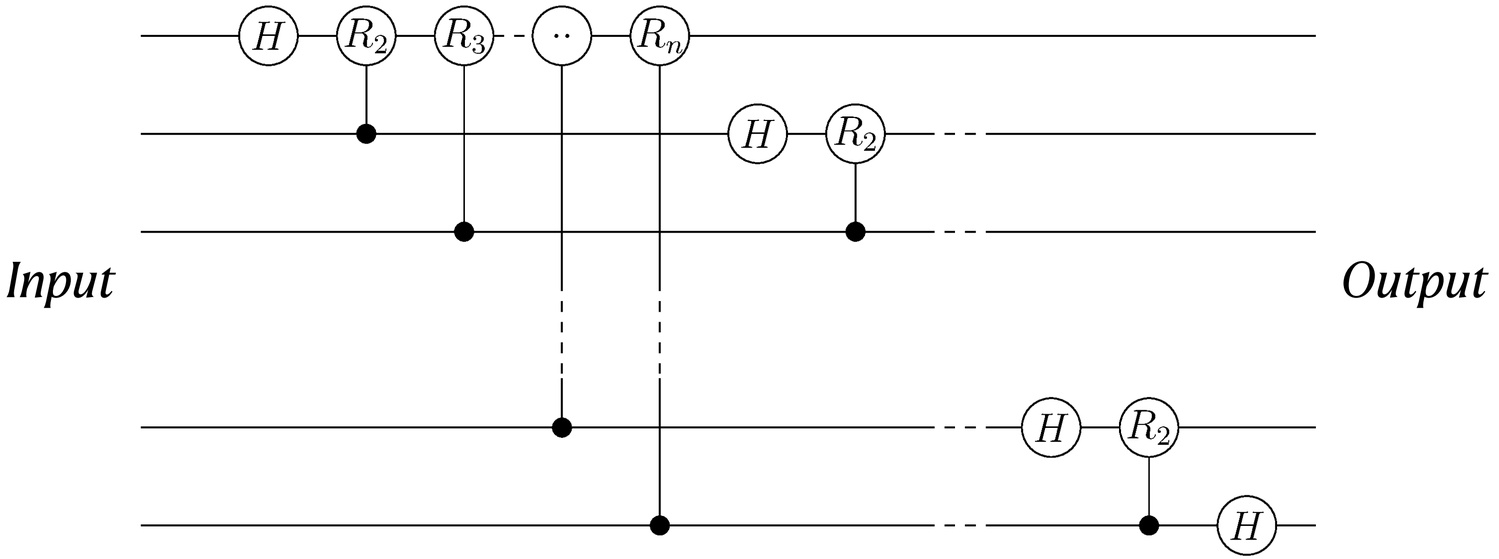


Efficiency Metrics for Quantum Circuits

- Similar to a classical circuit, but uses reversible logic gates on a fixed number of wires

Gates: number of elementary operations (circles)



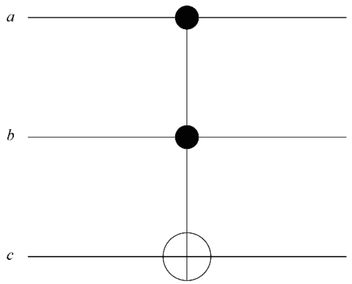


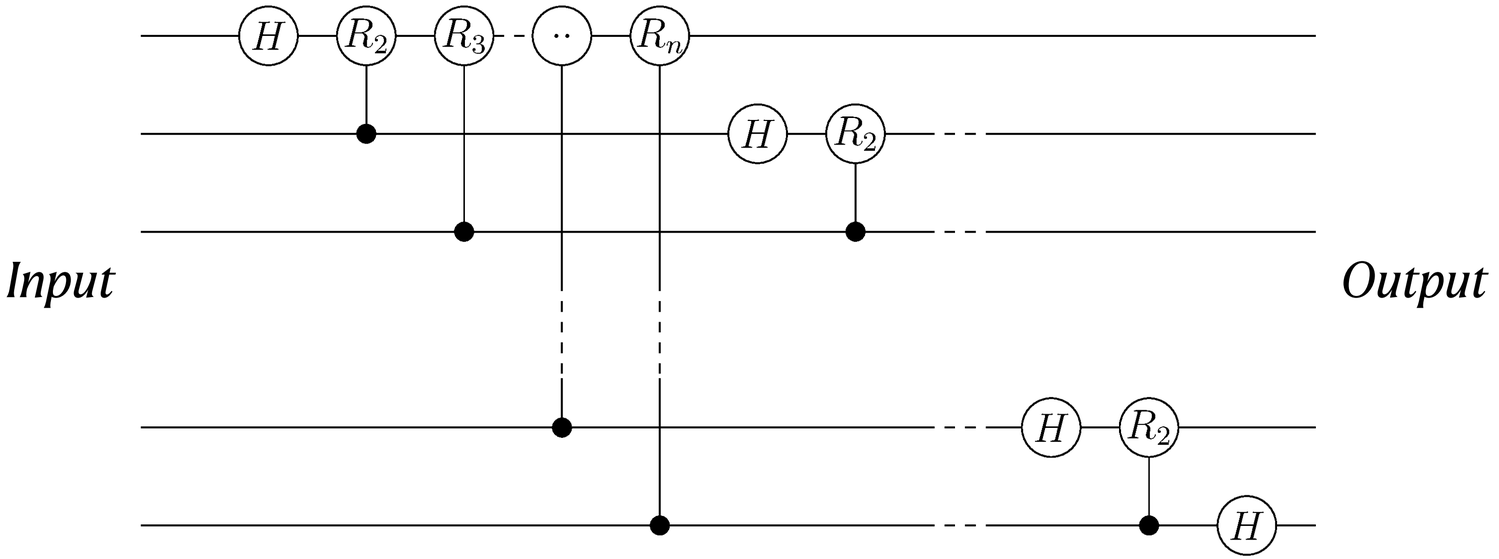
Efficiency Metrics for Quantum Circuits

- Similar to a classical circuit, but uses reversible logic gates on a fixed number of wires

Gates: number of elementary operations (circles)

Space: number of wires (lines)





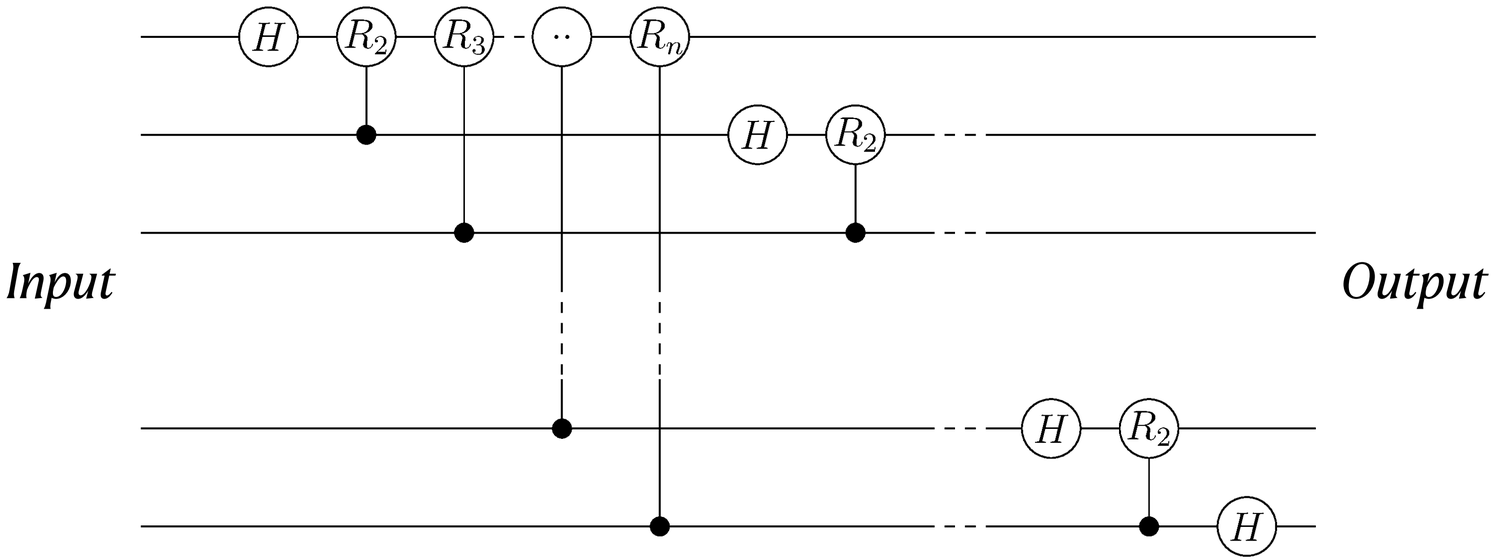
Efficiency Metrics for Quantum Circuits

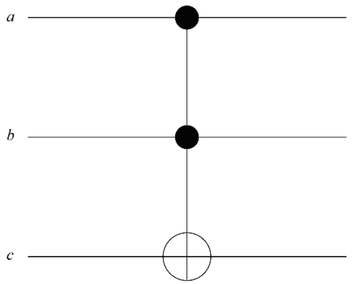
- Similar to a classical circuit, but uses reversible logic gates on a fixed number of wires

Gates: number of elementary operations (circles)

Space: number of wires (lines)

Depth: number of time steps (if gates acting on disjoint wires can be run in parallel)





Proofs of Quantumness from Factoring

Authors	Types of inputs	Gates	Space	Depth
Shor (1994)	Any	$\tilde{O}(n^2)$	$\tilde{O}(n)$	$\tilde{O}(n)$

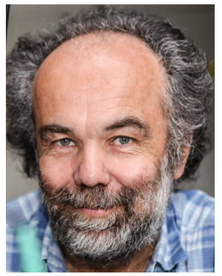
n is the number of bits in the input N

$\tilde{O}(\cdot)$ hides constant and $\text{poly}(\log n)$ factors

All results in this talk are using fast integer multiplication (multiply n-bit integers in $\tilde{O}(n)$ time)

Proofs of Quantumness from Factoring

Authors	Types of inputs	Gates	Space	Depth
Shor (1994)	Any	$\tilde{O}(n^2)$	$\tilde{O}(n)$	$\tilde{O}(n)$
LPDS (2012)	$N = P^2Q$	$\tilde{O}(n)$	$\tilde{O}(n)$	$\tilde{O}(n)$

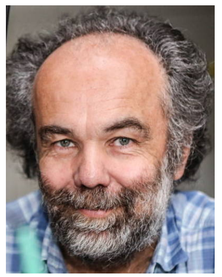


Proofs of Quantumness from Factoring

Authors	Types of inputs	Gates	Space	Depth
Shor (1994)	Any	$\tilde{O}(n^2)$	$\tilde{O}(n)$	$\tilde{O}(n)$
LPDS (2012)	$N = P^2Q$	$\tilde{O}(n)$	$\tilde{O}(n)$	$\tilde{O}(n)$

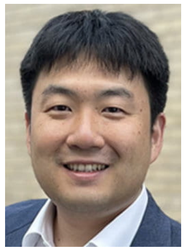
Any input: would break RSA cryptography, and suffice as a proof of quantumness

$N = P^2Q$: only suffices as a proof of quantumness



Proofs of Quantumness from Factoring

Authors	Types of inputs	Gates	Space	Depth
Shor (1994)	Any	$\tilde{O}(n^2)$	$\tilde{O}(n)$	$\tilde{O}(n)$
LPDS (2012)	$N = P^2Q$	$\tilde{O}(n)$	$\tilde{O}(n)$	$\tilde{O}(n)$
KCVY (2021)	N/A	$\tilde{O}(n)$	$\tilde{O}(n)$	$\tilde{O}(1)$

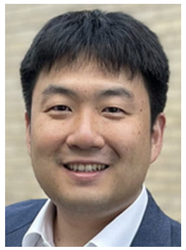



Proofs of Quantumness from Factoring

Authors	Types of inputs	Gates	Space	Depth
Shor (1994)	Any	$\tilde{O}(n^2)$	$\tilde{O}(n)$	$\tilde{O}(n)$
LPDS (2012)	$N = P^2Q$	$\tilde{O}(n)$	$\tilde{O}(n)$	$\tilde{O}(n)$
KCVY (2021)	N/A	$\tilde{O}(n)$	$\tilde{O}(n)$	$\tilde{O}(1)$

Note: KCVY is not a factoring algorithm!

- 3-round interactive proof of quantumness (building on previous work by BCMVV18) assuming factoring is classically hard

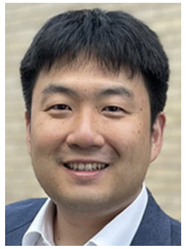



Proofs of Quantumness from Factoring

Authors	Types of inputs	Gates	Space	Depth
Shor (1994)	Any	$\tilde{O}(n^2)$	$\tilde{O}(n)$	$\tilde{O}(n)$
LPDS (2012)	$N = P^2Q$	$\tilde{O}(n)$	$\tilde{O}(n)$	$\tilde{O}(n)$
KCVY (2021)	N/A	$\tilde{O}(n)$	$\tilde{O}(n)$	$\tilde{O}(1)$

Note: KCVY is not a factoring algorithm!

- 3-round interactive proof of quantumness (building on previous work by BCMVV18) assuming factoring is classically hard
- Pro: easier than actually factoring

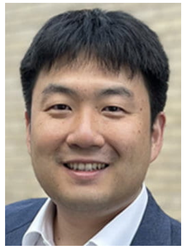



Proofs of Quantumness from Factoring

Authors	Types of inputs	Gates	Space	Depth
Shor (1994)	Any	$\tilde{O}(n^2)$	$\tilde{O}(n)$	$\tilde{O}(n)$
LPDS (2012)	$N = P^2Q$	$\tilde{O}(n)$	$\tilde{O}(n)$	$\tilde{O}(n)$
KCVY (2021)	N/A	$\tilde{O}(n)$	$\tilde{O}(n)$	$\tilde{O}(1)$

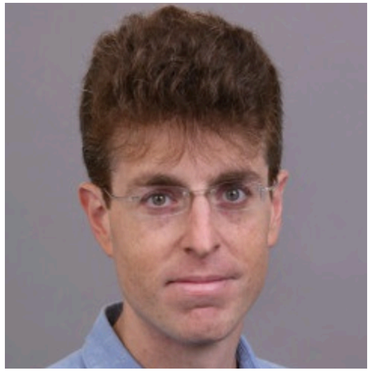
Note: KCVY is not a factoring algorithm!

- 3-round interactive proof of quantumness (building on previous work by BCMVV18) assuming factoring is classically hard
- Pro: easier than actually factoring
- Con: quantum prover needs to store state between rounds

Proofs of Quantumness from Factoring

Authors	Types of inputs	Gates	Space	Depth
Shor (1994)	Any	$\tilde{O}(n^2)$	$\tilde{O}(n)$	$\tilde{O}(n)$
LPDS (2012)	$N = P^2Q$	$\tilde{O}(n)$	$\tilde{O}(n)$	$\tilde{O}(n)$
KCVY (2021)	N/A	$\tilde{O}(n)$	$\tilde{O}(n)$	$\tilde{O}(1)$
Regev (2023)	Any	$\tilde{O}(n^{1.5})$	$O(n^{1.5})$	$\tilde{O}(n^{0.5})$



Proofs of Quantumness from Factoring

Authors	Types of inputs	Gates	Space	Depth
Shor (1994)	Any	$\tilde{O}(n^2)$	$\tilde{O}(n)$	$\tilde{O}(n)$
LPDS (2012)	$N = P^2Q$	$\tilde{O}(n)$	$\tilde{O}(n)$	$\tilde{O}(n)$
KCVY (2021)	N/A	$\tilde{O}(n)$	$\tilde{O}(n)$	$\tilde{O}(1)$
Regev (2023)	Any	$\tilde{O}(n^{1.5})$	$O(n^{1.5})$	$\tilde{O}(n^{0.5})$
RV (2024)	Any	$\tilde{O}(n^{1.5})$	$\tilde{O}(n)$	$\tilde{O}(n^{0.5})$

Result 1: optimising Regev to match the space complexity of Shor

Proofs of Quantumness from Factoring

Authors	Types of inputs	Gates	Space	Depth
Shor (1994)	Any	$\tilde{O}(n^2)$	$\tilde{O}(n)$	$\tilde{O}(n)$
LPDS (2012)	$N = P^2Q$	$\tilde{O}(n)$	$\tilde{O}(n)$	$\tilde{O}(n)$
KCVY (2021)	N/A	$\tilde{O}(n)$	$\tilde{O}(n)$	$\tilde{O}(1)$
Regev (2023)	Any	$\tilde{O}(n^{1.5})$	$O(n^{1.5})$	$\tilde{O}(n^{0.5})$
RV (2024)	Any	$\tilde{O}(n^{1.5})$	$\tilde{O}(n)$	$\tilde{O}(n^{0.5})$
K**R**VV (2024)	$N = P^2Q$ $(Q < 2^m)$	$\tilde{O}(n)$	$\tilde{O}(m)$	$\tilde{O}(n/m + m)$

Result 2: LPDS with space and depth proportional to $\log Q$ rather than $\log N$




Setting Parameters for Factoring P^2Q

How should we set $m = \log Q$?

Setting Parameters for Factoring P^2Q

How should we set $m = \log Q$?

- Balancing act:
  - If Q is too large: our quantum circuit is no better than the LPDS12 circuit

Setting Parameters for Factoring P^2Q

How should we set $m = \log Q$?

- Balancing act:
  - If Q is too large: our quantum circuit is no better than the LPDS12 circuit
  - If Q is too small: classical algorithms could exploit this structure to run faster than general-purpose classical factoring algorithms

Setting Parameters for Factoring P^2Q

How should we set $m = \log Q$?

- Balancing act:
  - If Q is too large: our quantum circuit is no better than the LPDS12 circuit
  - If Q is too small: classical algorithms could exploit this structure to run faster than general-purpose classical factoring algorithms
    - General NFS: $\exp(\tilde{O}(n^{1/3}))$
    - Lenstra elliptic curve method/Mulder '24: $\exp(\tilde{O}(m^{1/2}))$

Setting Parameters for Factoring P^2Q

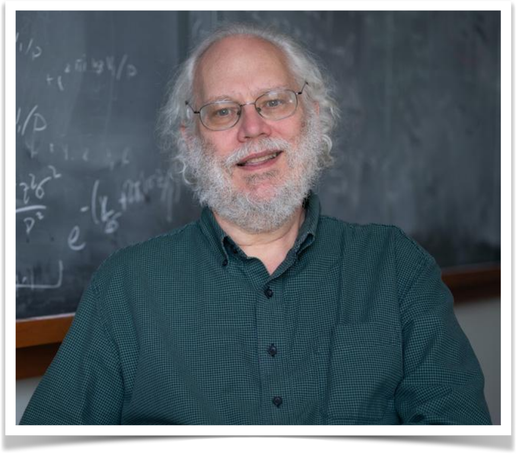
How should we set $m = \log Q$?

- Balancing act:
  - If Q is too large: our quantum circuit is no better than the LPDS12 circuit
  - If Q is too small: classical algorithms could exploit this structure to run faster than general-purpose classical factoring algorithms
    - General NFS: $\exp(\tilde{O}(n^{1/3}))$
    - Lenstra elliptic curve method/Mulder '24: $\exp(\tilde{O}(m^{1/2}))$
- Sweet spot: $m = \tilde{O}(n^{2/3}) \rightarrow$ gates $\tilde{O}(n)$, space $\tilde{O}(n^{2/3})$, depth $\tilde{O}(n^{2/3})$

Proofs of Quantumness from Factoring

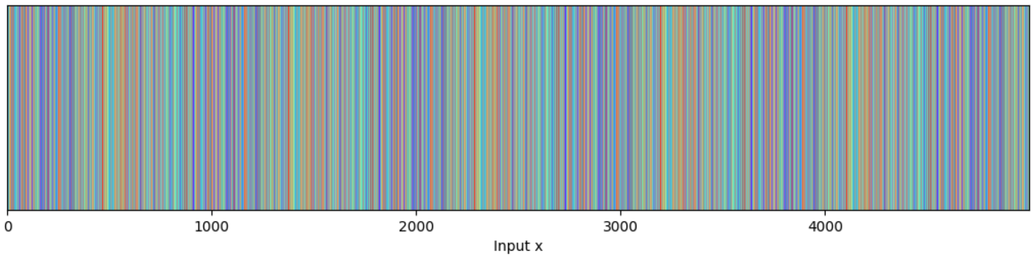
Authors	Types of inputs	Gates	Space	Depth
Shor (1994)	Any	$\tilde{O}(n^2)$	$\tilde{O}(n)$	$\tilde{O}(n)$
LPDS (2012)	$N = P^2Q$	$\tilde{O}(n)$	$\tilde{O}(n)$	$\tilde{O}(n)$
KCVY (2021)	N/A	$\tilde{O}(n)$	$\tilde{O}(n)$	$\tilde{O}(1)$
Regev (2023)	Any	$\tilde{O}(n^{1.5})$	$O(n^{1.5})$	$\tilde{O}(n^{0.5})$
RV (2024)	Any	$\tilde{O}(n^{1.5})$	$\tilde{O}(n)$	$\tilde{O}(n^{0.5})$
K**R**VV (2024)	$N = P^2Q$ $(Q < 2^{n^{2/3}})$	$\tilde{O}(n)$	$\tilde{O}(n^{2/3})$	$\tilde{O}(n^{2/3})$

Shor's Algorithm: A Sketch



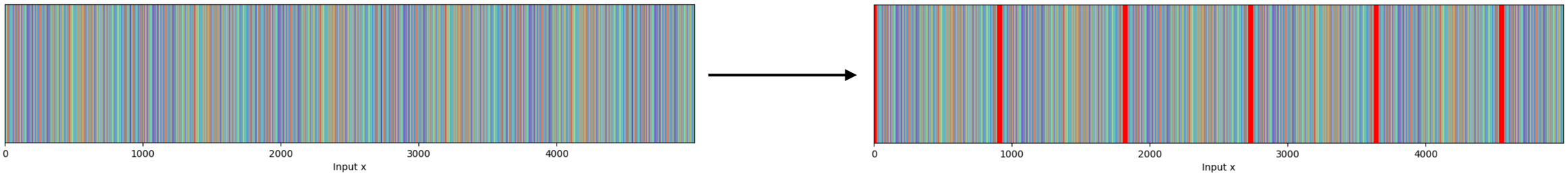
Preliminary: Quantum Period Finding

- Strictly periodic function $f : \mathbb{Z} \to \mathbb{Z}$ with unknown period T
  - $x \equiv y \pmod{T} \Leftrightarrow f(x) = f(y)$



Preliminary: Quantum Period Finding

- Strictly periodic function $f : \mathbb{Z} \to \mathbb{Z}$ with unknown period T
  - $x \equiv y \pmod{T} \Leftrightarrow f(x) = f(y)$
- Informal theorem statement: can quantumly recover a **uniformly random** multiple of $1/T$ (and hence T itself) using essentially only the gates/space needed to compute $f(x)$ for $|x| \leq \text{poly}(T)$



Shor overview: finding square roots of 1

- Goal: find $z \not\equiv \pm 1 \mod N$ such that $z^2 \equiv 1 \mod N$
  - N divides $z^2 - 1 = (z-1)(z+1)$ but not either factor individually
  - Hence $\gcd(z-1, N)$ is a nontrivial divisor of N

Shor overview: reduction to period-finding

Let's focus on $N = 3763$. *(* $= 53 \times 71$*, but we don't know this yet)*

Shor overview: reduction to period-finding

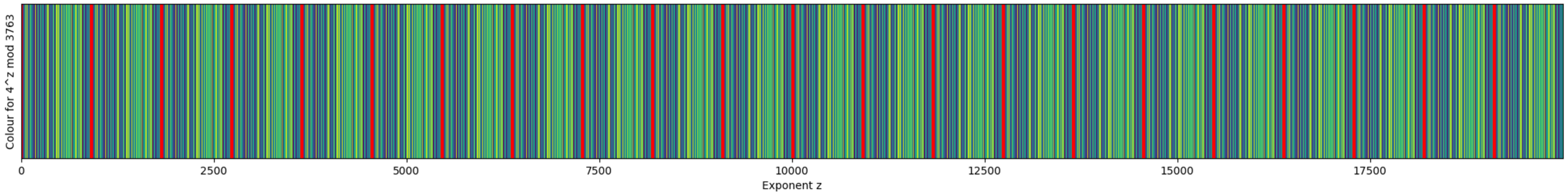
Let's focus on $N = 3763$**.** *(* $= 53 \times 71$*, but we don't know this yet)*

- Choose a random square as a base e.g. 4
- Powers of 4 mod 3763: $4^0 = 1, 4^1 = 4, 4^2 = 16, 64, 256, 1024, 333, 1332, \ldots$

Shor overview: reduction to period-finding

Let's focus on $N = 3763$**.** *(* $= 53 \times 71$*, but we don't know this yet)*

- Choose a random square as a base e.g. 4
- Powers of 4 mod 3763: $4^0 = 1, 4^1 = 4, 4^2 = 16, 64, 256, 1024, 333, 1332, \ldots$
  - Eventually repeats
  - Let $r < N$ be the period i.e. the first positive index where $4^r \equiv 1 \mod 3763$



Shor overview: reduction to period-finding

Let's focus on $N = 3763$. *(* $= 53 \times 71$*, but we don't know this yet)*

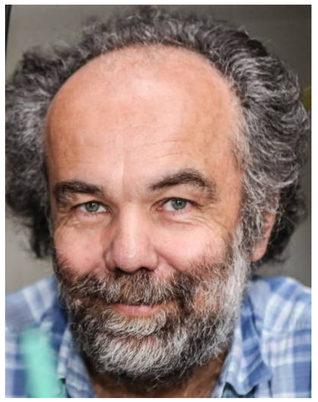
- Choose a random square as a base e.g. 4
- Powers of 4 mod 3763: $4^0 = 1, 4^1 = 4, 4^2 = 16, 64, 256, 1024, 333, 1332, \ldots$
  - Eventually repeats
  - Let $r < N$ be the period i.e. the first positive index where $4^r \equiv 1 \bmod 3763$
- But $4^r = (2^r)^2$, so 2^r is a square root of 1 mod 3763
  - With some luck: $2^r \not\equiv \pm 1 \bmod 3763$ so this would give us a factor!

Shor overview: reduction to period-finding

Let's focus on $N = 3763$. *(* $= 53 \times 71$*, but we don't know this yet)*

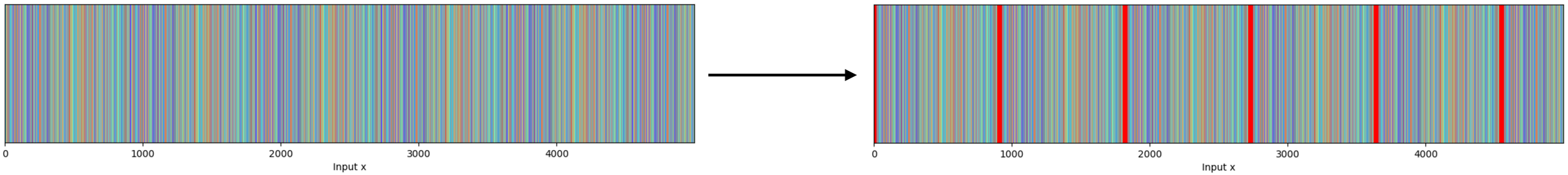
- Choose a random square as a base e.g. 4
- Powers of 4 mod 3763: $4^0 = 1, 4^1 = 4, 4^2 = 16, 64, 256, 1024, 333, 1332, \ldots$
  - Eventually repeats
  - Let $r < N$ be the period i.e. the first positive index where $4^r \equiv 1 \bmod 3763$
- But $4^r = (2^r)^2$, so 2^r is a square root of 1 mod 3763
  - With some luck: $2^r \not\equiv \pm 1 \bmod 3763$ so this would give us a factor!
  - *"Luck" is with respect to the randomly chosen base*

Factoring P^2Q with LPDS12: A Sketch



Refresher: Quantum Period Finding

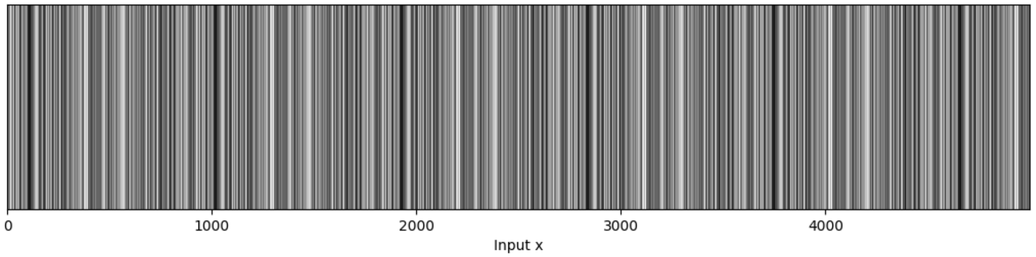
- Strictly periodic function $f : \mathbb{Z} \to \mathbb{Z}$ with unknown period T
  - $x \equiv y \pmod{T} \Leftrightarrow f(x) = f(y)$
- Informal theorem statement: can quantumly recover a **uniformly random** multiple of $1/T$ (and hence T itself) using essentially only the gates/space needed to compute $f(x)$ for $|x| \leq \text{poly}(T)$



Preliminary: General Quantum Period Finding

Hales-Hallgren '98, May-Schlieper '22

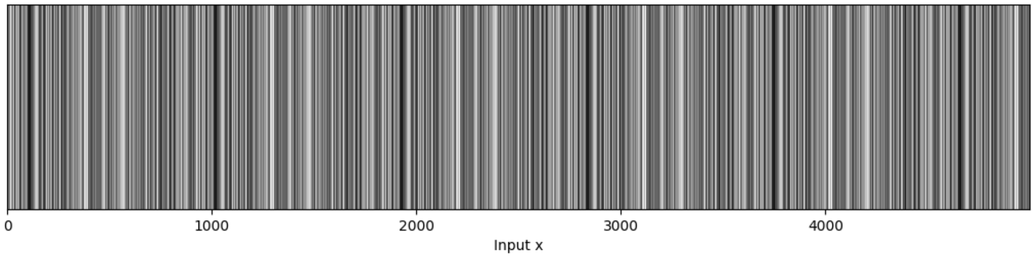
- ~~Strictly~~ periodic function $f : \mathbb{Z} \to \{-1,1\}$ with unknown period T
  - $x \equiv y \pmod{T} \Rightarrow f(x) = f(y)$



Preliminary: General Quantum Period Finding

Hales-Hallgren '98, May-Schlieper '22

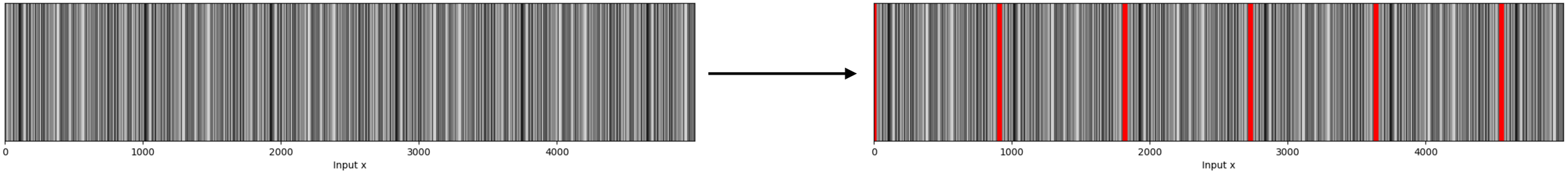
- ~~Strictly~~ periodic function $f : \mathbb{Z} \to \{-1,1\}$ with unknown period T
  - $x \equiv y \pmod{T} \Rightarrow f(x) = f(y)$
- Linearity of the Fourier transform → the same algorithm still outputs a **(not necessarily uniform) random** multiple of $1/T$



Preliminary: General Quantum Period Finding

Hales-Hallgren '98, May-Schlieper '22

- ~~Strictly~~ periodic function $f : \mathbb{Z} \to \{-1,1\}$ with unknown period T
  - $x \equiv y \pmod{T} \Rightarrow f(x) = f(y)$
- Linearity of the Fourier transform → the same algorithm still outputs a **(not necessarily uniform) random** multiple of $1/T$
- Informal theorem statement: for "reasonable" f, this is still sufficient to recover T



Preliminary: The Legendre Symbol

- a is a *quadratic residue* modulo an odd prime p if there exists integer x such that $a \equiv x^2 \pmod{p}$

Preliminary: The Legendre Symbol

- a is a *quadratic residue* modulo an odd prime p if there exists integer x such that $a \equiv x^2 \pmod{p}$
- Legendre symbol essentially indicates whether this is the case:

$$\left(\frac{a}{p}\right) = \begin{cases} 1, \text{ if } a \text{ is a nonzero quadratic residue modulo } p; \text{ and} \\ -1, \text{ if } a \text{ is not a quadratic residue modulo } p; \text{ and} \\ 0, \text{ if } a \text{ divisible by } p\,. \end{cases}$$

Preliminary: The Jacobi Symbol

- Essentially generalises the Legendre symbol to odd composite moduli

Preliminary: The Jacobi Symbol

- Essentially generalises the Legendre symbol to odd composite moduli
- For $N = p_1^{\alpha_1} \ldots p_r^{\alpha_r}$, define:

$$\left(\frac{a}{N}\right) = \left(\frac{a}{p_1}\right)^{\alpha_1} \left(\frac{a}{p_2}\right)^{\alpha_2} \ldots \left(\frac{a}{p_r}\right)^{\alpha_r}$$

Preliminary: The Jacobi Symbol

- Essentially generalises the Legendre symbol to odd composite moduli
- For $N = p_1^{\alpha_1} \ldots p_r^{\alpha_r}$, define:

$$\left(\frac{a}{N}\right) = \left(\frac{a}{p_1}\right)^{\alpha_1} \left(\frac{a}{p_2}\right)^{\alpha_2} \ldots \left(\frac{a}{p_r}\right)^{\alpha_r}$$

- Note for intuition: the quadratic residue characterisation does **not** carry over from the Legendre symbol

Preliminary: The Jacobi Symbol

- Essentially generalises the Legendre symbol to odd composite moduli
- For $N = p_1^{\alpha_1} \ldots p_r^{\alpha_r}$, define:

$$\left(\frac{a}{N}\right) = \left(\frac{a}{p_1}\right)^{\alpha_1} \left(\frac{a}{p_2}\right)^{\alpha_2} \ldots \left(\frac{a}{p_r}\right)^{\alpha_r}$$

- Note for intuition: the quadratic residue characterisation does **not** carry over from the Legendre symbol
  - Could have $\left(\frac{a}{N}\right) = 1$ without a being a quadratic residue modulo N

Preliminary: The Jacobi Symbol

- Essentially generalises the Legendre symbol to odd composite moduli
- For $N = p_1^{\alpha_1} \ldots p_r^{\alpha_r}$, define:

$$\left(\frac{a}{N}\right) = \left(\frac{a}{p_1}\right)^{\alpha_1} \left(\frac{a}{p_2}\right)^{\alpha_2} \ldots \left(\frac{a}{p_r}\right)^{\alpha_r}$$

- Useful property: $a \equiv b \pmod{N} \Rightarrow \left(\frac{a}{N}\right) = \left(\frac{b}{N}\right)$

Preliminary: The Jacobi Symbol

- Essentially generalises the Legendre symbol to odd composite moduli
- For $N = p_1^{\alpha_1} \ldots p_r^{\alpha_r}$, define:

$$\left(\frac{a}{N}\right) = \left(\frac{a}{p_1}\right)^{\alpha_1} \left(\frac{a}{p_2}\right)^{\alpha_2} \ldots \left(\frac{a}{p_r}\right)^{\alpha_r}$$

- Useful property: $a \equiv b \pmod{N} \Rightarrow \left(\frac{a}{N}\right) = \left(\frac{b}{N}\right)$
- Theorem (from Euclid to Schönhage 1971): can compute $\left(\frac{a}{N}\right)$ efficiently without knowing the factorisation of N — in fact, in time $\tilde{O}(\log N)$

Algorithms Computing the Jacobi Symbol

ft. Euclid, 2000 years ago

Jacobi Properties

- Periodicity: $\left(\frac{a}{b}\right) = \left(\frac{a \bmod b}{b}\right)$
- Reciprocity:* $\left(\frac{a}{b}\right) = (-1)^{f(a,b)}\left(\frac{b}{a}\right)$

  for a very simple f

* modulo minor technical caveats; requires a, b odd

Algorithms Computing the Jacobi Symbol

ft. Euclid, 2000 years ago

Jacobi Properties

- Periodicity: $\left(\frac{a}{b}\right) = \left(\frac{a \bmod b}{b}\right)$
- Reciprocity:* $\left(\frac{a}{b}\right) = (-1)^{f(a,b)}\left(\frac{b}{a}\right)$

  for a very simple f

$$f(a,b) = \begin{cases} 0, \text{ if } a \equiv 1 \pmod 4 \text{ or } b \equiv 1 \pmod 4 \\ 1, \text{ if } a \equiv b \equiv 3 \pmod 4 \end{cases}$$

* modulo minor technical caveats; requires a, b odd

Algorithms Computing the Jacobi Symbol

ft. Euclid, 2000 years ago

Jacobi Properties

- Periodicity: $\left(\frac{a}{b}\right) = \left(\frac{a \bmod b}{b}\right)$
- Reciprocity:* $\left(\frac{a}{b}\right) = (-1)^{f(a,b)}\left(\frac{b}{a}\right)$

  for a very simple f

Greatest Common Divisor (GCD) Properties

- Periodicity: $\gcd(a, b) = \gcd(a \bmod b, b)$
- Reciprocity: $\gcd(a, b) = \gcd(b, a)$

* modulo minor technical caveats; requires a, b odd

Algorithms Computing the Jacobi Symbol

ft. Euclid, 2000 years ago

Jacobi Properties

- Periodicity: $\left(\frac{a}{b}\right) = \left(\frac{a \bmod b}{b}\right)$
- Reciprocity:* $\left(\frac{a}{b}\right) = (-1)^{f(a,b)}\left(\frac{b}{a}\right)$

  for a very simple f

Greatest Common Divisor (GCD) Properties

- Periodicity: $\gcd(a, b) = \gcd(a \bmod b, b)$
- Reciprocity: $\gcd(a, b) = \gcd(b, a)$

Extended Euclidean algorithm solves both these problems!

* modulo minor technical caveats; requires a, b odd

Algorithms Computing the Jacobi Symbol

ft. Euclid, 2000 years ago

- Extended Euclidean recursion: if $a < b$, swap a, b. Else, update $a \leftarrow a \bmod b$.
  - Standard runtime: $O(\log a \log b)$

Algorithms Computing the Jacobi Symbol

ft. Euclid, 2000 years ago

- Extended Euclidean recursion: if $a < b$, swap a, b. Else, update $a \leftarrow a \bmod b$.
  - Standard runtime: $O(\log a \log b)$
  - Schönhage 1971: complicated (and little-known!) divide-and-conquer algorithm that outputs the "transcript" of extended Euclidean in $\tilde{O}(\log a + \log b)$ time

Algorithms Computing the Jacobi Symbol

ft. Euclid, 2000 years ago

- Extended Euclidean recursion: if $a < b$, swap a, b. Else, update $a \leftarrow a \bmod b$.
  - Standard runtime: $O(\log a \log b)$
  - Schönhage 1971: complicated (and little-known!) divide-and-conquer algorithm that outputs the "transcript" of extended Euclidean in $\tilde{O}(\log a + \log b)$ time

Acta Informatica 1, 139—144 (1971)

Schnelle Berechnung von Kettenbruchentwicklungen

A. Schönhage

Eingegangen am 16. September 1970

Summary. A method, given by D. E. Knuth for the computation of the greatest common divisor of two integers u, v and of the continued fraction for u/v is modified in such a way that only $O(n(\lg n)^2(\lg\lg n))$ elementary steps are used for $u, v < 2^n$.

Zusammenfassung. Ein von D. E. Knuth angegebenes Verfahren, für ganze Zahlen u, v den größten gemeinsamen Teiler und den Kettenbruch für u/v zu berechnen, wird so modifiziert, daß für n-stellige Zahlen nur $O(n(\lg n)^2(\lg\lg n))$ elementare Schritte gebraucht werden.

Algorithms Computing the Jacobi Symbol

ft. Euclid, 2000 years ago

- Extended Euclidean recursion: if $a < b$, swap a, b. Else, update $a \leftarrow a \bmod b$.
  - Standard runtime: $O(\log a \log b)$
  - Schönhage 1971: complicated (and little-known!) divide-and-conquer algorithm that outputs the "transcript" of extended Euclidean in $\tilde{O}(\log a + \log b)$ time

Acta Informatica 1, 139—144 (1971)

Schnelle Berechnung von Kettenbruchentwicklungen

A. Schönhage

Eingegangen am 16. September 1970

Summary. A method, given by D. E. Knuth for the computation of the greatest common divisor of two integers u, v and of the continued fraction for u/v is modified in such a way that only $O(n(\lg n)^2(\lg\lg n))$ elementary steps are used for $u, v < 2^n$.

Zusammenfassung. Ein von D. E. Knuth angegebenes Verfahren, für ganze Zahlen u, v den größten gemeinsamen Teiler und den Kettenbruch für u/v zu berechnen, wird so modifiziert, daß für n-stellige Zahlen nur $O(n(\lg n)^2(\lg\lg n))$ elementare Schritte gebraucht werden.

A Unified Approach to HGCD Algorithms for polynomials and integers

Klaus Thull and Chee K. Yap*

Freie Universität Berlin
Fachbereich Mathematik
Arnimallee 2-6
D-1000 Berlin 33
West Germany

March, 1990

Abstract

We present a unified framework for the asymptotically fast Half-GCD (HGCD) algorithms, based on properties of the norm. Two other benefits of our approach are (a) a simplified correctness proof of the polynomial HGCD algorithm and (b) the first explicit integer HGCD algorithm. The integer HGCD algorithm turns out to be rather intricate.

Keywords: Integer GCD, Euclidean algorithm, Polynomial GCD, Half GCD algorithm, efficient algorithm.

MATHEMATICS OF COMPUTATION
Volume 77, Number 261, January 2008, Pages 589–607
S 0025-5718(07)02017-0
Article electronically published on September 12, 2007

ON SCHÖNHAGE'S ALGORITHM AND SUBQUADRATIC INTEGER GCD COMPUTATION

NIELS MÖLLER

Abstract. We describe a new subquadratic left-to-right GCD algorithm, inspired by Schönhage's algorithm for reduction of binary quadratic forms, and compare it to the first subquadratic GCD algorithm discovered by Knuth and Schönhage, and to the binary recursive GCD algorithm of Stehlé and Zimmermann. The new GCD algorithm runs slightly faster than earlier algorithms, and it is much simpler to implement. The key idea is to use a stop condition for HGCD that is based not on the size of the remainders, but on the size of the next difference. This subtle change is sufficient to eliminate the back-up steps that are necessary in all previous subquadratic left-to-right GCD algorithms. The subquadratic GCD algorithms all have the same asymptotic running time, $O(n(\log n)^2 \log\log n)$.

Factoring from Jacobi Symbol Periodicity

- For RSA integers ($N = PQ$): product of two periodic functions with smaller periods but itself only has period N

$$\left(\frac{a}{N}\right) = \left(\frac{a}{P}\right)\left(\frac{a}{Q}\right)$$

Factoring from Jacobi Symbol Periodicity

- For RSA integers ($N = PQ$): product of two periodic functions with smaller periods but itself only has period N

$$\left(\frac{a}{N}\right) = \left(\frac{a}{P}\right)\left(\frac{a}{Q}\right)$$

- What about $N = P^2Q$?

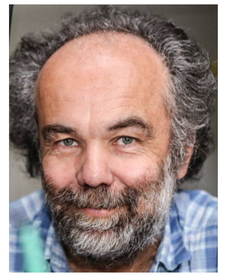
$$\left(\frac{a}{N}\right) = \left(\frac{a}{P}\right)^2\left(\frac{a}{Q}\right) = \left(\frac{a}{Q}\right), \text{ which is periodic}^* \text{ with period } Q!$$

* modulo minor technical caveats; could have $\left(\frac{a}{P}\right) = 0$ for a tiny fraction of inputs a

Quantumly Factoring $N = P^2Q$

Li, Peng, Du, Suter (2012)

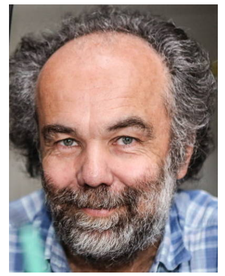
- We know $\left(\frac{a}{N}\right)$ is periodic with period Q!
- So quantum period finding → recover Q (and hence P)



Quantumly Factoring $N = P^2Q$

Li, Peng, Du, Suter (2012)

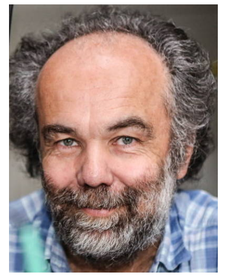
- We know $\left(\frac{a}{N}\right)$ is periodic with period Q!
- So quantum period finding → recover Q (and hence P)
- Gate complexity: cost of computing $\left(\frac{a}{N}\right)$ for $a \leq \text{poly}(Q)$, which is $\tilde{O}(\log N)$



Quantumly Factoring $N = P^2Q$

Li, Peng, Du, Suter (2012)

- We know $\left(\frac{a}{N}\right)$ is periodic with period Q!
- So quantum period finding → recover Q (and hence P)
- Gate complexity: cost of computing $\left(\frac{a}{N}\right)$ for $a \leq \text{poly}(Q)$, which is $\tilde{O}(\log N)$
- Space and depth (if naively implemented): also $\tilde{O}(\log N)$



Result 1: Pushing the Space and Depth Down to $\tilde{O}(\log Q)$




Why is There Any Hope for Sublinear Space?

- Recall: to solve period finding when the period is T, need to set up a superposition

$$\sum_{a=1}^{\text{poly}(T)} |a\rangle |f(a)\rangle$$

Why is There Any Hope for Sublinear Space?

- Recall: to solve period finding when the period is T, need to set up a superposition

$$\sum_{a=1}^{\mathsf{poly}(T)} |a\rangle |f(a)\rangle$$

- Even just writing down $|a\rangle$ requires $\log \mathsf{poly}(T) = O(\log T)$ qubits!

Why is There Any Hope for Sublinear Space?

- Recall: to solve period finding when the period is T, need to set up a superposition

$$\sum_{a=1}^{\mathrm{poly}(T)} |a\rangle |f(a)\rangle$$

- Even just writing down $|a\rangle$ requires $\log \mathrm{poly}(T) = O(\log T)$ qubits!
- In Shor (and Regev): the period is $\approx N \rightarrow$ stuck at $O(\log N)$ qubits

Why is There Any Hope for Sublinear Space?

- Recall: to solve period finding when the period is T, need to set up a superposition

$$\sum_{a=1}^{\mathrm{poly}(T)} |a\rangle |f(a)\rangle$$

- Even just writing down $|a\rangle$ requires $\log \mathrm{poly}(T) = O(\log T)$ qubits!
- In Shor (and Regev): the period is $\approx N \rightarrow$ stuck at $O(\log N)$ qubits
- **<u>Hope 1:</u>** when factoring $P^2 Q$ with Jacobi: the period is just $Q \rightarrow O(\log Q)$ qubits could suffice!

Why is There Any Hope for Sublinear Space?

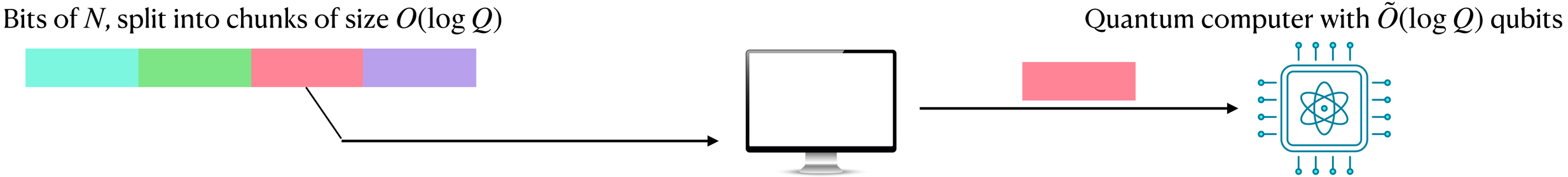
- Goal: compute $\left(\frac{a}{N}\right)$ for $a \leq \mathsf{poly}(Q)$
- How could we compute this without ever writing down all of N quantumly?

Why is There Any Hope for Sublinear Space?

- Goal: compute $\left(\frac{a}{N}\right)$ for $a \leq \mathsf{poly}(Q)$
- How could we compute this without ever writing down all of N quantumly?
- **Hope 2:** N is classically known → could quantumly "stream" through bits of N to save space

Why is There Any Hope for Sublinear Space?

- Goal: compute $\left(\frac{a}{N}\right)$ for $a \leq \text{poly}(Q)$
- How could we compute this without ever writing down all of N quantumly?
- **Hope 2:** N is classically known → could quantumly "stream" through bits of N to save space



Classical computer sending instructions to the quantum computer

Computing the Jacobi Symbol

It's all about $N \bmod a$

- Our task: compute $\left(\frac{a}{N}\right)$ for $a \leq \text{poly}(Q)$

Computing the Jacobi Symbol

It's all about $N \bmod a$

- Our task: compute $\left(\frac{a}{N}\right)$ for $a \leq \text{poly}(Q)$
- Recall:

$$\left(\frac{a}{N}\right) = (-1)^{f(a,N)}\left(\frac{N}{a}\right) = (-1)^{f(a,N)}\left(\frac{N \bmod a}{a}\right)$$

Computing the Jacobi Symbol

It's all about $N \bmod a$

- Our task: compute $\left(\frac{a}{N}\right)$ for $a \leq \text{poly}(Q)$
- Recall:

$$\left(\frac{a}{N}\right) = (-1)^{f(a,N)}\left(\frac{N}{a}\right) = (-1)^{f(a,N)}\left(\frac{N \bmod a}{a}\right)$$

- Two step procedure:
  1. Compute $N \bmod a$

Computing the Jacobi Symbol

It's all about $N \bmod a$

- Our task: compute $\left(\frac{a}{N}\right)$ for $a \leq \text{poly}(Q)$
- Recall:

$$\left(\frac{a}{N}\right) = (-1)^{f(a,N)}\left(\frac{N}{a}\right) = (-1)^{f(a,N)}\left(\frac{N \bmod a}{a}\right)$$

- Two step procedure:
  1. Compute $N \bmod a$
  2. Now $(N \bmod a) < a < \text{poly}(Q) \rightarrow$ can finish in $\tilde{O}(\log Q)$ gates/space using Schönhage

Computing the Jacobi Symbol

It's all about $N \bmod a$

- Our task: compute $\left(\frac{a}{N}\right)$ for $a \leq \mathsf{poly}(Q)$
- Recall:

$$\left(\frac{a}{N}\right) = (-1)^{f(a,N)}\left(\frac{N}{a}\right) = (-1)^{f(a,N)}\left(\frac{N \bmod a}{a}\right)$$

- Two step procedure:
  1. Compute $N \bmod a$
  2. Now $(N \bmod a) < a < \mathsf{poly}(Q) \rightarrow$ can finish in $\tilde{O}(\log Q)$ gates/space using Schönhage

The "only" bottleneck: computing $|a\rangle \mapsto |a\rangle|N \bmod a\rangle$

Our Result, Distilled

- Theorem (K**R**VV24): for quantum a and classically known N, we can compute

$$|a\rangle \mapsto |a\rangle\,|N \bmod a\rangle$$

in $\tilde{O}(\log N)$ gates (near-linear) and $\tilde{O}(\log a)$ qubits (enough qubits to write down a)

Our Result, Distilled

- Theorem (K**R**VV24): for quantum a and classically known N, we can compute

$$|a\rangle \mapsto |a\rangle\,|N \bmod a\rangle$$

  in $\tilde{O}(\log N)$ gates (near-linear) and $\tilde{O}(\log a)$ qubits (enough qubits to write down a)

- Corollary 1: all of the following can be computed in $\tilde{O}(\log N)$ gates (near-linear) and $\tilde{O}(\log a)$ qubits for quantum a and classical N:
  - Jacobi symbol: $\left(\frac{a}{N}\right)$

Our Result, Distilled

- Theorem (K**R**VV24): for quantum a and classically known N, we can compute

$$|a\rangle \mapsto |a\rangle\,|N \bmod a\rangle$$

  in $\tilde{O}(\log N)$ gates (near-linear) and $\tilde{O}(\log a)$ qubits (enough qubits to write down a)

- Corollary 1: all of the following can be computed in $\tilde{O}(\log N)$ gates (near-linear) and $\tilde{O}(\log a)$ qubits for quantum a and classical N:
  - Jacobi symbol: $\left(\frac{a}{N}\right)$
  - GCD: $\gcd(a, N)$
  - Modular inverse: $a^{-1} \bmod N$ (provided $\gcd(a, N) = 1$)

Our Result, Distilled

- Theorem (K**R**VV24): for quantum a and classically known N, we can compute

$$|a\rangle \mapsto |a\rangle\,|N \bmod a\rangle$$

  in $\tilde{O}(\log N)$ gates (near-linear) and $\tilde{O}(\log a)$ qubits (enough qubits to write down a)

- Corollary 1: all of the following can be computed in $\tilde{O}(\log N)$ gates (near-linear) and $\tilde{O}(\log a)$ qubits for quantum a and classical N:
  - Jacobi symbol: $\left(\frac{a}{N}\right)$
  - GCD: $\gcd(a, N)$
  - Modular inverse: $a^{-1} \bmod N$ (provided $\gcd(a, N) = 1$)

Open question: other applications of these results?

Our Result, Distilled

- Theorem (K**R**VV24): for quantum a and classically known N, we can compute

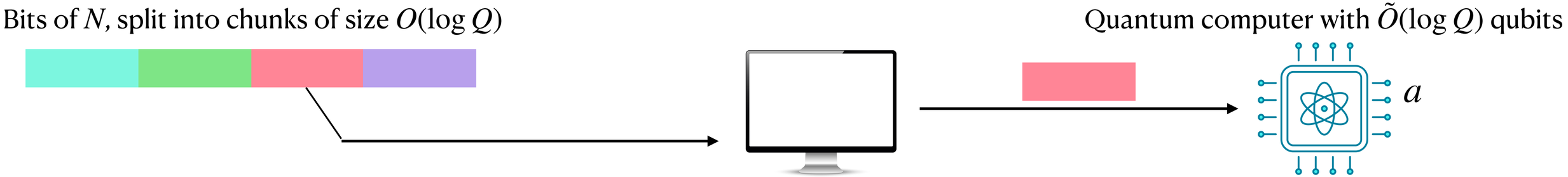
$$|a\rangle \mapsto |a\rangle\,|N \bmod a\rangle$$

  in $\tilde{O}(\log N)$ gates (near-linear) and $\tilde{O}(\log a)$ qubits (enough qubits to write down a)

- **Corollary 2: we can factor $N = P^2Q$ in $\tilde{O}(\log N)$ gates and $\tilde{O}(\log Q)$ qubits**
  - **Just need the above theorem for $a \leq \mathrm{poly}(Q)$**

Computing N mod a with Quantum Streaming

Notation: N has n bits, a has $m = O(\log Q)$ bits

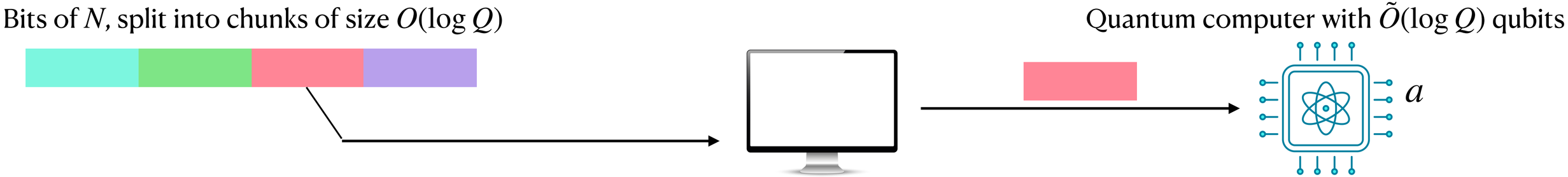


Classical computer sending instructions to the quantum computer

Computing N mod a with Quantum Streaming

- Proceeds in $t_{\max} = O(n/m)$ time steps (one for each m-bit chunk of N)

Notation: N has n bits, a has $m = O(\log Q)$ bits

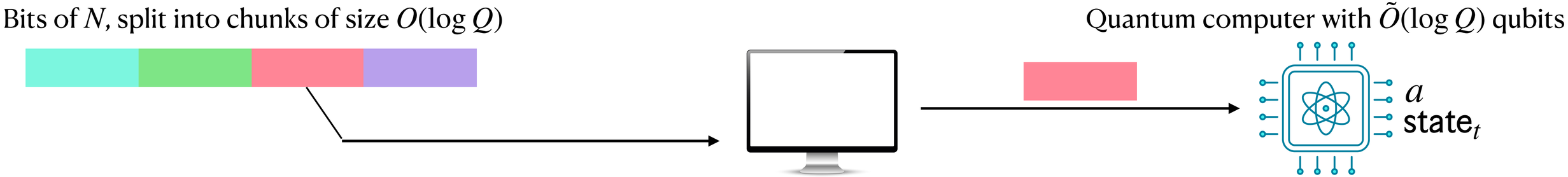


Classical computer sending instructions to the quantum computer

Computing N mod a with Quantum Streaming

- Proceeds in $t_{\max} = O(n/m)$ time steps (one for each m-bit chunk of N)
  - After time step t: quantum computer has some state_t

Notation: N has n bits, a has $m = O(\log Q)$ bits

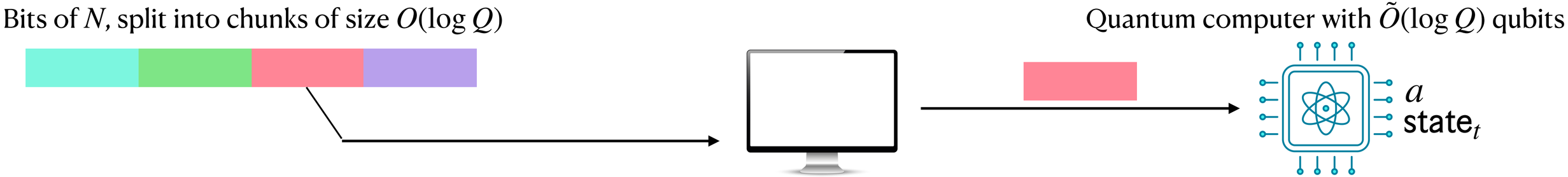


Classical computer sending instructions to the quantum computer

Computing N mod a with Quantum Streaming

- Proceeds in $t_{\max} = O(n/m)$ time steps (one for each m-bit chunk of N)
  - After time step t: quantum computer has some state_t
- Desiderata:
  - Correctness: N mod a is efficiently recoverable from $\mathsf{state}_{t_{\max}}$

Notation: N has n bits, a has $m = O(\log Q)$ bits

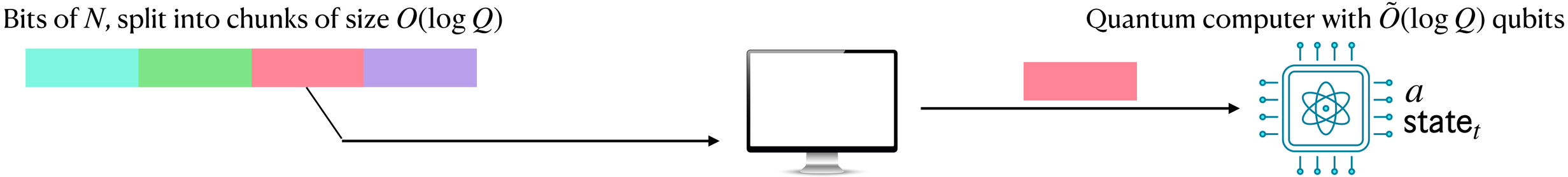


Classical computer sending instructions to the quantum computer

Computing N mod a with Quantum Streaming

- Proceeds in $t_{\max} = O(n/m)$ time steps (one for each m-bit chunk of N)
  - After time step t: quantum computer has some state_t
- Desiderata:
  - Correctness: N mod a is efficiently recoverable from $\mathsf{state}_{t_{\max}}$
  - Compactness: state_t has $O(m)$ bits for all t

Notation: N has n bits, a has $m = O(\log Q)$ bits

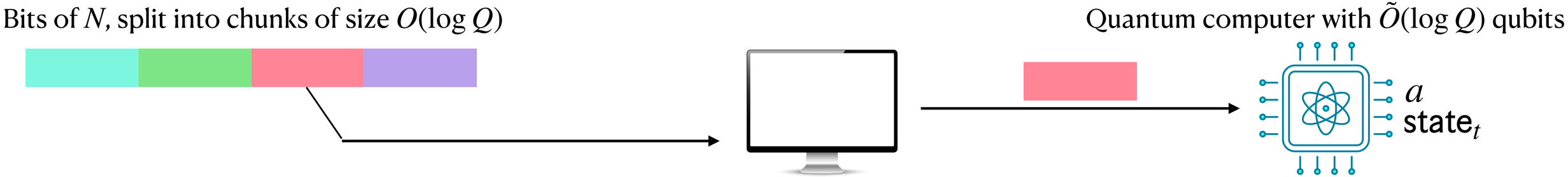


Classical computer sending instructions to the quantum computer

Computing N mod a with Quantum Streaming

- Proceeds in $t_{\max} = O(n/m)$ time steps (one for each m-bit chunk of N)
  - After time step t: quantum computer has some state_t
- Desiderata:
  - Correctness: N mod a is efficiently recoverable from $\mathsf{state}_{t_{\max}}$
  - Compactness: state_t has $O(m)$ bits for all t
  - **Reversibility:** state_{t-1} can be reconstructed (and therefore uncomputed) from state_t

Notation: N has n bits, a has $m = O(\log Q)$ bits



Our Construction, Simplified

- **TLDR: final state is a multiple of a that agrees with N in the $n - m$ lowest-order bits**

Our Construction, Simplified

- **TLDR: final state is a multiple of a that agrees with N in the $n - m$ lowest-order bits**
  - It turns out that this suffices to reconstruct $N \bmod a$

Our Construction, Simplified

- **TLDR: final state is a multiple of a that agrees with N in the $n - m$ lowest-order bits**
  - It turns out that this suffices to reconstruct $N \bmod a$
- At time $t = 0, \ldots, (n - m)/m$, let N_t be a multiple of a that agrees with N on the mt lowest-order bits

Our Construction, Simplified

- **TLDR: final state is a multiple of a that agrees with N in the $n - m$ lowest-order bits**
  - It turns out that this suffices to reconstruct $N \bmod a$
- At time $t = 0, \ldots, (n - m)/m$, let N_t be a multiple of a that agrees with N on the mt lowest-order bits
  - Equivalently: $N \equiv N_t \pmod{2^{mt}}$

Our Construction, Simplified

- **TLDR: final state is a multiple of a that agrees with N in the $n - m$ lowest-order bits**
  - It turns out that this suffices to reconstruct $N \bmod a$
- At time $t = 0, \ldots, (n - m)/m$, let N_t be a multiple of a that agrees with N on the mt lowest-order bits
  - Equivalently: $N \equiv N_t \pmod{2^{mt}}$
- N_t is constructible from N_{t-1} with essentially a multiplication of m-bit integers

Our Construction, Simplified

- **TLDR: final state is a multiple of a that agrees with N in the $n - m$ lowest-order bits**
  - It turns out that this suffices to reconstruct $N \bmod a$
- At time $t = 0, \ldots, (n - m)/m$, let N_t be a multiple of a that agrees with N on the mt lowest-order bits
  - Equivalently: $N \equiv N_t \pmod{2^{mt}}$
- N_t is constructible from N_{t-1} with essentially a multiplication of m-bit integers
- Bits of N_t split into two parts:

Our Construction, Simplified

- **TLDR: final state is a multiple of a that agrees with N in the $n - m$ lowest-order bits**
  - It turns out that this suffices to reconstruct $N \bmod a$
- At time $t = 0, \ldots, (n - m)/m$, let N_t be a multiple of a that agrees with N on the mt lowest-order bits
  - Equivalently: $N \equiv N_t \pmod{2^{mt}}$
- N_t is constructible from N_{t-1} with essentially a multiplication of m-bit integers
- Bits of N_t split into two parts:
  - mt low-order bits: these match $N \rightarrow$ classically known $\rightarrow$ no need to store quantumly

Our Construction, Simplified

- **TLDR: final state is a multiple of a that agrees with N in the $n - m$ lowest-order bits**
  - It turns out that this suffices to reconstruct $N \bmod a$
- At time $t = 0, \ldots, (n - m)/m$, let N_t be a multiple of a that agrees with N on the mt lowest-order bits
  - Equivalently: $N \equiv N_t \pmod{2^{mt}}$
- N_t is constructible from N_{t-1} with essentially a multiplication of m-bit integers
- Bits of N_t split into two parts:
  - mt low-order bits: these match $N \rightarrow$ classically known $\rightarrow$ no need to store quantumly
  - Higher-order bits: this must be held quantumly, and is state_t

Efficiency of Our Algorithm for $N \bmod a$

- Boils down to computing $\mathsf{state}_0 \to \mathsf{state}_1 \to \ldots \to \mathsf{state}_{(n-m)/m}$
  - Each step can be done by essentially multiplying m-bit integers $O(1)$ times

Efficiency of Our Algorithm for $N \bmod a$

- Boils down to computing $\mathsf{state}_0 \to \mathsf{state}_1 \to \ldots \to \mathsf{state}_{(n-m)/m}$
  - Each step can be done by essentially multiplying m-bit integers $O(1)$ times
- Space usage: $|\mathsf{state}_i| + |\text{multiplication workspace}| = \tilde{O}(m)$

Efficiency of Our Algorithm for $N \bmod a$

- Boils down to computing $\mathsf{state}_0 \to \mathsf{state}_1 \to \ldots \to \mathsf{state}_{(n-m)/m}$
  - Each step can be done by essentially multiplying m-bit integers $O(1)$ times
- Space usage: $|\mathsf{state}_i| + |\text{multiplication workspace}| = \tilde{O}(m)$
- Computational work: $O(n/m)$ multiplications of m-bit integers

Efficiency of Our Algorithm for $N \bmod a$

- Boils down to computing $\mathsf{state}_0 \to \mathsf{state}_1 \to \ldots \to \mathsf{state}_{(n-m)/m}$
  - Each step can be done by essentially multiplying m-bit integers $O(1)$ times
- Space usage: $|\mathsf{state}_i| + |\text{multiplication workspace}| = \tilde{O}(m)$
- Computational work: $O(n/m)$ multiplications of m-bit integers
  - Gates: $O(n/m) \times \tilde{O}(m) = \tilde{O}(n)$
  - Depth: $O(n/m) \times \tilde{O}(1) = \tilde{O}(n/m)$

Summary of Our Algorithm

To compute $\left(\frac{a}{N}\right)$ for $a < 2^m$ (where $m = O(\log Q)$):

1. Compute $N \bmod a$

Summary of Our Algorithm

To compute $\left(\frac{a}{N}\right)$ for $a < 2^m$ (where $m = O(\log Q)$):

1. Compute $N \bmod a$

   - Technical challenge: achieving this in gates $\tilde{O}(n)$, space $\tilde{O}(m)$, and depth $\tilde{O}(n/m)$

Summary of Our Algorithm

To compute $\left(\frac{a}{N}\right)$ for $a < 2^m$ (where $m = O(\log Q)$):

1. Compute $N \bmod a$
   - Technical challenge: achieving this in gates $\tilde{O}(n)$, space $\tilde{O}(m)$, and depth $\tilde{O}(n/m)$
2. Finish by computing $\left(\frac{N \bmod a}{a}\right)$ using Schönhage's algorithm

Summary of Our Algorithm

To compute $\left(\frac{a}{N}\right)$ for $a < 2^m$ (where $m = O(\log Q)$):

1. Compute $N \bmod a$
   - Technical challenge: achieving this in gates $\tilde{O}(n)$, space $\tilde{O}(m)$, and depth $\tilde{O}(n/m)$
2. Finish by computing $\left(\frac{N \bmod a}{a}\right)$ using Schönhage's algorithm
   - Contributes an additional depth of $\tilde{O}(m)$ (dominant term in depth if $m \gg \sqrt{n}$)

Summary: Our Factoring Circuit for P^2Q

- Main theorem (K**R**VV24): for $N = P^2Q < 2^n$ such that Q is squarefree and $< 2^m$, we can recover P, Q from N in $\tilde{O}(n)$ gates, $\tilde{O}(m)$ qubits, and $\tilde{O}(n/m + m)$ depth

Summary: Our Factoring Circuit for P^2Q

- Main theorem (K**R**VV24): for $N = P^2Q < 2^n$ such that Q is squarefree and $< 2^m$, we can recover P, Q from N in $\tilde{O}(n)$ gates, $\tilde{O}(m)$ qubits, and $\tilde{O}(n/m + m)$ depth

Central workhorse: an efficient quantum circuit for computing $N \bmod a$ for classical $N < 2^n$ and quantum $a < 2^m$

Regev's Algorithm: A Sketch

Regev's key idea: add dimensions

Before: exploited periodicity of

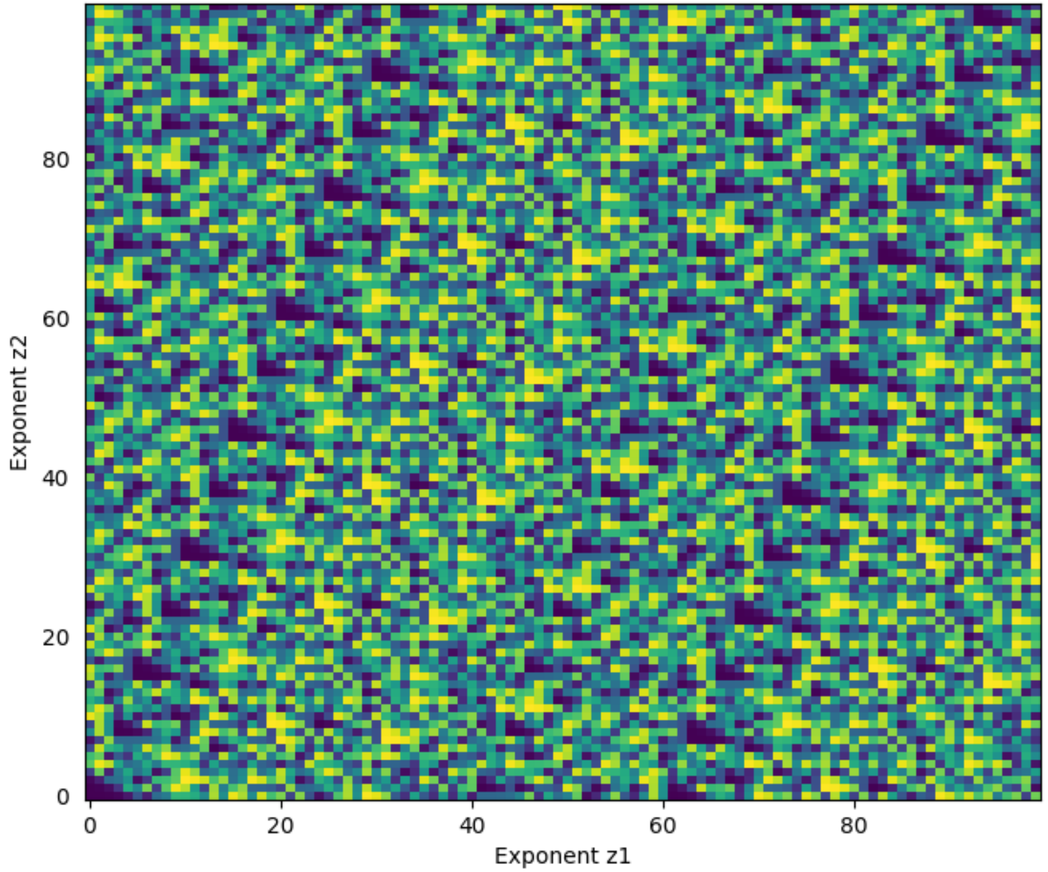
$$z \mapsto 4^z \bmod 3763.$$

Now: let's look at

$$(z_1, z_2) \mapsto 4^{z_1} 9^{z_2} \bmod 3763.$$

Regev's key idea: add dimensions

Q: How does this help us?

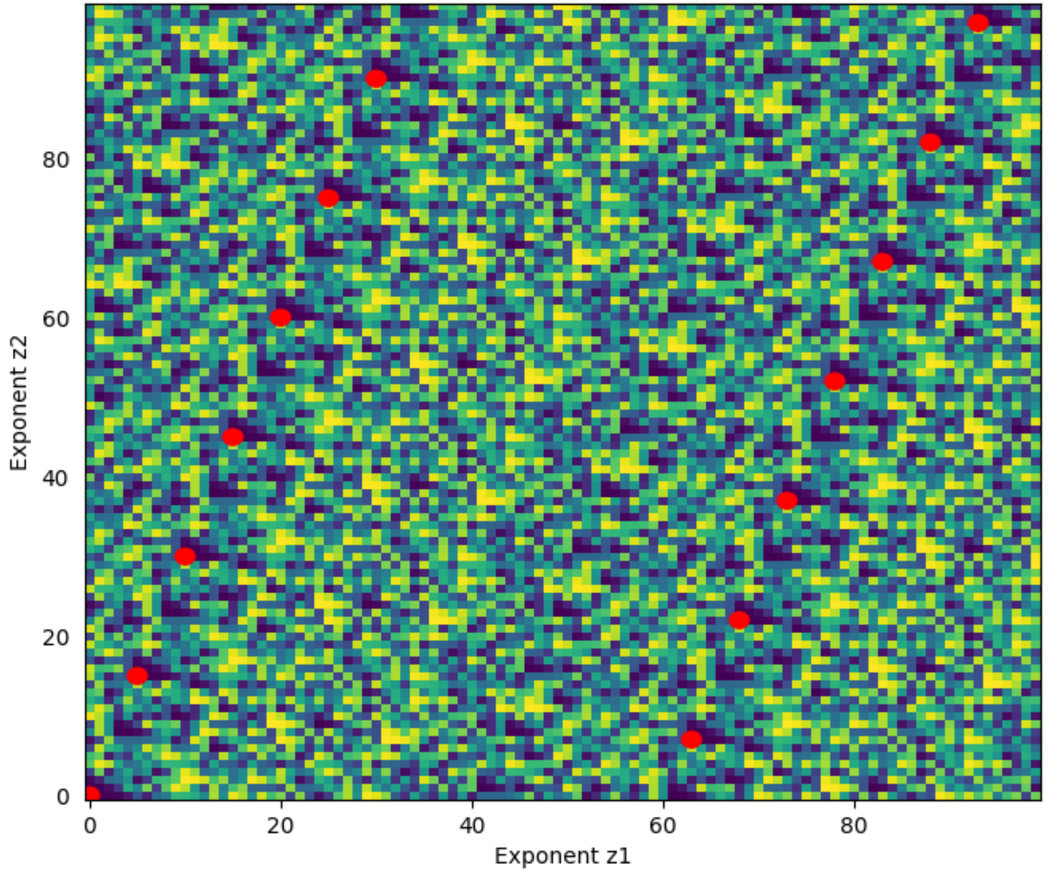


Regev's key idea: add dimensions

Q: How does this help us?

A: There are many "2D periods" now.

Crucially, these periods are much closer to $(0, 0)$ **than in Shor!**



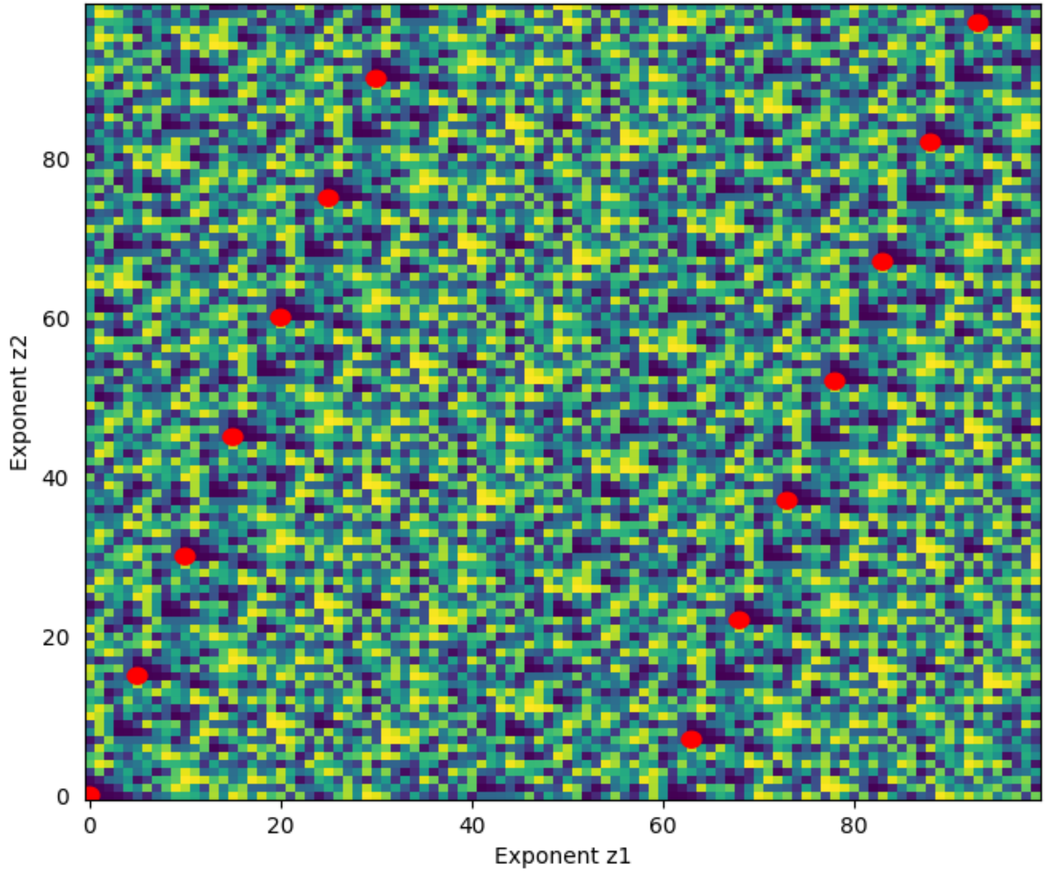
Regev's key idea: add dimensions

Q: How does this help us?

A: There are many "2D periods" now.

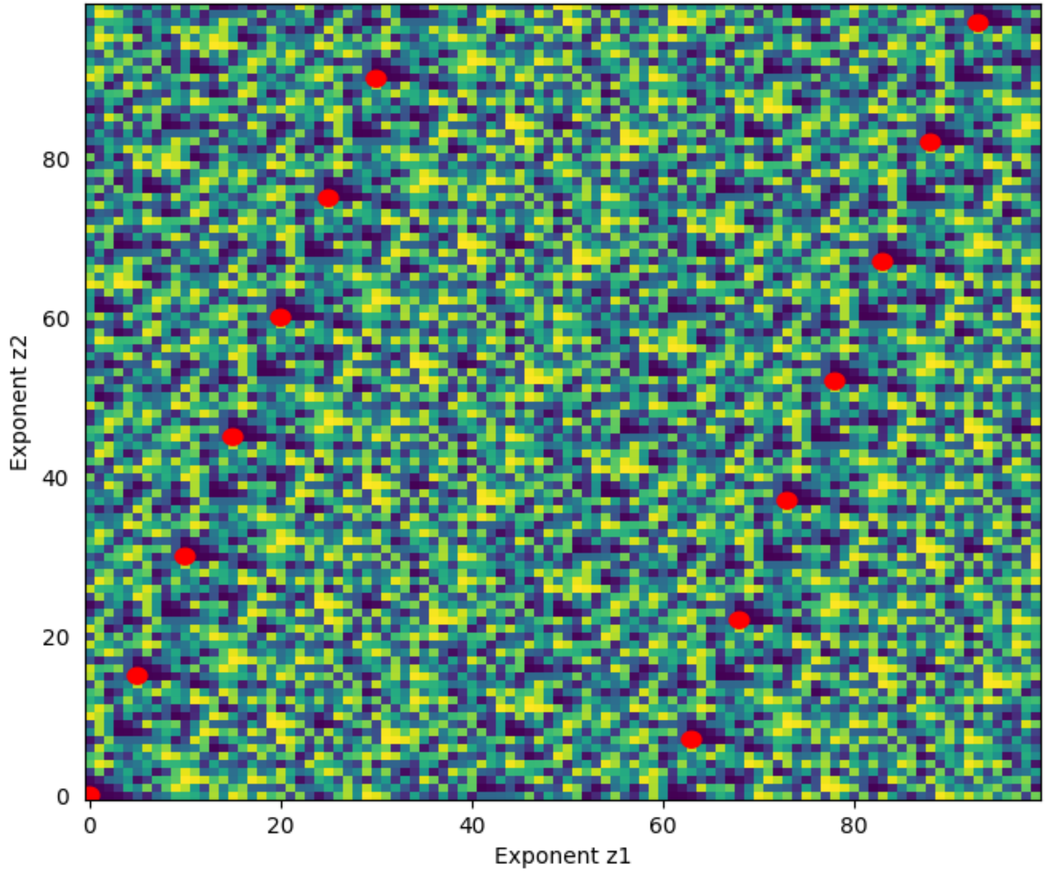
Crucially, these periods are much closer to $(0, 0)$ **than in Shor!**

Quantum circuit follows the same blueprint as Shor.



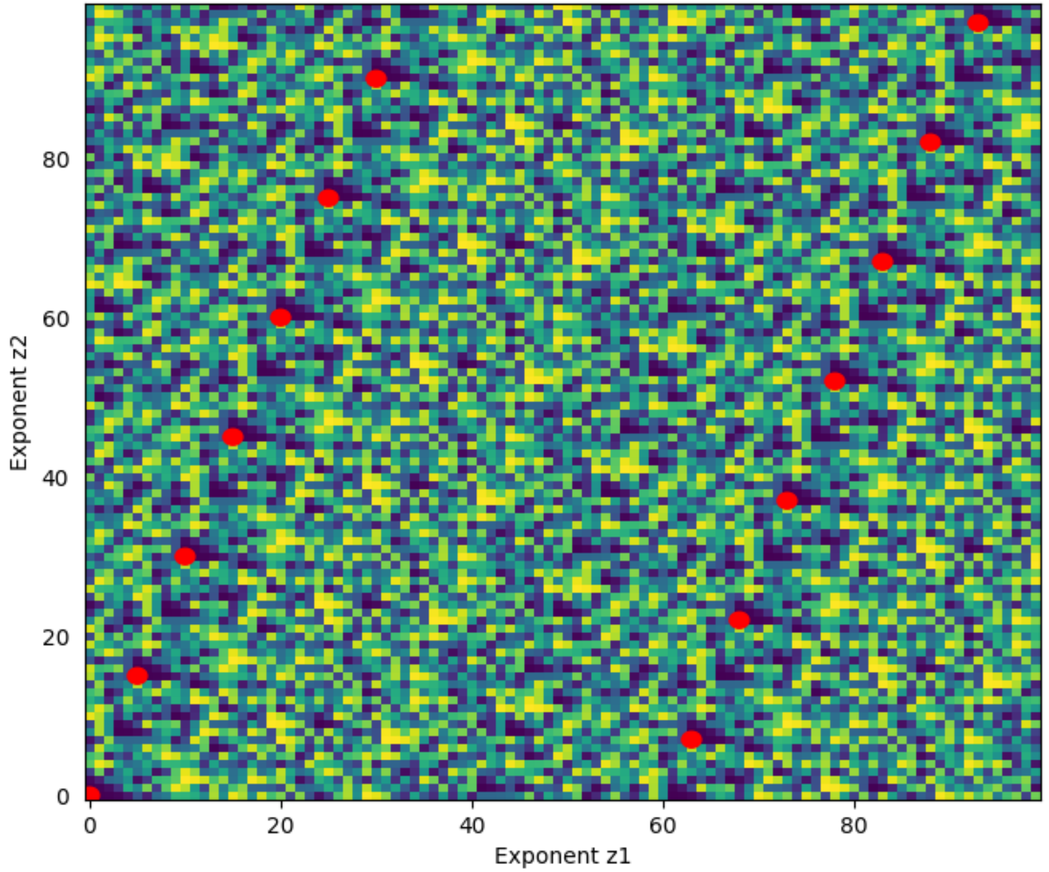
Regev's algorithm: efficiency

- Size of periods $\approx 2^{n/d}$ (we considered $d = 2$)



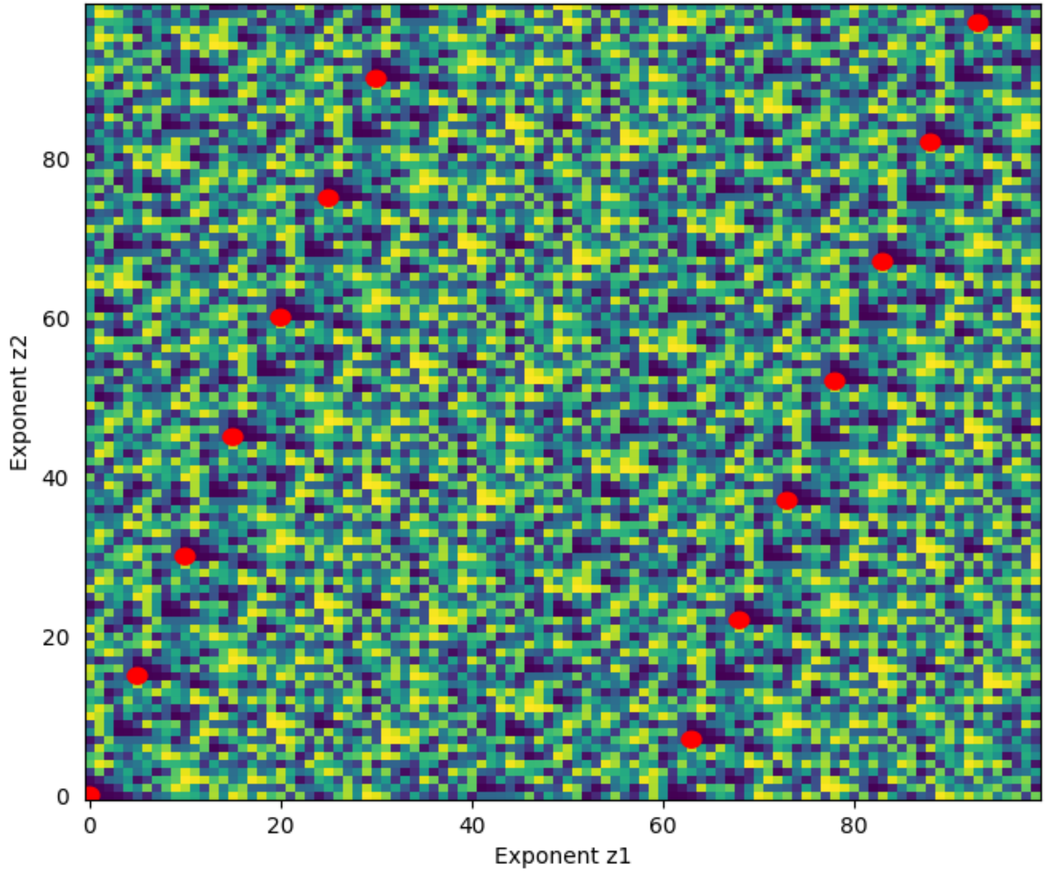
Regev's algorithm: efficiency

- Size of periods $\approx 2^{n/d}$ (we considered $d = 2$)
  - Regev's circuit: compute $a_1^{z_1} \ldots a_d^{z_d} \bmod N$ in superposition for $z_i \leq 2^{n/d}$



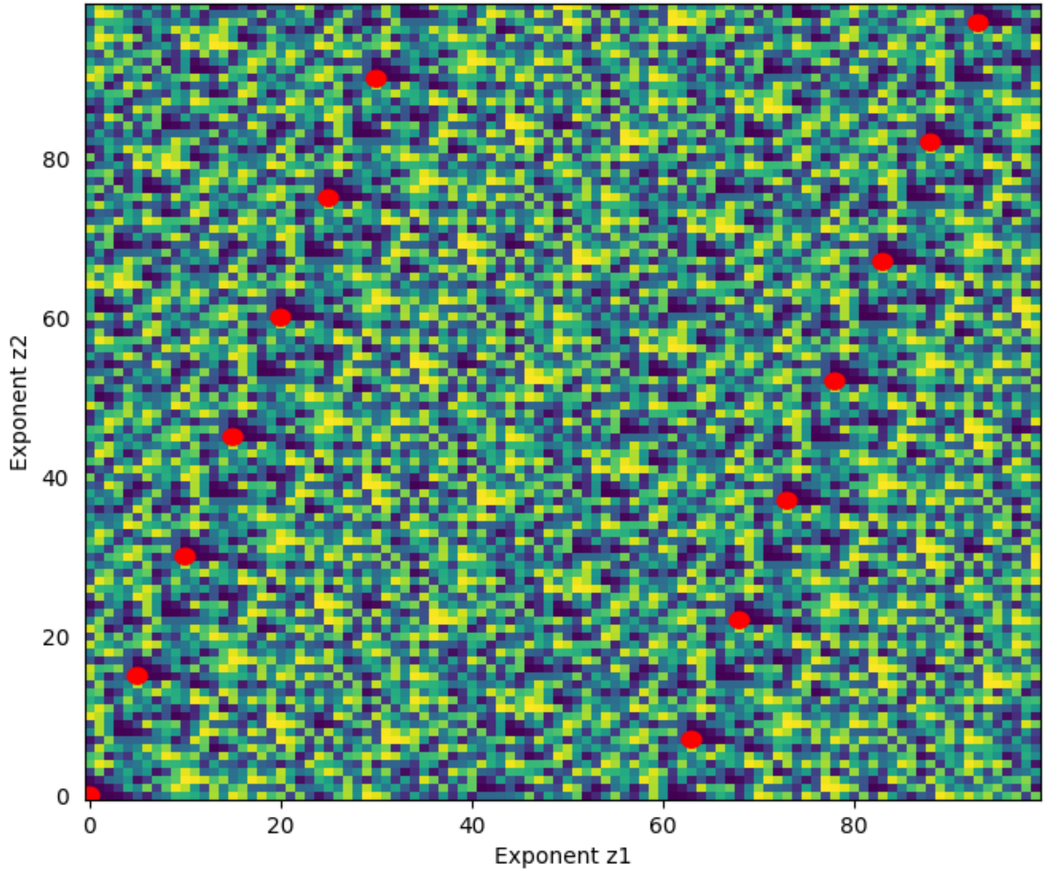
Regev's algorithm: efficiency

- Size of periods $\approx 2^{n/d}$ (we considered $d = 2$)
  - Regev's circuit: compute $a_1^{z_1} \ldots a_d^{z_d} \bmod N$ in superposition for $z_i \leq 2^{n/d}$
  - Repeated squaring $\rightarrow$ <u>at least</u> n/d multiplications needed



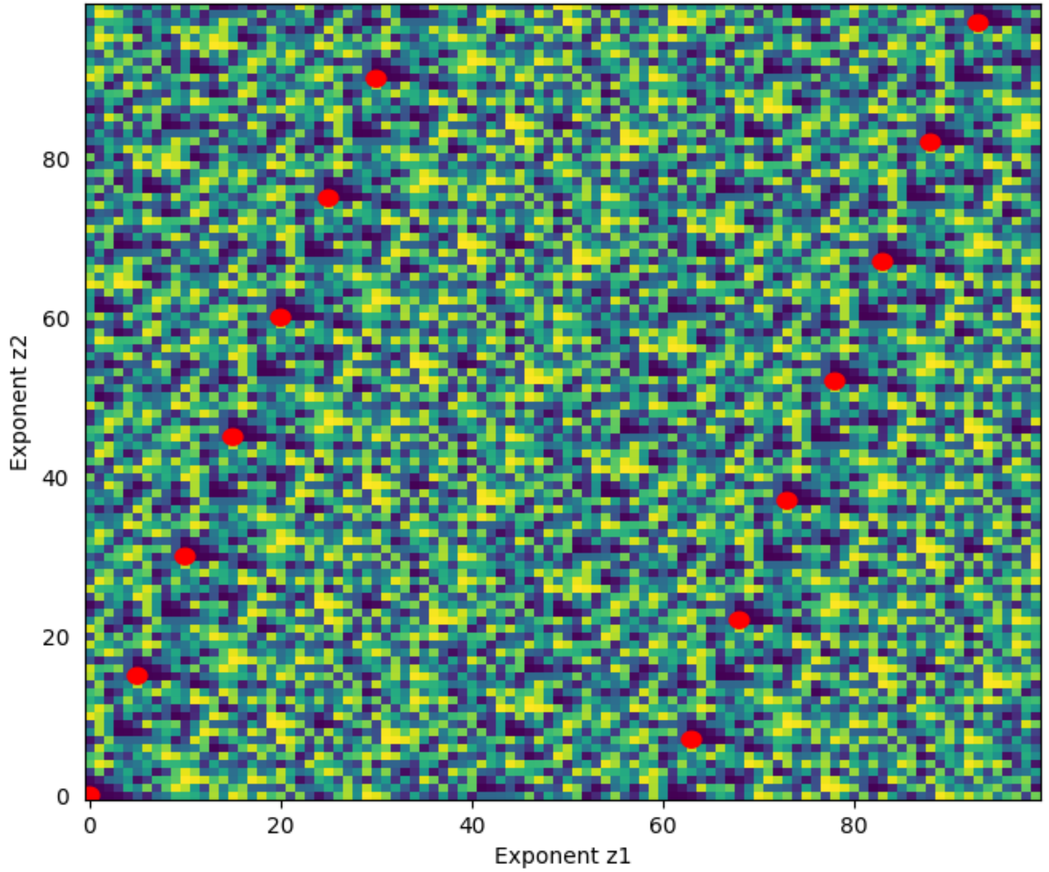
Regev's algorithm: efficiency

- Size of periods $\approx 2^{n/d}$ (we considered $d = 2$)
  - Regev's circuit: compute $a_1^{z_1} \ldots a_d^{z_d} \bmod N$ in superposition for $z_i \leq 2^{n/d}$
  - Repeated squaring → <u>at least</u> n/d multiplications needed
  - Regev magic → actually enough! *(Assuming the a_i's are small)*



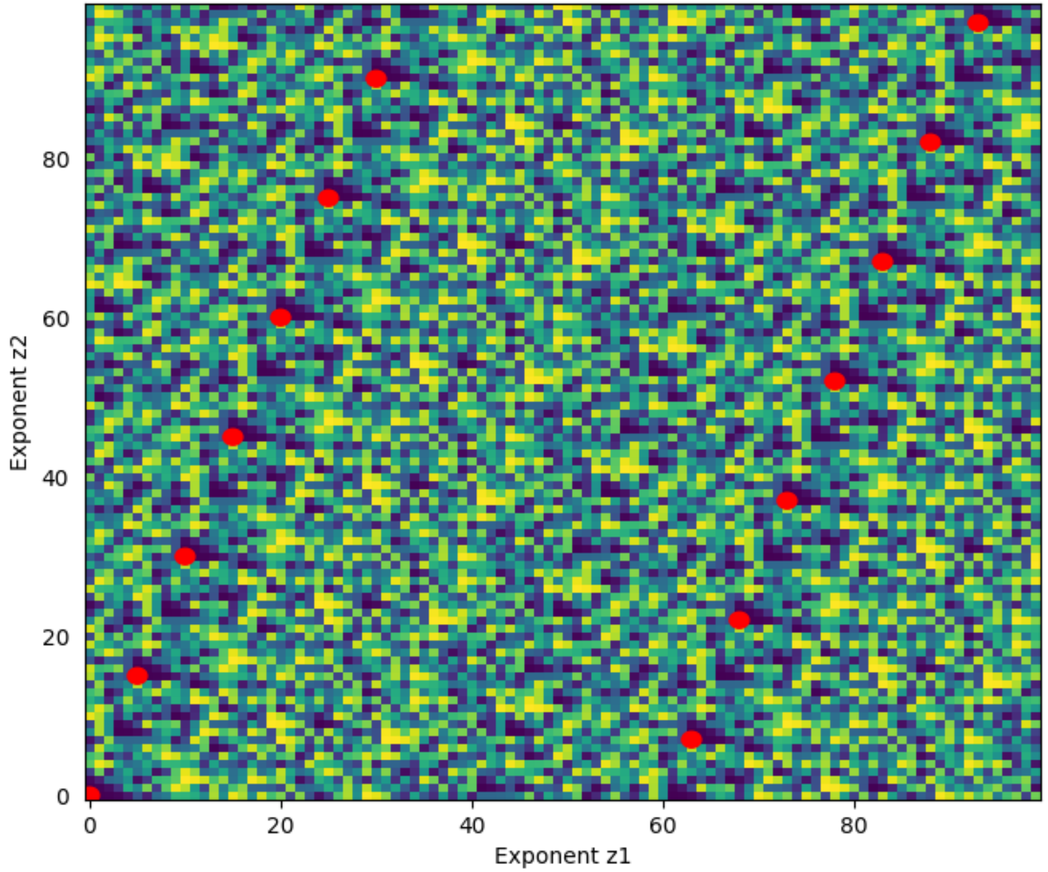
Regev's algorithm: efficiency

- Size of periods $\approx 2^{n/d}$ (we considered $d = 2$)
  - Regev's circuit: compute $a_1^{z_1} \ldots a_d^{z_d} \bmod N$ in superposition for $z_i \leq 2^{n/d}$
  - Repeated squaring → <u>at least</u> n/d multiplications needed
  - Regev magic → actually enough! *(Assuming the a_i's are small)*
  - In comparison, Shor needs n multiplications → we save a factor of d



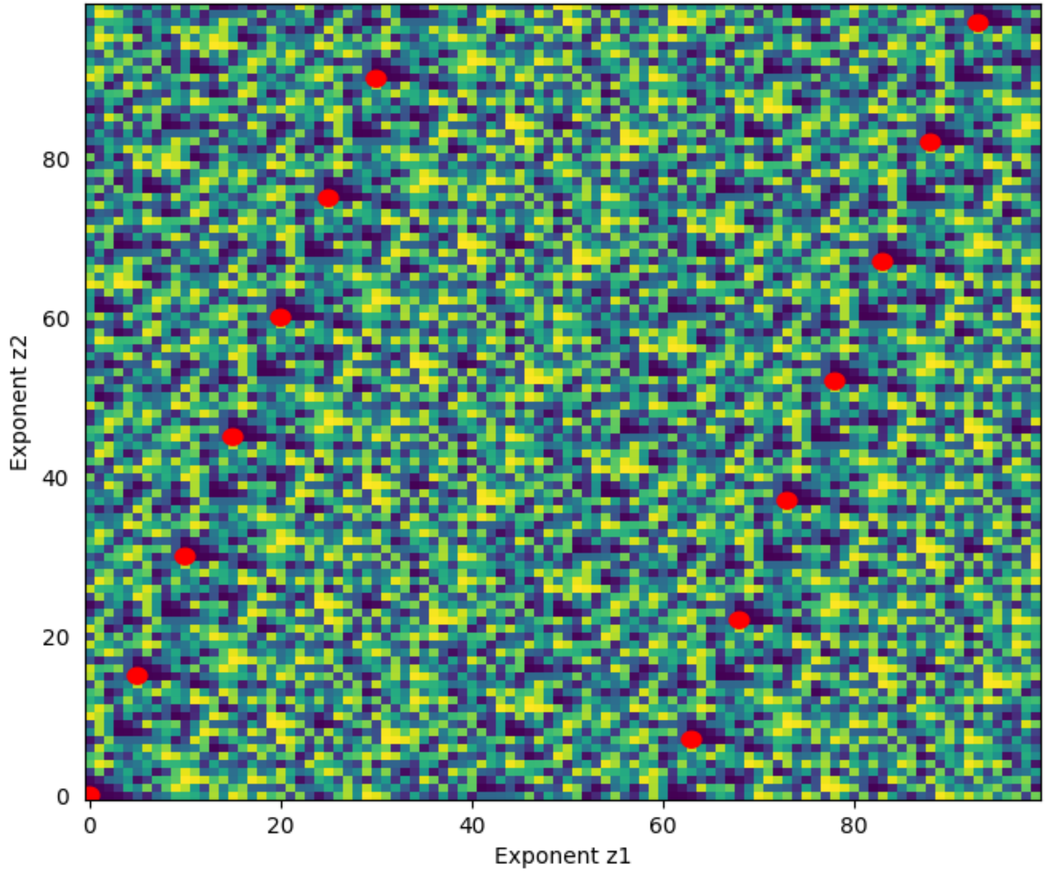
Regev's algorithm: efficiency

- Size of periods $\approx 2^{n/d} \to d \times$ faster



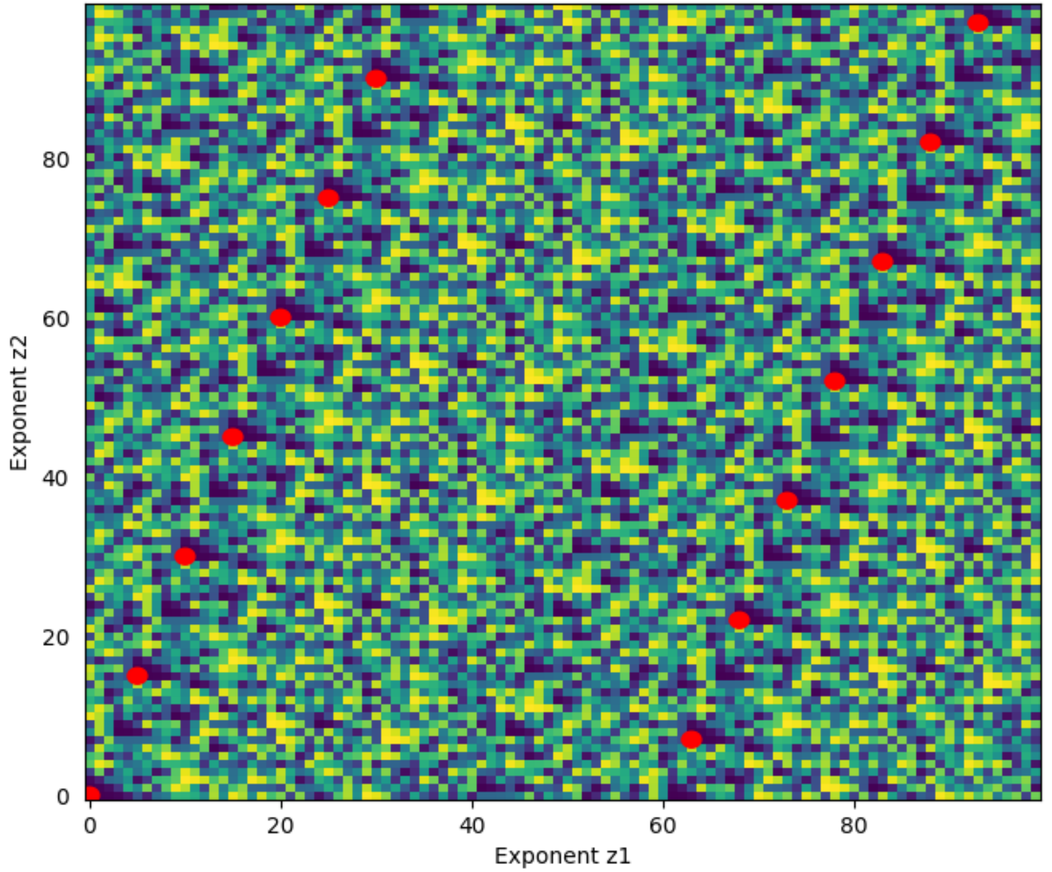
Regev's algorithm: efficiency

- Size of periods $\approx 2^{n/d} \to d \times$ faster
- Q: Why not set $d = n$?
  - Would give a $O(n \log n)$ size quantum circuit for factoring



Regev's algorithm: efficiency

- Size of periods $\approx 2^{n/d} \to d \times$ faster
- Q: Why not set $d = n$?
  - Would give a $O(n \log n)$ size quantum circuit for factoring
- A: Classical post-processing needs to solve a d-dimensional lattice problem with approximation factor $2^{O(n/d)}$
  - Sweet spot: $d = \sqrt{n}$ (LLL)



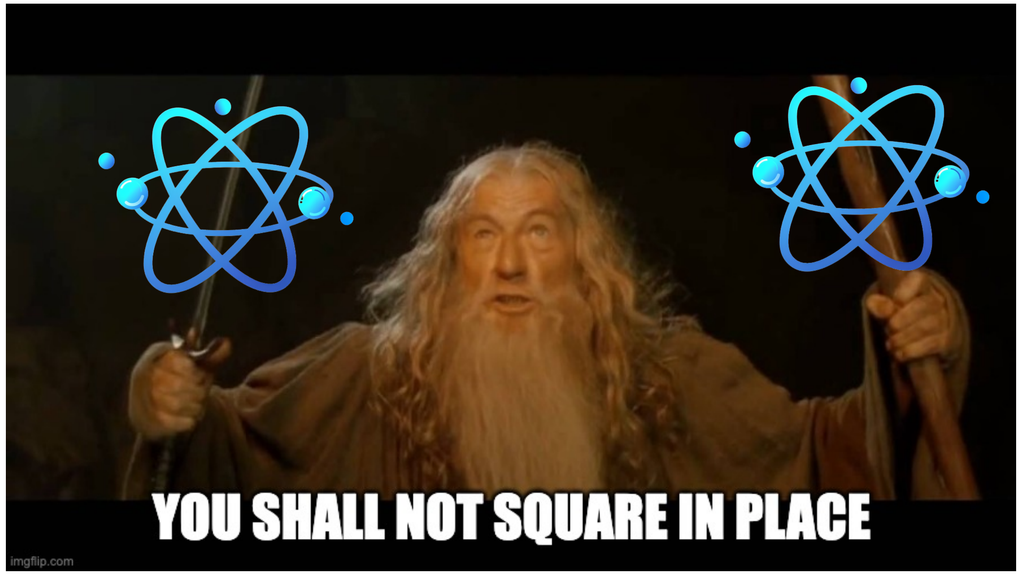
Result 2: Improving Space Complexity of Regev

Regev's qubit problem

- Performance bottleneck: computing $(z_1, \ldots, z_d) \mapsto a_1^{z_1} \ldots a_d^{z_d} \bmod N$
- Repeated squaring mod N cannot be done in place!

Regev's qubit problem

- Performance bottleneck: computing $(z_1, \ldots, z_d) \mapsto a_1^{z_1} \ldots a_d^{z_d} \bmod N$
- Repeated squaring mod N cannot be done in place!
  - Quantum circuits need to be reversible, but squaring mod N is not e.g. $-1,1$



Regev's qubit problem

- Performance bottleneck: computing $(z_1, \ldots, z_d) \mapsto a_1^{z_1} \ldots a_d^{z_d} \bmod N$
- Repeated squaring mod N cannot be done in place!
- Instead, Regev has to square *out-of-place*:

$$|a\rangle \rightsquigarrow |a, a^2\rangle \rightsquigarrow |a, a^2, a^4\rangle \rightsquigarrow |a, a^2, a^4, a^8\rangle$$

Regev's qubit problem

- Performance bottleneck: computing $(z_1, \ldots, z_d) \mapsto a_1^{z_1} \ldots a_d^{z_d} \bmod N$
- Repeated squaring mod N cannot be done in place!
- Instead, Regev has to square *out-of-place*:

$$|a\rangle \rightsquigarrow |a, a^2\rangle \rightsquigarrow |a, a^2, a^4\rangle \rightsquigarrow |a, a^2, a^4, a^8\rangle$$

Each squaring adds n qubits $\rightarrow O\left(\frac{n}{d} \times n\right) = O(n^{3/2})$ qubits total.

Aside: qubit complexity of Shor

- Does this mean Shor also requires $O\left(\frac{n}{d} \times n\right) = O(n^2)$ qubits? **No!**
  - Can precompute $a^1, a^2, a^4, a^8, a^{16}, \ldots$ classically
  - Quantum part: instead of squaring, multiply these together

Aside: qubit complexity of Shor

- Does this mean Shor also requires $O\left(\frac{n}{d} \times n\right) = O(n^2)$ qubits? **No!**
  - Can precompute $a^1, a^2, a^4, a^8, a^{16}, \ldots$ classically
  - Quantum part: instead of squaring, multiply these together
    - *Unlike squaring, multiplication can be done reversibly (not obvious... stay tuned)*

Aside: qubit complexity of Shor

- Does this mean Shor also requires $O\left(\frac{n}{d} \times n\right) = O(n^2)$ qubits? **No!**
  - Can precompute $a^1, a^2, a^4, a^8, a^{16}, \ldots$ classically
  - Quantum part: instead of squaring, multiply these together
- **Why doesn't the same precomputation trick work for Regev?**

Aside: qubit complexity of Shor

- Does this mean Shor also requires $O\left(\frac{n}{d} \times n\right) = O(n^2)$ qubits? **No!**
  - Can precompute $a^1, a^2, a^4, a^8, a^{16}, \ldots$ classically
  - Quantum part: instead of squaring, multiply these together
- **Why doesn't the same precomputation trick work for Regev?**
  - # of precomputed bits: d bases, n/d exponents, n-bit answers $\rightarrow O(n^2)$

Aside: qubit complexity of Shor

- Does this mean Shor also requires $O\left(\frac{n}{d} \times n\right) = O(n^2)$ qubits? **No!**
  - Can precompute $a^1, a^2, a^4, a^8, a^{16}, \ldots$ classically
  - Quantum part: instead of squaring, multiply these together
- **Why doesn't the same precomputation trick work for Regev?**
  - # of precomputed bits: d bases, n/d exponents, n-bit answers $\rightarrow O(n^2)$
  - Any circuit using these results should require $O(n^2)$ gates

Main Idea: Fibonacci exponentiation

- What if we used two accumulators?

$$|a, b\rangle \rightarrow |a, ab \bmod N\rangle$$

Main Idea: Fibonacci exponentiation

- What if we used two accumulators?

$$|a, b\rangle \to |a, ab \bmod N\rangle$$

- Observation by Kaliski 2017:

$$|a, a\rangle \rightsquigarrow |a, a^2\rangle \rightsquigarrow |a^3, a^2\rangle \rightsquigarrow |a^3, a^5\rangle \rightsquigarrow |a^8, a^5\rangle$$

Main Idea: Fibonacci exponentiation

- What if we used two accumulators?

$$|a, b\rangle \to |a, ab \bmod N\rangle$$

- Observation by Kaliski 2017:

$$|a, a\rangle \rightsquigarrow |a, a^2\rangle \rightsquigarrow |a^3, a^2\rangle \rightsquigarrow |a^3, a^5\rangle \rightsquigarrow |a^8, a^5\rangle$$

We can efficiently compute a^{F_k} for any Fibonacci number F_k!

Our Space Improvement

Concrete Results and Summary

- Regev's original circuit: $\approx 3n^{3/2}$ qubits, $\approx 6n^{1/2}$ multiplications mod N
- With our space optimisations: $\approx 10.4n$ qubits, $\approx 45.7n^{1/2}$ multiplications mod N

Our Space Improvement

Concrete Results and Summary

- Regev's original circuit: $\approx 3n^{3/2}$ qubits, $\approx 6n^{1/2}$ multiplications mod N
- With our space optimisations: $\approx 10.4n$ qubits, $\approx 45.7n^{1/2}$ multiplications mod N

Squaring mod N *cannot be implemented in place, while multiplication* mod N *can* $\rightarrow$ *can exponentiate with two registers using Fibonacci exponentiation,* ***without consuming additional space per step.***

Outlook and Future Directions

The True Difficulty of Factoring

The True Difficulty of Factoring

Things I do NOT expect to find in The Book

The True Difficulty of Factoring

Things I do NOT expect to find in The Book

- The time needed to classically factor an n-bit integer N is $\exp(\tilde{O}(n^{1/3}))$

The True Difficulty of Factoring

Things I do NOT expect to find in The Book

- The time needed to classically factor an n-bit integer N is $\exp(\tilde{O}(n^{1/3}))$
- The smallest quantum circuit for factoring an n-bit integer N has $\tilde{O}(n^{3/2})$ gates (Regev)...

The True Difficulty of Factoring

Things I do NOT expect to find in The Book

- The time needed to classically factor an n-bit integer N is $\exp(\tilde{O}(n^{1/3}))$
- The smallest quantum circuit for factoring an n-bit integer N has $\tilde{O}(n^{3/2})$ gates (Regev)...
  - ... unless N has a square factor, in which case we just need $\tilde{O}(n)$ gates (Jacobi)

Concrete Efficiency of Jacobi

Q: Does Jacobi factoring bring us closer to quantumly solving some factoring problem on par with RSA-2048 in classical difficulty?

Concrete Efficiency of Jacobi

Q: Does Jacobi factoring bring us closer to quantumly solving some factoring problem on par with RSA-2048 in classical difficulty?

A: Possibly!

$\tilde{O}(n^{2/3})$ qubit count, and the hidden factors don't seem bad!

Concrete Efficiency of Jacobi

Q: Does Jacobi factoring bring us closer to quantumly solving some factoring problem on par with RSA-2048 in classical difficulty?

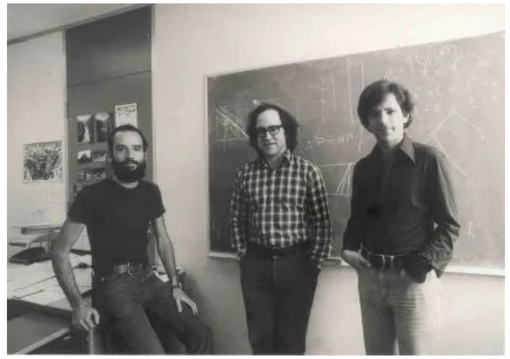
A: Possibly!

$\tilde{O}(n^{2/3})$ qubit count, and the hidden factors don't seem bad!

"Idling is just as expensive as doing operations... memory isn't cheap."

Concrete Efficiency of Regev

Q: Does Regev's algorithm (and its follow-ups by us) bring us closer to breaking RSA-2048?

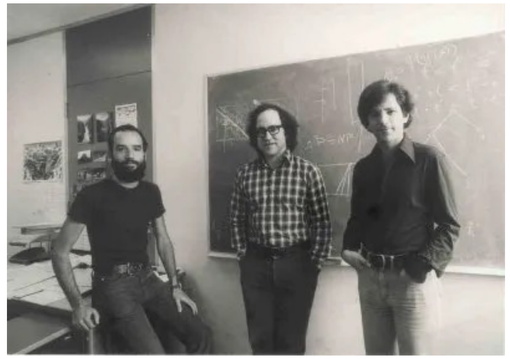


Concrete Efficiency of Regev

Q: Does Regev's algorithm (and its follow-ups by us) bring us closer to breaking RSA-2048?

A: It might with further optimisations, but not yet.

Several constant and polylog-factor optimisations to Shor make it very effective for small problem sizes.



Bespoke Fault Tolerance?

Q: Is there a clever fault-tolerant implementation of Regev or Jacobi with lower concrete overheads than a generic approach?

Bespoke Fault Tolerance?

Q: Is there a clever fault-tolerant implementation of Regev or Jacobi with lower concrete overheads than a generic approach?

A: This seems hard, but...

Bonus Slides

Our Result, Distilled

- Theorem (K**R**VV24): for $a < 2^m$ (in our application, $m = O(\log Q)$) and classically known $N < 2^n$, we can compute

$$|a\rangle \mapsto |a\rangle |N \bmod a\rangle$$

in $\tilde{O}(n)$ gates, $\tilde{O}(m)$ qubits, and $\tilde{O}(n/m)$ depth

Our Result, Distilled

- Theorem (K**R**VV24): for $a < 2^m$ (in our application, $m = O(\log Q)$) and classically known $N < 2^n$, we can compute

$$|a\rangle \mapsto |a\rangle |N \bmod a\rangle$$

  in $\tilde{O}(n)$ gates, $\tilde{O}(m)$ qubits, and $\tilde{O}(n/m)$ depth

- Corollary 1: all of the following can be computed in $\tilde{O}(n)$ gates, $\tilde{O}(m)$ qubits, and $\tilde{O}(n/m + m)$ depth for quantum a and classical N:
  - Jacobi symbol: $\left(\frac{a}{N}\right)$

Our Result, Distilled

- Theorem (K**R**VV24): for $a < 2^m$ (in our application, $m = O(\log Q)$) and classically known $N < 2^n$, we can compute

$$|a\rangle \mapsto |a\rangle\,|N \bmod a\rangle$$

  in $\tilde{O}(n)$ gates, $\tilde{O}(m)$ qubits, and $\tilde{O}(n/m)$ depth

- Corollary 1: all of the following can be computed in $\tilde{O}(n)$ gates, $\tilde{O}(m)$ qubits, and $\tilde{O}(n/m + m)$ depth for quantum a and classical N:
  - Jacobi symbol: $\left(\frac{a}{N}\right)$
  - GCD: $\gcd(a, N)$
  - Modular inverse: $a^{-1} \bmod N$ (provided $\gcd(a, N) = 1$)

Our Result, Distilled

- Theorem (K**R**VV24): for $a < 2^m$ (in our application, $m = O(\log Q)$) and classically known $N < 2^n$, we can compute

$$|a\rangle \mapsto |a\rangle\,|N \bmod a\rangle$$

  in $\tilde{O}(n)$ gates, $\tilde{O}(m)$ qubits, and $\tilde{O}(n/m)$ depth

- Corollary 1: all of the following can be computed in $\tilde{O}(n)$ gates, $\tilde{O}(m)$ qubits, and $\tilde{O}(n/m + m)$ depth for quantum a and classical N:
  - Jacobi symbol: $\left(\frac{a}{N}\right)$
  - GCD: $\gcd(a, N)$
  - Modular inverse: $a^{-1} \bmod N$ (provided $\gcd(a, N) = 1$)

Open question: other applications of these results?

From Backwards Long Division to $N \bmod a$

- The example we just worked out:
  - $N = 55, a = 3$ $(n = 6, m = 2)$

simplified presentation based on an observation by Daniel J. Bernstein

From Backwards Long Division to $N \bmod a$

- The example we just worked out:
  - $N = 55, a = 3$ $(n = 6, m = 2)$
  - End up with $ka = \text{state} = 39 = 13 \times 3 \equiv N \pmod{2^{n-m} = 16}$

simplified presentation based on an observation by Daniel J. Bernstein

From Backwards Long Division to $N \bmod a$

- The example we just worked out:
  - $N = 55, a = 3$ $(n = 6, m = 2)$
  - End up with $ka = \mathsf{state} = 39 = 13 \times 3 \equiv N \pmod{2^{n-m} = 16}$
- Now define $N' = \dfrac{N - \mathsf{state}}{2^{n-m}} = \dfrac{N - ka}{2^{n-m}}$ (informally, this is the "remainder")

simplified presentation based on an observation by Daniel J. Bernstein

From Backwards Long Division to $N \bmod a$

- The example we just worked out:
  - $N = 55, a = 3$ $(n = 6, m = 2)$
  - End up with $ka = \mathsf{state} = 39 = 13 \times 3 \equiv N \pmod{2^{n-m} = 16}$
- Now define $N' = \dfrac{N - \mathsf{state}}{2^{n-m}} = \dfrac{N - ka}{2^{n-m}}$ (informally, this is the "remainder")
- Finally: $N \equiv N' \cdot \left(2^{n-m} \bmod a\right) \pmod{a}$, and the RHS is easily computable with $\tilde{O}(m)$ gates!

simplified presentation based on an observation by Daniel J. Bernstein

From Backwards Long Division to $N \bmod a$

- The example we just worked out:
  - $N = 55, a = 3$ $(n = 6, m = 2)$
  - End up with $ka = \mathsf{state} = 39 = 13 \times 3 \equiv N \pmod{2^{n-m} = 16}$
- Now define $N' = \dfrac{N - \mathsf{state}}{2^{n-m}} = \dfrac{N - ka}{2^{n-m}}$ (informally, this is the "remainder")
- Finally: $N \equiv N' \cdot \left(2^{n-m} \bmod a\right) \pmod{a}$, and the RHS is easily computable with $\tilde{O}(m)$ gates!

That's it! We computed $N \bmod a$!

simplified presentation based on an observation by Daniel J. Bernstein

Regev's Number-Theoretic Assumption

- Regev: relies on $(z_1, \ldots, z_d) \mapsto a_1^{z_1} \ldots a_d^{z_d} \bmod N$ having periods of size $2^{O(n/d)}$
- **But these periods could just yield a trivial square root of 1** $\bmod N$
- Regev relies on a conjecture that *at least one* small period yields a non-trivial square root of 1
- Follow-up work by Pilatte proves* Regev's conjecture

** proves correctness for a variant of Regev's algorithm that is worse by polylog factors and likely impractical*