

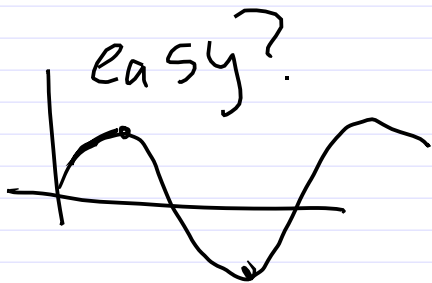
# Period Finding (real setting)

Consider  $f(x) = b^x \bmod N$

"discrete exponential"

Suppose  $f(x+r) = f(x)$

has period  $r$ ,  $b^r \bmod N = 1$



try  $m$  times  $\binom{m}{2} = \frac{m(m-1)}{2}$  pairs

$f$  is an  $n$ -bit function

so  $x$  has  $2^n$  values

$$\binom{m}{2} \sim m^2 \sim 2^n$$

$$m \sim 2^{n/2}$$

What does period finding have to do with factoring?

$$f(x) = b^x \pmod{N}$$

$$f(x+r) = b^{x+r} \pmod{N}$$

$$b^r \pmod{N} = 1$$

---

Need two conditions: 1)  $r$  even (if not, try another  $b$ )

$$(b^r - 1) \pmod{N} = 0$$

$$2) b^{r/2} + 1 \not\equiv 0 \pmod{N}$$

$$(b^{r/2} - 1)(b^{r/2} + 1) = 0 \pmod{N}$$

$\neq 0$  since  $r$  is smallest such period

$b \neq 0$

$$N = pq$$

neither  $(b^{r/2} - 1)$  nor  $(b^{r/2} + 1)$   
is a multiple of  $N$

but their product is!

Therefore...

one must have a factor of  $p$   
the other a factor of  $q$

e.g.

$$\gcd(b^{r/2} - 1, N) = p$$

$$\gcd(b^{r/2} + 1, N) = q$$

$$N=15 \quad p=3 \quad q=5$$

$$f(x) = 2^x \pmod{15}$$

$$(x=0, 1, \dots) = 1, 2, 4, 8, 1$$

$$\text{so } b=2 \Rightarrow r=4$$

$$(2^2 - 1)(2^2 + 1) = 3 \cdot 5 \\ = 0 \pmod{15}$$

$$b=7 \quad f(x) = 1, 7, 4, 13, 1$$

$$(7^2 - 1)(7^2 + 1) = 48 \cdot 50$$

$$\gcd(48, 15) = 3 \quad \gcd(50, 15) = 5 \\ \text{or } 48 = 3 \pmod{15} \quad 50 = 5 \pmod{15}$$

QM: method to find  $r$  polynomial  $O(n^3)$  in number of bits  $n$ .

---

Why is this problem relevant?  
(RSA) encryption

A  $\xrightarrow{\text{public}}$  B

Alice sends message to Bob over public channel

Bob first makes public some  $N$   
( $N$  is a product of large primes,  
 $N = p \cdot q$ ),  
and some number  $c$

Alice has message  $a$ ,  
Over public channel, she sends

encrypted  $b = a^c \pmod N$

[in python use `pow(a, c, N)`, not `a**c % N`]

Bob knows as well a  
secret  $d$  (computed by  
knowing  $p, q$ )

$$\text{s.t. } b^d \bmod N = a$$

recovers original message

( $d$  "inverts" the action of  $c$ )

QM: period of  $f(x+r) = f(x)$

can be used either to  
factor  $N$ , to get  $p, q$  and compute  $d$ ;  
or to calculate a  $d'$  which also

$$\text{has } b^{d'} \bmod N = a,$$

recovering original message  $a$

naive algorithm: divide by prime  
numbers  $\leq \sqrt{N}$

to factor  $\sim 10^{100}$   
 $\sqrt{10^{100}} \approx 10^{50}$

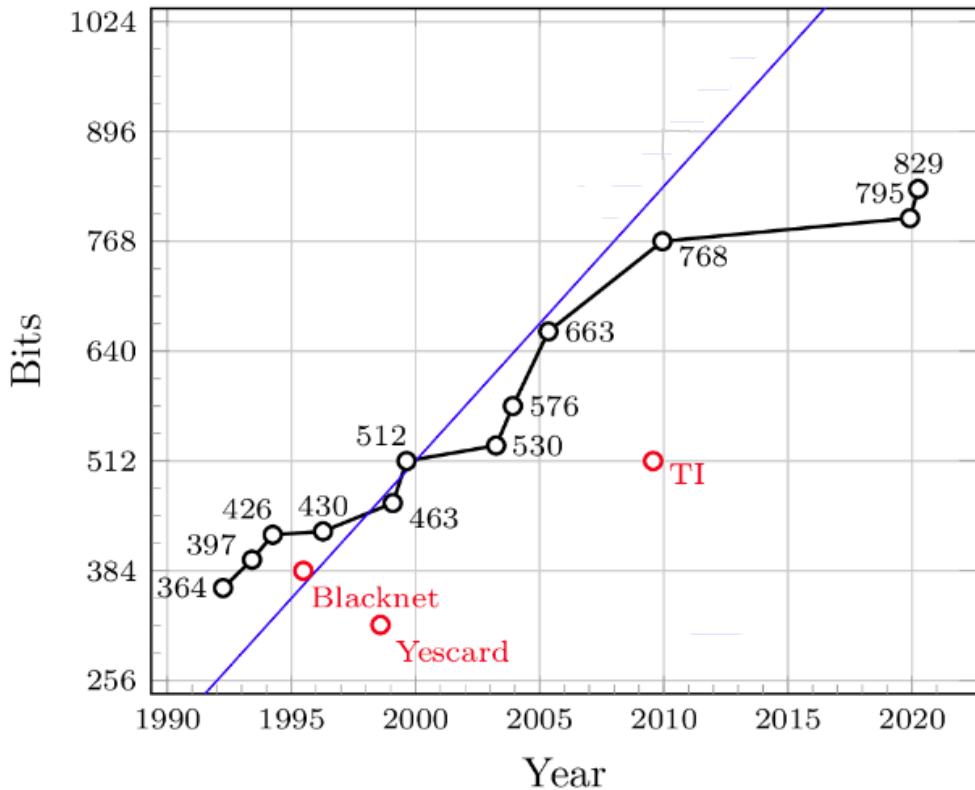
at  $10^{-9}$  / sec takes  $10^{41}$  sec

or since sec/year  $\sim \pi \cdot 10^7$

that's  $\sim 3 \cdot 10^{33}$  years 3.16

age of universe  $\sim 10^{10}$  years

best algorithm  $\sim 2^{n^{1/3}}$   
 $n = \log_2 N$



RSA numbers factored

768 bits = 232 decimal digits

795 bits = 240 " "

829 bits = 250 " "

## Quantum Physics

*[Submitted on 23 May 2019 (v1), last revised 5 Dec 2019 (this version, v2)]***How to factor 2048 bit RSA integers in 8 hours using 20 million noisy qubits**

Craig Gidney, Martin Ekerå

We significantly reduce the cost of factoring integers and computing discrete logarithms in finite fields on a quantum computer by combining techniques from Shor 1994, Griffiths–Niu 1996, Zalka 2006, Fowler 2012, Ekerå–Håstad 2017, Ekerå 2017, Ekerå 2018, Gidney–Fowler 2019, Gidney 2019. We estimate the approximate cost of our construction using plausible physical assumptions for large-scale superconducting qubit platforms: a planar grid of qubits with nearest-neighbor connectivity, a characteristic physical gate error rate of  $10^{-3}$ , a surface code cycle time of 1 microsecond, and a reaction time of 10 microseconds. We account for factors that are normally ignored such as noise, the need to make repeated attempts, and the spacetime layout of the computation. When factoring 2048 bit RSA integers, our construction's spacetime volume is a hundredfold less than comparable estimates from earlier works (Fowler et al. 2012, Gheorghiu et al. 2019). In the abstract circuit model (which ignores overheads from distillation, routing, and error correction) our construction uses  $3n + 0.002n \lg n$  logical qubits,  $0.3n^3 + 0.0005n^3 \lg n$  Toffolis, and  $500n^2 + n^2 \lg n$  measurement depth to factor  $n$ -bit RSA integers. We quantify the cryptographic implications of our work, both for RSA and for schemes based on the DLP in finite fields.

Dec 2009: 768 bits, 2000 core years

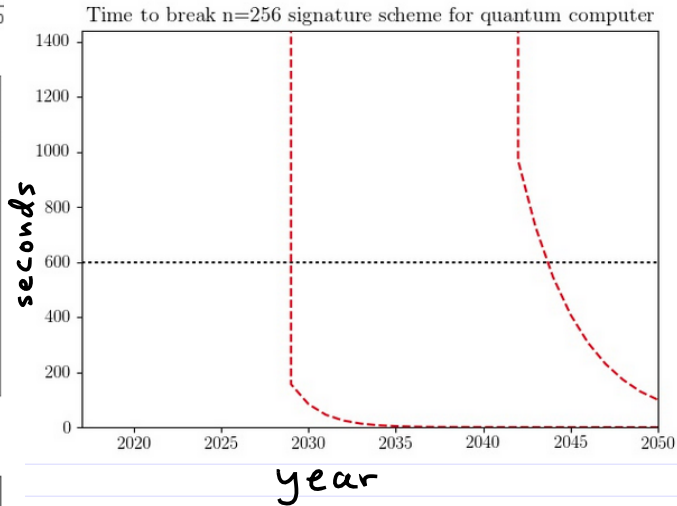
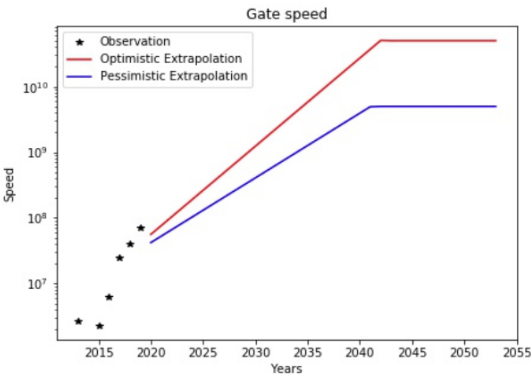
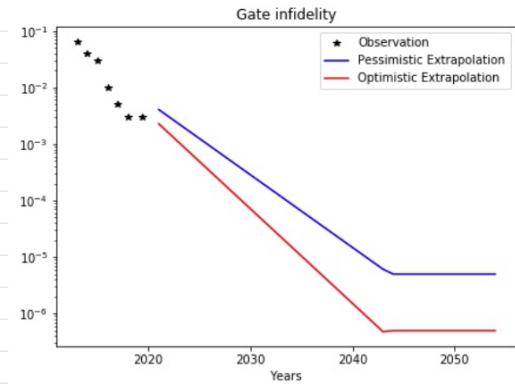
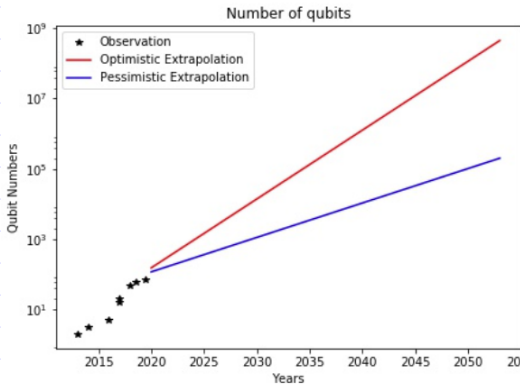
better algorithms:

Dec 2019: 795 bits took 900 core years  
[Intel Xeon Gold 2.1 GHz CPUs]

Feb 2020: 829 bits took 2700 core years

⇒ 2048 bits would take  $>10^{10}$  x more

(and  
arXiv: 1710.10377)



It's easy to find  $c^{-1} \pmod{(p-1)(q-1)}$  [or  $\pmod{n}$ ]

"Euclidean Algorithm"

("Die Hard 3 Water jug problem")

$$\cup \cup \rightarrow 4 \quad 5x + 3y = 4$$

$$x=2 \quad y=-2$$

In general, given  $N, c$   
find  $m, d$  with  $Nm + cd = j$   
then  $cd \pmod{N} = j$ .

In particular, finding  $d$   
for  $j=1$  gives the  
modular inverse:  $cd \pmod{N} = 1$

RSA  $A \rightarrow B$  message over public channel

Bob	Alice	public
$p, q$ (primes) $c, d$ s.t. $cd = 1 \pmod{(p-1)(q-1)}$ $b$ (encoded) Decode: $a = b^d \pmod N$	$a$ (her message) $c$ (not $d$ ) $N = pq$ (not $p, q$ ) $b = a^c \pmod N$	$b$ (encoded) $c$ (not $d$ ) $N = pq$ (not $p, q$ )  $QC$ : find $r$ $b^r = 1 \pmod N$ classical computer finds $d'$ with $cd' = 1 \pmod r$ $a = b^{d'} \pmod N$ decrypts

$QC$ :  $f(x) = b^x \pmod N$

find period  $r$  of  $f(x) = f(x+r)$

i.e.  $b^r \pmod N = 1$

Show how period can be used to "invert"  $c$ .

If we know  $r$ , then we can calculate  $d'$ , s.t.

$$cd' = 1 \pmod{r}$$

$$b = a^c \pmod{N}$$

$$b^{d'} \pmod{N} = a^{cd'} = a^{1+mr}$$

$$= a \pmod{N}$$

Since  $a = b^d \pmod{N}$  for some  $d$ ,

$$a^r = b^{dr} = 1^d = 1 \pmod{N}$$

$a$  also has order  $r$ :

$\{1, b, b^2, \dots, \underset{\substack{\parallel \\ a}}{b^d}, \dots, b^{r-1}\}$  contains all powers of  $b$

$\{1, a, a^2, \dots, \underset{\substack{\parallel \\ b}}{a^c}, \dots\}$

contains all powers of  $b$

must be the same

---

Example:  $r=5$   $b^3=a$

$1, b, b^2, \underset{\substack{\parallel \\ a}}{b^3}, b^4$

$1, a, a^2, a^3, a^4 = \{1, b^3, b, b^4, b^2\}$