Challenges in quantum algorithms for integer factorization

D. J. Bernstein

University of Illinois at Chicago

Prelude: What is the fastest algorithm to sort an array?

```
def blindsort(x):
  while not issorted(x):
    permuterandomly(x)
```



```
def bubblesort(x):
  for j in range(len(x)):
    for i in reversed(range(j)):
      x[i],x[i+1] = (
        min(x[i],x[i+1]),
        max(x[i],x[i+1])
      )
```

`bubblesort` takes poly time.
$\Theta(n^2)$ comparisons.
Huge speedup over `blindsort`!

Is this the end of the story?

Challenges in quantum algorithms for integer factorization

D. J. Bernstein

University of Illinois at Chicago

Prelude: What is the fastest algorithm to sort an array?

```
def blindsort(x):
  while not issorted(x):
    permuterandomly(x)
```



```
def bubblesort(x):
  for j in range(len(x)):
    for i in reversed(range(j)):
      x[i],x[i+1] = (
        min(x[i],x[i+1]),
        max(x[i],x[i+1])
      )
```

`bubblesort` takes poly time.
$\Theta(n^2)$ comparisons.
Huge speedup over `blindsort`!

Is this the end of the story?
No, still not optimal.

ges in
n algorithms for
factorization

ernstein

ty of Illinois at Chicago

What is the fastest
m to sort an array?

```
ndsort(x):
 not issorted(x):
muterandomly(x)
```



```
def bubblesort(x):
  for j in range(len(x)):
    for i in reversed(range(j)):
      x[i],x[i+1] = (
        min(x[i],x[i+1]),
        max(x[i],x[i+1])
      )
```

`bubblesort` takes poly time.
$\Theta(n^2)$ comparisons.
Huge speedup over `blindsort`!

Is this the end of the story?
No, still not optimal.

Analogo
algorithr

Shor's a
Huge sp
$b^2(\log b$
to factor
using sta
for fast i

Is this th

ns for
on

is at Chicago

---

the fastest
an array?

```
:
rted(x):
mly(x)
```



```
def bubblesort(x):
  for j in range(len(x)):
    for i in reversed(range(j)):
      x[i],x[i+1] = (
        min(x[i],x[i+1]),
        max(x[i],x[i+1])
      )
```

`bubblesort` takes poly time.
$\Theta(n^2)$ comparisons.
Huge speedup over `blindsort`!

Is this the end of the story?
No, still not optimal.

Analogous: What
algorithm to facto

Shor's algorithm t
Huge speedup ove

$b^2(\log b)^{1+o(1)}$ qu
to factor b-bit inte
using standard sub
for fast integer ari

Is this the end of t

ago

t


```
def bubblesort(x):
  for j in range(len(x)):
    for i in reversed(range(j)):
      x[i],x[i+1] = (
        min(x[i],x[i+1]),
        max(x[i],x[i+1])
      )
```

`bubblesort` takes poly time.
$\Theta(n^2)$ comparisons.
Huge speedup over `blindsort`!

Is this the end of the story?
No, still not optimal.

Analogous: What is the fast
algorithm to factor integers?

Shor's algorithm takes poly
Huge speedup over NFS!

$b^2(\log b)^{1+o(1)}$ qubit operat
to factor b-bit integer,
using standard subroutines
for fast integer arithmetic.

Is this the end of the story?

```
def bubblesort(x):
  for j in range(len(x)):
    for i in reversed(range(j)):
      x[i],x[i+1] = (
        min(x[i],x[i+1]),
        max(x[i],x[i+1])
      )
```

`bubblesort` takes poly time.
$\Theta(n^2)$ comparisons.
Huge speedup over `blindsort`!

Is this the end of the story?
No, still not optimal.

Analogous: What is the fastest algorithm to factor integers?

Shor's algorithm takes poly time.
Huge speedup over NFS!

$b^2(\log b)^{1+o(1)}$ qubit operations
to factor b-bit integer,
using standard subroutines
for fast integer arithmetic.

Is this the end of the story?

```
def bubblesort(x):
  for j in range(len(x)):
    for i in reversed(range(j)):
      x[i],x[i+1] = (
        min(x[i],x[i+1]),
        max(x[i],x[i+1])
      )
```

`bubblesort` takes poly time.
$\Theta(n^2)$ comparisons.
Huge speedup over `blindsort`!

Is this the end of the story?
No, still not optimal.

Analogous: What is the fastest algorithm to factor integers?

Shor's algorithm takes poly time.
Huge speedup over NFS!

$b^2(\log b)^{1+o(1)}$ qubit operations to factor b-bit integer, using standard subroutines for fast integer arithmetic.

Is this the end of the story?
No, still not optimal.

"Shor's algorithm: the bubble sort of integer factorization."

```
blesort(x):
 in range(len(x)):
 i in reversed(range(j)):
[i],x[i+1] = (
 min(x[i],x[i+1]),
 max(x[i],x[i+1])
```

sort takes poly time.

omparisons.

eedup over blindsort!

he end of the story?

 not optimal.

Analogous: What is the fastest algorithm to factor integers?

Shor's algorithm takes poly time. Huge speedup over NFS!

$b^2(\log b)^{1+o(1)}$ qubit operations to factor b-bit integer, using standard subroutines for fast integer arithmetic.

Is this the end of the story? No, still not optimal.

"Shor's algorithm: the bubble sort of integer factorization."

A simple

suboptin

Find a p

314159265358979323
98628034825342117
2841027019385211O5
5271201909145648566
7488152092096282925
43305727036575959
48912279381830119
70539217176293176
173637178721468440
08640344181598136
95024459455346908
38142061717766914
21712268066130019
68230301952035301
95082953311686172
285836160356370766
462080466842590694
03558764024749647
02861829745557067
60236480665499119
081647060016145249
84385233239073941
90494601653466804
22512520511739298
50471237137869609
99465764078951269
13639443745530506
74105978859597729
49972524680845987
78079771569143599
60168427394522674
3559363456817432411
56010150330861792
16829989487226588
21051141354735739
40374200731057853
10053706146806749
19561814675142691

```
):
(len(x)):
ersed(range(j)):
] = (
,x[i+1]),
,x[i+1])
```

s poly time.
s.
r `blindsort`!

the story?
nal.

Analogous: What is the fastest algorithm to factor integers?

Shor's algorithm takes poly time. Huge speedup over NFS!

$b^2(\log b)^{1+o(1)}$ qubit operations to factor b-bit integer, using standard subroutines for fast integer arithmetic.

Is this the end of the story? No, still not optimal.

"Shor's algorithm: the bubble sort of integer factorization."

A simple exercise t
suboptimality of S
Find a prime divis

3141592653589793238462643383279502884197
9862803482534211706798214808651328230664
2841027019385211055596446229489549303819
5271201909145648566923460348610454326648
7488152092096282925409171536436789259036
4330572703657595919530921861173819326117
4891227938183011949129833673362440656643
7053921717629317675238467481846766940513
1736371787214684409012249534301465495853
0864034418159813629774771309960518707211
9502445945534690830264252230825334468503
3814206171776691473035982534904287554687
2171226806613001927876611195909216420198
6823030195203530185296899577362259941389
9508295331168617278558890750983817546374
2858361603563707660104710181942955596198
4620804668425906949129331367702898915210
0355876402474964732639141992726042699227
0286182974555706749838505494588586926995
6023648066549911988183479775356636980742
0816470600161452491921732172147723501414
8438523323907394143334547762416862518983
9049460165346680498862723279178608578438
2251252051173929848960841284886269456042
5047123713786960956364371917287467764657
9946576407895126946839835259570982582262
1363944374553050682034962524517493996514
7410597885959772975498930161753928468138
4997252468084598727364469584865383673622
7807977156914359977001296160894416948685
6016842739452267467678895252138522549954
3559363456817432411251507606947945109659
5601015033086179286809208747609178249385
1682998948722658804857564014270477555132
2105114135473573952311342716610213596953
4037420073105785390621983874478084784896
1005370614680674919278191197939952061419
1956181467514269123974894090718649423196

ge(j)):

e.

ort!

Analogous: What is the fastest algorithm to factor integers?

Shor's algorithm takes poly time. Huge speedup over NFS!

$b^2(\log b)^{1+o(1)}$ qubit operations to factor b-bit integer, using standard subroutines for fast integer arithmetic.

Is this the end of the story? No, still not optimal.

"Shor's algorithm: the bubble sort of integer factorization."

A simple exercise to illustrat
suboptimality of Shor's algo
Find a prime divisor of $\lfloor 10^{3}$

31415926535897932384626433832795028841971693993751058209749445
98628034825342117067982148086513282306647093844609550582231725
28410270193852110555964462294895493038196442881097566593344612
52712019091456485669234603486104543266482133936072602491412737
74881520920962829254091715364367892590360011330530548820466521
43305727036575959195309218611738193261179310511854807446237996
48912279381830119491298336733624406566430860213949463952247371
70539217176293176752384674818467669405132000568127145263560827
17363717872146844090122495343014654958537105079227968925892354
08640344181598136297747713099605187072113499999983729780499510
95024459455346908302642522308253344685035261931188171010003137
38142061717766914730359825349042875546873115956286388235378759
21712268066130019278766111959092164201989380952572010654858632
68230301952035301852968995773622599413891249721775283479131515
95082953311686172785588907509838175463746493931925506040092770
28583616035637076601047101819429555961989467678374494482553797
46208046684259069491293313677028989152104752162056966024058038
03558764024749647326391419927260426992279678235478163600934172
02861829745557067498385054945885869269956909272107975093029553
60236480665499119881834797753566369807426542527862551818417574
08164706001614524919217321721477235014144197356854816136115735
84385233239073941433345477624168625189835694855620992192221842
90494601653466804988627232791786085784383827967976681454100953
22512520511739298489608412848862694560424196528502221066118630
50471237137869609563643719172874677646575739624138908658326459
99465764078951269468398352595709825822620522489407726719478268
13639443745530506820349625245174939965143142980919065925093722
74105978859597729754989301617539284681382686838689427741559918
49972524680845987273644695848653836736222626099124608051243884
78079771569143599770012961608944169486855584840635342207222582
60168427394522674676788952521385225499546667278239864565961163
35593634568174324112515076069479451096596094025228879710893145
56010150330861792868092087476091782493858900971490967598526136
16829989487226588048575640142704775551323796414515237462343645
21051141354735739523113427166102135969536231442952484937187110
40374200731057853906219838744780847848968332144571386875194350
10053706146806749192781911979399520614196634287544406437451237
19561814675142691239748940907186494231961567945208

Analogous: What is the fastest algorithm to factor integers?

Shor's algorithm takes poly time. Huge speedup over NFS!

$b^2(\log b)^{1+o(1)}$ qubit operations to factor b-bit integer, using standard subroutines for fast integer arithmetic.

Is this the end of the story? No, still not optimal.

"Shor's algorithm: the bubble sort of integer factorization."

A simple exercise to illustrate suboptimality of Shor's algorithm: Find a prime divisor of $\lfloor 10^{3009}\pi \rfloor$.

31415926535897932384626433832795028841971693993751058209749445923078164062862089
98628034825342117067982148086513282306647093844609550582231725359408128481117450
28410270193852110555964462294895493038196442881097566593344612847564823378678316
52712019091456485669234603486104543266482133936072602491412737245870066063155881
74881520920962829254091715364367892590360011330530548820466521384146951941511609
43305727036575959195309218611738193261179310511854807446237996274956735188575272
48912279381830119491298336733624406566430860213949463952247371907021798609437027
70539217176293176752384674818467669405132000568127145263560827785771342757789609
17363717872146844090122495343014654958537105079227968925892354201995611212902196
08640344181598136297747713099605187072113499999983729780499510597317328160963185
95024459455346908302642522308253344685035261931188171010003137838752886587533208
38142061717766914730359825349042875546873115956286388235378759375195778185778053
21712268066130019278766111959092164201989380952572010654858632788659361533818279
68230301952035301852968995773622599413891249721775283479131515574857242454150695
95082953311686172785588907509838175463746493931925506040092770167113900984882401
28583616035637076601047101819429555961989467678374494482553797747268471040475346
46208046684259069491293313677028989152104752162056966024058038150193511253382430
03558764024749647326391419927260426992279678235478163600934172164121992458631503
02861829745557067498385054945885869269956909272107975093029553211653449872027559
60236480665499119881834797753566369807426542527862551818417574672890977772793800
08164706001614524919217321721477235014144197356854816136115735255213347574184946
84385233239073941433345477624168625189835694855620992192221842725502542568876717
90494601653466804988627232791786085784383827967976681454100953883786360950680064
22512520511739298489608412848862694560424196528502221066118630674427862203919494
50471237137869609563643719172874677646575739624138908658326459958133904780275900
99465764078951269468398352595709825822620522489407726719478268482601476990902640
13639443745530506820349625245174939965143142980919065925093722169646151570985838
74105978859597729754989301617539284681382686838689427741559918559252459539594310
49972524680845987273644695848653836736222626099124608051243884390451244136549762
78079771569143599770012961608944169486855584840635342207222582848864815845602850
60168427394522674676788952521385225499546667278239864565961163548862305774564980
35593634568174324112515076069479451096596094025228879710893145669136867228748940
56010150330861792868092087476091782493858900971490967598526136554978189312978482
16829989487226588048575640142704775551323796414515237462343645428584447952658678
21051141354735739523113427166102135969536231442952484937187110145765403590279934
40374200731057853906219838744780847848968332144571386875194350643021845319104848
10053706146806749192781911979399520614196634287544406437451237181921799983910159
19561814675142691239748940907186494231961567945208

us: What is the fastest
m to factor integers?

lgorithm takes poly time.
eedup over NFS!

$)^{1+o(1)}$ qubit operations
r b-bit integer,
andard subroutines
integer arithmetic.

he end of the story?
not optimal.

algorithm: the bubble sort
er factorization."

A simple exercise to illustrate suboptimality of Shor's algorithm: Find a prime divisor of $\lfloor 10^{3009}\pi \rfloor$.

31415926535897932384626433832795028841971693993751058209749445923078164062862089
98628034825342117067982148086513282306647093844609550582231725359408128481117450
28410270193852110555964462294895493038196442881097566593344612847564823378678316
52712019091456485669234603486104543266482133936072602491412737245870066063155881
74881520920962829254091715364367892590360011330530548820466521384146951941511609
43305727036575959195309218611738193261179310511854807446237996274956735188575272
48912279381830119491298336733624406566430860213949463952247371907021798609437027
70539217176293176752384674818467669405132000568127145263560827785771342757789609
17363717872146844090122495343014654958537105079227968925892354201995611212902196
08640344181598136297747713099605187072113499999983729780499510597317328160963185
95024459455346908302642522308253344685035261931188171010003137838752886587533208
38142061717766914730359825349042875546873115956286388235378759375195778185778053
21712268066130019278766111959092164201989380952572010654858632788659361533818279
68230301952035301852968995773622599413891249721775283479131515574857242454150695
95082953311686172785588907509838175463746493931925506040092770167113900984882401
28583616035637076601047101819429555961989467678374494482553797747268471040475346
46208046684259069491293313677028989152104752162056966024058038150193511253382430
03558764024749647326391419927260426992279678235478163600934172164121992458631503
02861829745557067498385054945885869269956909272107975093029553211653449872027559
60236480665499119881834797753566369807426542527862551818417574672890977772793800
08164706001614524919217321721477235014144197356854816136115735255213347574184946
84385233239073941433345477624168625189835694855620992192221842725502542568876717
90494601653466804988627232791786085784383827967976681454100953883786360950680064
22512520511739298489608412848862694560424196528502221066118630674427862203919494
50471237137869609563643719172874677646575739624138908658326459958133904780275900
99465764078951269468398352595709825822620522489407726719478268482601476990902640
13639443745530506820349625245174939965143142980919065925093722169646151570985838
74105978859597729754989301617539284681382686838689427741559918559252459539594310
49972524680845987273644695848653836736222626099124608051243884390451244136549762
78079771569143599770012961608944169486855584840635342207222582848864815845602850
60168427394522674676788952521385225499546667278239864565961163548862305774564980
35593634568174324112515076069479451096596094025228879710893145669136867228748940
56010150330861792868092087476091782493858900971490967598526136554978189312978482
16829989487226588048575640142704775551323796414515237462343645428584447952658678
21051141354735739523113427166102135969536231442952484937187110145765403590279934
40374200731057853906219838744780847848968332144571386875194350643021845319104848
10053706146806749192781911979399520614196634287544406437451237181921799983910159
19561814675142691239748940907186494231961567945208

Importa
factoriza

- Maybe
- Maybe
- Maybe
- Maybe
- Maybe
- Maybe

Importa
(even as

- Qubits
- Area (
- Qubit
- Depth
- Time (

is the fastest
r integers?

akes poly time.
r NFS!

bit operations
eger,
oroutines
thmetic.

the story?
nal.

the bubble sort
ation."

A simple exercise to illustrate suboptimality of Shor's algorithm: Find a prime divisor of $\lfloor 10^{3009}\pi \rfloor$.

31415926535897932384626433832795028841971693993751058209749445923078164062862089
98628034825342117067982148086513282306647093844609550582231725359408128481117450
28410270193852110555964462294895493038196442881097566593344612847564823378678316
52712019091456485669234603486104543266482133936072602491412737245870066063155881
74881520920962829254091715364367892590360011330530548820466521384146951941511609
43305727036575959195309218611738193261179310511854807446237996274956735188575272
48912279381830119491298336733624406566430860213949463952247371907021798609437027
70539217176293176752384674818467669405132000568127145263560827785771342757789609
17363717872146844090122495343014654958537105079227968925892354201995611212902196
08640344181598136297747713099605187072113499999983729780499510597317328160963185
95024459455346908302642522308253344685035261931188171010003137838752886587533208
38142061717766914730359825349042875546873115956286388235378759375195778185778053
21712268066130019278766111959092164201989380952572010654858632788659361533818279
68230301952035301852968995773622599413891249721775283479131515574857242454150695
95082953311686172785588907509838175463746493931925506040092770167113900984882401
28583616035637076601047101819429555961989467678374494482553797747268471040475346
46208046684259069491293313677028989152104752162056966024058038150193511253382430
03558764024749647326391419927260426992279678235478163600934172164121992458631503
02861829745557067498385054945885869269956909272107975093029553211653449872027559
60236480665499119881834797753566369807426542527862551818417574672890977772793800
08164706001614524919217321721477235014144197356854816136115735255213347574184946
84385233239073941433345477624168625189835694855620992192221842725502542568876717
90494601653466804988627232791786085784383827967976681454100953883786360950680064
22512520511739298489608412848862694560424196528502221066118630674427862203919494
50471237137869609563643719172874677646575739624138908658326459958133904780275900
99465764078951269468398352595709825822620522489407726719478268482601476990902640
13639443745530506820349625245174939965143142980919065925093722169646151570985838
74105978859597729754989301617539284681382686838689427741559918559252459539594310
49972524680845987273644695848653836736222626099124608051243884390451244136549762
78079771569143599770012961608944169486855584840635342207222582848864815845602850
60168427394522674676788952521385225499546667278239864565961163548862305774564980
35593634568174324112515076069479451096596094025228879710893145669136867228748940
56010150330861792868092087476091782493858900971490967598526136554978189312978482
16829989487226588048575640142704775551323796414515237462343645428584447952658678
21051141354735739523113427166102135969536231442952484937187110145765403590279934
40374200731057853906219838744780847848968332144571386875194350643021845319104848
10053706146806749192781911979399520614196634287544406437451237181921799983910159
19561814675142691239748940907186494231961567945208

Important variatio
factorization probl
- Maybe need one
- Maybe need all
- Maybe factors a
- Maybe factors a
- Maybe there are
- Maybe inputs in

Important variatio
(even assuming pe
- Qubits.
- Area ("A", inclu
- Qubit operations
- Depth.
- Time ("T": late

test

?

time.

ions

le sort

A simple exercise to illustrate suboptimality of Shor's algorithm: Find a prime divisor of $\lfloor 10^{3009}\pi \rfloor$.

31415926535897932384626433832795028841971693993751058209749445923078164062862089
98628034825342117067982148086513282306647093844609550582231725359408128481117450
28410270193852110555964462294895493038196442881097566593344612847564823378678316
52712019091456485669234603486104543266482133936072602491412737245870066063155881
74881520920962829254091715364367892590360011330530548820466521384146951941511609
43305727036575959195309218611738193261179310511854807446237996274956735188575272
48912279381830119491298336733624406566430860213949463952247371907021798609437027
70539217176293176752384674818467669405132000568127145263560827785771342757789609
17363717872146844090122495343014654958537105079227968925892354201995611212902196
08640344181598136297747713099605187072113499999983729780499510597317328160963185
95024459455346908302642522308253344685035261931188171010003137838752886587533208
38142061717766914730359825349042875546873115956286388235378759375195778185778053
21712268066130019278766111959092164201989380952572010654858632788659361533818279
68230301952035301852968995773622599413891249721775283479131515574857242454150695
95082953311686172785588907509838175463746493931925506040092770167113900984882401
28583616035637076601047101819429555961989467678374494482553797747268471040475346
46208046684259069491293313677028989152104752162056966024058038150193511253382430
03558764024749647326391419927260426992279678235478163600934172164121992458631503
02861829745557067498385054945885869269956909272107975093029553211653449872027559
60236480665499119881834797753566369807426542527862551818417574672890977772793800
08164706001614524919217321721477235014144197356854816136115735255213347574184946
84385233239073941433345477624168625189835694855620992192221842725502542568876717
90494601653466804988627232791786085784383827967976681454100953883786360950680064
22512520511739298489608412848862694560424196528502221066118630674427862203919494
50471237137869609563643719172874677646575739624138908658326459958133904780275900
99465764078951269468398352595709825822620522489407726719478268482601476990902640
13639443745530506820349625245174939965143142980919065925093722169646151570985838
74105978859597729754989301617539284681382686838689427741559918559252459539594310
49972524680845987273644695848653836736222626099124608051243884390451244136549762
78079771569143599770012961608944169486855584840635342207222582848864815845602850
60168427394522674676788952521385225499546667278239864565961163548862305774564980
35593634568174324112515076069479451096596094025228879710893145669136867228748940
56010150330861792868092087476091782493858900971490967598526136554978189312978482
16829989487226588048575640142704775551323796414515237462343645428584447952658678
21051141354735739523113427166102135969536231442952484937187110145765403590279934
40374200731057853906219838744780847848968332144571386875194350643021845319104848
10053706146806749192781911979399520614196634287544406437451237181921799983910159
19561814675142691239748940907186494231961567945208

Important variations in the factorization problem:

- Maybe need one factor.
- Maybe need all factors.
- Maybe factors are small.
- Maybe factors are large.
- Maybe there are many inp
- Maybe inputs in superposi

Important variations in metr (even assuming perfect devic

- Qubits.
- Area ("A", including wire
- Qubit operations ("gates"
- Depth.
- Time ("T": latency).

A simple exercise to illustrate suboptimality of Shor's algorithm: Find a prime divisor of $\lfloor 10^{3009}\pi \rfloor$.

31415926535897932384626433832795028841971693993751058209749445923078164062862089
98628034825342117067982148086513282306647093844609550582231725359408128481117450
28410270193852110555964462294895493038196442881097566593344612847564823378678316
52712019091456485669234603486104543266482133936072602491412737245870066063155881
74881520920962829254091715364367892590360011330530548820466521384146951941511609
43305727036575959195309218611738193261179310511854807446237996274956735188575272
48912279381830119491298336733624406566430860213949463952247371907021798609437027
70539217176293176752384674818467669405132000568127145263560827785771342757789609
17363717872146844090122495343014654958537105079227968925892354201995611212902196
08640344181598136297747713099605187072113499999983729780499510597317328160963185
95024459455346908302642522308253344685035261931188171010003137838752886587533208
38142061717766914730359825349042875546873115956286388235378759375195778185778053
21712268066130019278766111959092164201989380952572010654858632788659361533818279
68230301952035301852968995773622599413891249721775283479131515574857242454150695
95082953311686172785588907509838175463746493931925506040092770167113900984882401
28583616035637076601047101819429555961989467678374494482553797747268471040475346
46208046684259069491293313677028989152104752162056966024058038150193511253382430
03558764024749647326391419927260426992279678235478163600934172164121992458631503
02861829745557067498385054945885869269956909272107975093029553211653449872027559
60236480665499119881834797753566369807426542527862551818417574672890977772793800
08164706001614524919217321721477235014144197356854816136115735255213347574184946
84385233239073941433345477624168625189835694855620992192221842725502542568876717
90494601653466804988627232791786085784383827967976681454100953883786360950680064
22512520511739298489608412848862694560424196528502221066118630674427862203919494
50471237137869609563643719172874677646575739624138908658326459958133904780275900
99465764078951269468398352595709825822620522489407726719478268482601476990902640
13639443745530506820349625245174939965143142980919065925093722169646151570985838
74105978859597729754989301617539284681382686838689427741559918559252459539594310
49972524680845987273644695848653836736222626099124608051243884390451244136549762
78079771569143599770012961608944169486855584840635342207222582848864815845602850
60168427394522674676788952521385225499546667278239864565961163548862305774564980
35593634568174324112515076069479451096596094025228879710893145669136867228748940
56010150330861792868092087476091782493858900971490967598526136554978189312978482
16829989487226588048575640142704775551323796414515237462343645428584447952658678
21051141354735739523113427166102135969536231442952484937187110145765403590279934
40374200731057853906219838744780847848968332144571386875194350643021845319104848
10053706146806749192781911979399520614196634287544406437451237181921799983910159
19561814675142691239748940907186494231961567945208

Important variations in the factorization problem:

- Maybe need one factor.
- Maybe need all factors.
- Maybe factors are small.
- Maybe factors are large.
- Maybe there are many inputs.
- Maybe inputs in superposition.

Important variations in metrics (even assuming perfect devices):

- Qubits.
- Area ("A", including wire area).
- Qubit operations ("gates").
- Depth.
- Time ("T": latency).

e exercise to illustrate
nality of Shor's algorithm:
prime divisor of $\lfloor 10^{3009}\pi \rfloor$.

84626433832795028841971693993751058209749445923078164062862089
67982148086513282306647093844609550582231725359408128481117450
55964462294895493038196442881097566593344612847564823378678316
69234603486104543266482133936072602491412737245870066063155881
54091715364367892590360011330530548820466521384146951941511609
95309218611738193261179310511854807446237996274956735188575272
91298336733624406566430860213949463952247371907021798609437027
52384674818467669405132000568127145263560827785771342757789609
90122495343014654958537105079227968925892354201995611212902196
97747713099605187072113499999983729780499510597317328160963185
02642522308253344685035261931188171010003137838752886587533208
30359825349042875546873115956286388235378759375195778185778053
78766111959092164201989380952572010654858632788659361533818279
52968995773622599413891249721775283479131515574857242454150695
85588907509838175463746493931925506040092770167113900984882401
01047101819429555961989467678374494482553797747268471040475346
91293313677028989152104752162056966024058038150193511253382430
26391419927260426992279678235478163600934172164121992458631503
98385054945885869269956909272107975093029553211653449872027559
81834797753566369807426542527862551818417574672890977772793800
19217321721477235014144197356854816136115735255213347574184946
33345477624168625189835694855620992192221842725502542568876717
88627232791786085784383827967976681454100953883786360950680064
89608412848862694560424196528502221066118630674427862203919494
63643719172874677646575739624138908658326459958133904780275900
68398352595709825822620522489407726719478268482601476990902640
20349625245174939965143142980919065925093722169646151570985838
54989301617539284681382686838689427741559918559252459539594310
73644695848653836736222626099124608051243884390451244136549762
70012961608944169486855584840635342207222582848864815845602850
76788952521385225499546667278239864565961163548862305774564980
12515076069479451096596094025228879710893145669136867228748940
68092087476091782493858900971490967598526136554978189312978482
48575640142704775551323796414515237462343645428584447952658678
23113427166102135969536231442952484937187110145765403590279934
06219838744780847848968332144571386875194350643021845319104848
92781911979399520614196634287544406437451237181921799983910159
39748940907186494231961567945208

Important variations in the factorization problem:

- Maybe need one factor.
- Maybe need all factors.
- Maybe factors are small.
- Maybe factors are large.
- Maybe there are many inputs.
- Maybe inputs in superposition.

Important variations in metrics (even assuming perfect devices):

- Qubits.
- Area ("*A*", including wire area).
- Qubit operations ("gates").
- Depth.
- Time ("*T*": latency).

Short-te

1995 Kit
Barenco-
Chari–D
1998 Za
2000 Pa
2002 Kit
Beaureg
Kunihiro
2014 Sv
2015 Gr
Smith, 2
Svore, 2
Johnston
factors c

to illustrate
hor's algorithm:
or of $\lfloor 10^{3009}\pi \rfloor$.

1693993751058209749445923078164062862089
7093844609550582231725359408128481117450
6442881097566593344612847564823378678316
2133936072602491412737245870066063155881
0011330530548820466521384146951941511609
9310511854807446237996274956735188575272
0860213949463952247371907021798609437027
2000568127145263560827785771342757789609
7105079227968925892354201995611212902196
3499999983729780499510597317328160963185
5261931188171010003137838752886587533208
3115956286388235378759375195778185778053
9380952572010654858632788659361533818279
1249721775283479131515574857242454150695
6493931925506040092770167113900984882401
9467678374494482553797747268471040475346
4752162056966024058038150193511253382430
9678235478163600934172164121992458631503
6909272107975093029553211653449872027559
6542527862551818417574672890977772793800
4197356854816136115735255213347574184946
5694855620992192221842725502542568876717
3827967976681454100953883786360950680064
4196528502221066118630674427862203919494
5739624138908658326459958133904780275900
0522489407726719478268482601476990902640
3142980919065925093722169646151570985838
2686838689427741559918559252459539594310
2626099124608051243884390451244136549762
5584840635342207222582848864815845602850
6667278239864565961163548862305774564980
6094025228879710893145669136867228748940
8900971490967598526136554978189312978482
3796414515237462343645428584447952658678
6231442952484937187110145765403590279934
8332144571386875194350643021845319104848
6634287544406437451237181921799983910159
1567945208

Important variations in the factorization problem:

- Maybe need one factor.
- Maybe need all factors.
- Maybe factors are small.
- Maybe factors are large.
- Maybe there are many inputs.
- Maybe inputs in superposition.

Important variations in metrics (even assuming perfect devices):

- Qubits.
- Area (“A”, including wire area).
- Qubit operations (“gates”).
- Depth.
- Time (“T”: latency).

Short-term RSA se

1995 Kitaev, 1996
Barenco–Ekert, 19
Chari–Devabhaktu
1998 Zalka, 1999
2000 Parker–Pleni
2002 Kitaev–Shen
Beauregard, 2006
Kunihiro, 2010 Ah
2014 Svore–Hastir
2015 Grosshans–L
Smith, 2016 Häne
Svore, 2017 Eker
Johnston: try to s
factors out of Sho

te

rithm:

$009\pi\rfloor$.

923078164062862089
359408128481117450
847564823378678316
245870066063155881
384146951941511609
274956735188575272
907021798609437027
785771342757789609
201995611212902196
597317328160963185
838752886587533208
375195778185778053
788659361533818279
574857242454150695
167113900984882401
747268471040475346
150193511253382430
164121992458631503
211653449872027559
672890977772793800
255213347574184946
725502542568876717
883786360950680064
674427862203919494
958133904780275900
482601476990902640
169646151570985838
559252459539594310
390451244136549762
848864815845602850
548862305774564980
669136867228748940
554978189312978482
428584447952658678
145765403590279934
643021845319104848
181921799983910159

Important variations in the factorization problem:

- Maybe need one factor.
- Maybe need all factors.
- Maybe factors are small.
- Maybe factors are large.
- Maybe there are many inputs.
- Maybe inputs in superposition.

Important variations in metrics (even assuming perfect devices):

- Qubits.
- Area ("*A*", including wire area).
- Qubit operations ("gates").
- Depth.
- Time ("*T*": latency).

Short-term RSA security

1995 Kitaev, 1996 Vedral–
Barenco–Ekert, 1996 Beckm
Chari–Devabhaktuni–Preskil
1998 Zalka, 1999 Mosca–Ek
2000 Parker–Plenio, 2001 Se
2002 Kitaev–Shen–Vyalyi, 2
Beauregard, 2006 Takahashi
Kunihiro, 2010 Ahmadi–Chi
2014 Svore–Hastings–Freedm
2015 Grosshans–Lawson–Mo
Smith, 2016 Häner–Roettele
Svore, 2017 Ekerå–Håstad, 2
Johnston: try to squeeze co
factors out of Shor's algorit

Important variations in the factorization problem:

- Maybe need one factor.
- Maybe need all factors.
- Maybe factors are small.
- Maybe factors are large.
- Maybe there are many inputs.
- Maybe inputs in superposition.

Important variations in metrics (even assuming perfect devices):

- Qubits.
- Area (“A”, including wire area).
- Qubit operations (“gates”).
- Depth.
- Time (“T”: latency).

Short-term RSA security

1995 Kitaev, 1996 Vedral–Barenco–Ekert, 1996 Beckman–Chari–Devabhaktuni–Preskill, 1998 Zalka, 1999 Mosca–Ekert, 2000 Parker–Plenio, 2001 Seifert, 2002 Kitaev–Shen–Vyalyi, 2003 Beauregard, 2006 Takahashi–Kunihiro, 2010 Ahmadi–Chiang, 2014 Svore–Hastings–Freedman, 2015 Grosshans–Lawson–Morain–Smith, 2016 Häner–Roetteler–Svore, 2017 Ekerå–Håstad, 2017 Johnston: try to squeeze constant factors out of Shor's algorithm.

nt variations in the
ation problem:
e need one factor.
e need all factors.
e factors are small.
e factors are large.
e there are many inputs.
e inputs in superposition.

nt variations in metrics
suming perfect devices):
s.
"A", including wire area).
operations ("gates").
.
("T": latency).

Short-term RSA security

1995 Kitaev, 1996 Vedral–Barenco–Ekert, 1996 Beckman–Chari–Devabhaktuni–Preskill, 1998 Zalka, 1999 Mosca–Ekert, 2000 Parker–Plenio, 2001 Seifert, 2002 Kitaev–Shen–Vyalyi, 2003 Beauregard, 2006 Takahashi–Kunihiro, 2010 Ahmadi–Chiang, 2014 Svore–Hastings–Freedman, 2015 Grosshans–Lawson–Morain–Smith, 2016 Häner–Roetteler–Svore, 2017 Ekerå–Håstad, 2017 Johnston: try to squeeze constant factors out of Shor's algorithm.

2003 Be
. . . 2016
$2b + 2$ c
Toffoli g
CNOT g

ns in the
em:
e factor.
factors.
re small.
re large.
e many inputs.
superposition.

ns in metrics
erfect devices):

uding wire area).
s ("gates").

ency).

Short-term RSA security

1995 Kitaev, 1996 Vedral–Barenco–Ekert, 1996 Beckman–Chari–Devabhaktuni–Preskill, 1998 Zalka, 1999 Mosca–Ekert, 2000 Parker–Plenio, 2001 Seifert, 2002 Kitaev–Shen–Vyalyi, 2003 Beauregard, 2006 Takahashi–Kunihiro, 2010 Ahmadi–Chiang, 2014 Svore–Hastings–Freedman, 2015 Grosshans–Lawson–Morain–Smith, 2016 Häner–Roetteler–Svore, 2017 Ekerå–Håstad, 2017 Johnston: try to squeeze constant factors out of Shor's algorithm.

2003 Beauregard:
. . . 2016 Häner–R
$2b + 2$ qubits; $64b$
Toffoli gates; simil
CNOT gates; dept

uts.
tion.

rics
ces):

area).
).

Short-term RSA security

1995 Kitaev, 1996 Vedral–Barenco–Ekert, 1996 Beckman–Chari–Devabhaktuni–Preskill, 1998 Zalka, 1999 Mosca–Ekert, 2000 Parker–Plenio, 2001 Seifert, 2002 Kitaev–Shen–Vyalyi, 2003 Beauregard, 2006 Takahashi–Kunihiro, 2010 Ahmadi–Chiang, 2014 Svore–Hastings–Freedman, 2015 Grosshans–Lawson–Morain–Smith, 2016 Häner–Roetteler–Svore, 2017 Ekerå–Håstad, 2017 Johnston: try to squeeze constant factors out of Shor's algorithm.

2003 Beauregard: $2b+3$ qu

. . . 2016 Häner–Roetteler–S
$2b+2$ qubits; $64b^3(\lg b +$ O
Toffoli gates; similar numbe
CNOT gates; depth $O(b^3)$.

Short-term RSA security

1995 Kitaev, 1996 Vedral–Barenco–Ekert, 1996 Beckman–Chari–Devabhaktuni–Preskill, 1998 Zalka, 1999 Mosca–Ekert, 2000 Parker–Plenio, 2001 Seifert, 2002 Kitaev–Shen–Vyalyi, 2003 Beauregard, 2006 Takahashi–Kunihiro, 2010 Ahmadi–Chiang, 2014 Svore–Hastings–Freedman, 2015 Grosshans–Lawson–Morain–Smith, 2016 Häner–Roetteler–Svore, 2017 Ekerå–Håstad, 2017 Johnston: try to squeeze constant factors out of Shor's algorithm.

2003 Beauregard: $2b + 3$ qubits.

. . . 2016 Häner–Roetteler–Svore: $2b + 2$ qubits; $64b^3(\lg b + O(1))$ Toffoli gates; similar number of CNOT gates; depth $O(b^3)$.

Short-term RSA security

1995 Kitaev, 1996 Vedral–Barenco–Ekert, 1996 Beckman–Chari–Devabhaktuni–Preskill, 1998 Zalka, 1999 Mosca–Ekert, 2000 Parker–Plenio, 2001 Seifert, 2002 Kitaev–Shen–Vyalyi, 2003 Beauregard, 2006 Takahashi–Kunihiro, 2010 Ahmadi–Chiang, 2014 Svore–Hastings–Freedman, 2015 Grosshans–Lawson–Morain–Smith, 2016 Häner–Roetteler–Svore, 2017 Ekerå–Håstad, 2017 Johnston: try to squeeze constant factors out of Shor's algorithm.

2003 Beauregard: $2b + 3$ qubits.

... 2016 Häner–Roetteler–Svore: $2b + 2$ qubits; $64b^3(\lg b + O(1))$ Toffoli gates; similar number of CNOT gates; depth $O(b^3)$.

Conventional wisdom: cannot avoid $2b$ qubits for controlled mulmod.

e.g. 4096 qubits for $b = 2048$, very common RSA key size.

So 2048-bit factorization needs 4096 qubits?

Short-term RSA security

1995 Kitaev, 1996 Vedral–Barenco–Ekert, 1996 Beckman–Chari–Devabhaktuni–Preskill, 1998 Zalka, 1999 Mosca–Ekert, 2000 Parker–Plenio, 2001 Seifert, 2002 Kitaev–Shen–Vyalyi, 2003 Beauregard, 2006 Takahashi–Kunihiro, 2010 Ahmadi–Chiang, 2014 Svore–Hastings–Freedman, 2015 Grosshans–Lawson–Morain–Smith, 2016 Häner–Roetteler–Svore, 2017 Ekerå–Håstad, 2017 Johnston: try to squeeze constant factors out of Shor's algorithm.

2003 Beauregard: $2b + 3$ qubits.
... 2016 Häner–Roetteler–Svore: $2b + 2$ qubits; $64b^3(\lg b + O(1))$ Toffoli gates; similar number of CNOT gates; depth $O(b^3)$.

Conventional wisdom: cannot avoid $2b$ qubits for controlled mulmod.

e.g. 4096 qubits for $b = 2048$, very common RSA key size.

So 2048-bit factorization needs 4096 qubits?
No: NFS uses 0 qubits.

rm RSA security

taev, 1996 Vedral–
–Ekert, 1996 Beckman–
evabhaktuni–Preskill,
lka, 1999 Mosca–Ekert,
rker–Plenio, 2001 Seifert,
taev–Shen–Vyalyi, 2003
ard, 2006 Takahashi–
, 2010 Ahmadi–Chiang,
ore–Hastings–Freedman,
osshans–Lawson–Morain–
2016 Häner–Roetteler–
017 Ekerå–Håstad, 2017
n: try to squeeze constant
out of Shor's algorithm.

2003 Beauregard: $2b+3$ qubits.
. . . 2016 Häner–Roetteler–Svore:
$2b+2$ qubits; $64b^3(\lg b + O(1))$
Toffoli gates; similar number of
CNOT gates; depth $O(b^3)$.

Conventional wisdom:
cannot avoid $2b$ qubits
for controlled mulmod.

e.g. 4096 qubits for $b = 2048$,
very common RSA key size.

So 2048-bit factorization
needs 4096 qubits?
No: NFS uses 0 qubits.

NFS tak
with $p =$
$\log L =$

Analysis
very rou

ecurity

Vedral–
96 Beckman–
ni–Preskill,
Mosca–Ekert,
o, 2001 Seifert,
–Vyalyi, 2003
Takahashi–
madi–Chiang,
gs–Freedman,
awson–Morain–
r–Roetteler–
–Håstad, 2017
queeze constant
r's algorithm.

2003 Beauregard: $2b+3$ qubits.
. . . 2016 Häner–Roetteler–Svore:
$2b+2$ qubits; $64b^3(\lg b + O(1))$
Toffoli gates; similar number of
CNOT gates; depth $O(b^3)$.

Conventional wisdom:
cannot avoid $2b$ qubits
for controlled mulmod.

e.g. 4096 qubits for $b = 2048$,
very common RSA key size.

So 2048-bit factorization
needs 4096 qubits?
No: NFS uses 0 qubits.

NFS takes $L^{p+o(1)}$
with $p = \sqrt[3]{92+2$
$\log L = (\log 2^b)^{1/3}$

Analysis for $b = 2$
very roughly 2^{112}

an–
l,
ert,
eifert,
003
–
ang,
man,
rain–
er–
2017
nstant
m.

2003 Beauregard: $2b+3$ qubits.

... 2016 Häner–Roetteler–Svore: $2b+2$ qubits; $64b^3(\lg b + O(1))$ Toffoli gates; similar number of CNOT gates; depth $O(b^3)$.

Conventional wisdom: cannot avoid $2b$ qubits for controlled mulmod.

e.g. 4096 qubits for $b = 2048$, very common RSA key size.

So 2048-bit factorization needs 4096 qubits?
No: NFS uses 0 qubits.

NFS takes $L^{p+o(1)}$ operatio
with $p = \sqrt[3]{92+26\sqrt{13}}/3$ >
$\log L = (\log 2^b)^{1/3}(\log\log 2^b$

Analysis for $b = 2048$ (not e
very roughly 2^{112} operations

2003 Beauregard: $2b+3$ qubits.
... 2016 Häner–Roetteler–Svore: $2b+2$ qubits; $64b^3(\lg b+O(1))$ Toffoli gates; similar number of CNOT gates; depth $O(b^3)$.

Conventional wisdom: cannot avoid $2b$ qubits for controlled mulmod.

e.g. 4096 qubits for $b=2048$, very common RSA key size.

So 2048-bit factorization needs 4096 qubits?
No: NFS uses 0 qubits.

NFS takes $L^{p+o(1)}$ operations with $p=\sqrt[3]{92+26\sqrt{13}}/3>1.9$, $\log L=(\log 2^b)^{1/3}(\log\log 2^b)^{2/3}$.

Analysis for $b=2048$ (not easy!): very roughly 2^{112} operations.

2003 Beauregard: $2b+3$ qubits.

... 2016 Häner–Roetteler–Svore: $2b+2$ qubits; $64b^3(\lg b + O(1))$ Toffoli gates; similar number of CNOT gates; depth $O(b^3)$.

Conventional wisdom: cannot avoid $2b$ qubits for controlled mulmod.

e.g. 4096 qubits for $b = 2048$, very common RSA key size.

So 2048-bit factorization needs 4096 qubits?
No: NFS uses 0 qubits.

NFS takes $L^{p+o(1)}$ operations with $p = \sqrt[3]{92+26\sqrt{13}}/3 > 1.9$, $\log L = (\log 2^b)^{1/3}(\log\log 2^b)^{2/3}$.

Analysis for $b = 2048$ (not easy!): very roughly 2^{112} operations.

2017 Bernstein–Biasse–Mosca: $L^{q+o(1)}$ operations with $q = \sqrt[3]{8/3} \approx 1.387$, using $b^{2/3+o(1)}$ qubits (and many non-quantum bits).

2003 Beauregard: $2b + 3$ qubits.

... 2016 Häner–Roetteler–Svore: $2b + 2$ qubits; $64b^3(\lg b + O(1))$ Toffoli gates; similar number of CNOT gates; depth $O(b^3)$.

Conventional wisdom: cannot avoid $2b$ qubits for controlled mulmod.

e.g. 4096 qubits for $b = 2048$, very common RSA key size.

So 2048-bit factorization needs 4096 qubits?
No: NFS uses 0 qubits.

NFS takes $L^{p+o(1)}$ operations with $p = \sqrt[3]{92 + 26\sqrt{13}}/3 > 1.9$, $\log L = (\log 2^b)^{1/3}(\log\log 2^b)^{2/3}$.

Analysis for $b = 2048$ (not easy!): very roughly 2^{112} operations.

2017 Bernstein–Biasse–Mosca: $L^{q+o(1)}$ operations with $q = \sqrt[3]{8/3} \approx 1.387$, using $b^{2/3+o(1)}$ qubits (and many non-quantum bits).

Open: Analyze for $b = 2048$.
Fewer than 4096 qubits?
Fewer than 2048 qubits?

auregard: $2b+3$ qubits.
6 Häner–Roetteler–Svore:
qubits; $64b^3(\lg b + O(1))$
gates; similar number of
gates; depth $O(b^3)$.

ional wisdom:
avoid $2b$ qubits
rolled mulmod.

6 qubits for $b = 2048$,
nmon RSA key size.

-bit factorization
096 qubits?
S uses 0 qubits.

NFS takes $L^{p+o(1)}$ operations
with $p = \sqrt[3]{92+26\sqrt{13}}/3 > 1.9$,
$\log L = (\log 2^b)^{1/3}(\log\log 2^b)^{2/3}$.

Analysis for $b = 2048$ (not easy!):
very roughly 2^{112} operations.

2017 Bernstein–Biasse–Mosca:
$L^{q+o(1)}$ operations
with $q = \sqrt[3]{8/3} \approx 1.387$,
using $b^{2/3+o(1)}$ qubits
(and many non-quantum bits).

Open: Analyze for $b = 2048$.
Fewer than 4096 qubits?
Fewer than 2048 qubits?

Countin
oversimp
commun
See, e.g.
theorem

2$b+3$ qubits.
oetteler–Svore:
$b^3(\lg b + O(1))$
ar number of
th $O(b^3)$.

om:
ubits
mod.

or $b = 2048$,
key size.

ization
?
ubits.

NFS takes $L^{p+o(1)}$ operations
with $p = \sqrt[3]{92 + 26\sqrt{13}}/3 > 1.9$,
$\log L = (\log 2^b)^{1/3}(\log\log 2^b)^{2/3}$.

Analysis for $b = 2048$ (not easy!):
very roughly 2^{112} operations.

2017 Bernstein–Biasse–Mosca:
$L^{q+o(1)}$ operations
with $q = \sqrt[3]{8/3} \approx 1.387$,
using $b^{2/3+o(1)}$ qubits
(and many non-quantum bits).

Open: Analyze for $b = 2048$.
Fewer than 4096 qubits?
Fewer than 2048 qubits?

Counting operatio
oversimplified cost
communication co
See, e.g., 1981 Br
theorem for realist

ubits.
Svore:
$O(1))$
r of

48,

NFS takes $L^{p+o(1)}$ operations
with $p = \sqrt[3]{92 + 26\sqrt{13}}/3 > 1.9$,
$\log L = (\log 2^b)^{1/3}(\log\log 2^b)^{2/3}$.

Analysis for $b = 2048$ (not easy!):
very roughly 2^{112} operations.

2017 Bernstein–Biasse–Mosca:
$L^{q+o(1)}$ operations
with $q = \sqrt[3]{8/3} \approx 1.387$,
using $b^{2/3+o(1)}$ qubits
(and many non-quantum bits).

Open: Analyze for $b = 2048$.
Fewer than 4096 qubits?
Fewer than 2048 qubits?

Counting operations is an
oversimplified cost model: ig
communication costs, parall
See, e.g., 1981 Brent–Kung
theorem for realistic chip mo

NFS takes $L^{p+o(1)}$ operations
with $p = \sqrt[3]{92 + 26\sqrt{13}}/3 > 1.9$,
$\log L = (\log 2^b)^{1/3}(\log\log 2^b)^{2/3}$.

Analysis for $b = 2048$ (not easy!):
very roughly 2^{112} operations.

2017 Bernstein–Biasse–Mosca:
$L^{q+o(1)}$ operations
with $q = \sqrt[3]{8/3} \approx 1.387$,
using $b^{2/3+o(1)}$ qubits
(and many non-quantum bits).

Open: Analyze for $b = 2048$.
Fewer than 4096 qubits?
Fewer than 2048 qubits?

Counting operations is an
oversimplified cost model: ignores
communication costs, parallelism.
See, e.g., 1981 Brent–Kung *AT*
theorem for realistic chip model.

NFS takes $L^{p+o(1)}$ operations with $p = \sqrt[3]{92 + 26\sqrt{13}}/3 > 1.9$, $\log L = (\log 2^b)^{1/3}(\log\log 2^b)^{2/3}$.

Analysis for $b = 2048$ (not easy!): very roughly 2^{112} operations.

2017 Bernstein–Biasse–Mosca: $L^{q+o(1)}$ operations with $q = \sqrt[3]{8/3} \approx 1.387$, using $b^{2/3+o(1)}$ qubits (and many non-quantum bits).

Open: Analyze for $b = 2048$. Fewer than 4096 qubits? Fewer than 2048 qubits?

Counting operations is an oversimplified cost model: ignores communication costs, parallelism. See, e.g., 1981 Brent–Kung AT theorem for realistic chip model.

NFS suffers somewhat from communication costs inside big linear-algebra subroutine.

2001 Bernstein: $AT = L^{p'+o(1)}$ with $p' \approx 1.976$.

2017 Bernstein–Biasse–Mosca: $AT = L^{q'+o(1)}$ with $q' \approx 1.456$ using $b^{2/3+o(1)}$ qubits. Open: Analyze for $b = 2048$.

es $L^{p+o(1)}$ operations
$= \sqrt[3]{92 + 26\sqrt{13}}/3 > 1.9$,
$(\log 2^b)^{1/3}(\log\log 2^b)^{2/3}$.

for $b = 2048$ (not easy!):
ghly 2^{112} operations.

rnstein–Biasse–Mosca:
operations
$= \sqrt[3]{8/3} \approx 1.387$,
$/3+o(1)$ qubits
ny non-quantum bits).

nalyze for $b = 2048$.
han 4096 qubits?
han 2048 qubits?

Counting operations is an oversimplified cost model: ignores communication costs, parallelism. See, e.g., 1981 Brent–Kung AT theorem for realistic chip model.

NFS suffers somewhat from communication costs inside big linear-algebra subroutine.

2001 Bernstein:
$AT = L^{p'+o(1)}$ with $p' \approx 1.976$.

2017 Bernstein–Biasse–Mosca:
$AT = L^{q'+o(1)}$ with $q' \approx 1.456$
using $b^{2/3+o(1)}$ qubits.
Open: Analyze for $b = 2048$.

Actually
Lower co
Lower co

) operations
$26\sqrt{13}/3 > 1.9$,
$^{3}(\log\log 2^b)^{2/3}$.

048 (not easy!):
operations.

asse–Mosca:
s
1.387,
ubits
uantum bits).

$b = 2048$.
qubits?
qubits?

Counting operations is an oversimplified cost model: ignores communication costs, parallelism. See, e.g., 1981 Brent–Kung *AT* theorem for realistic chip model.

NFS suffers somewhat from communication costs inside big linear-algebra subroutine.

2001 Bernstein:
$AT = L^{p'+o(1)}$ with $p' \approx 1.976$.

2017 Bernstein–Biasse–Mosca:
$AT = L^{q'+o(1)}$ with $q' \approx 1.456$ using $b^{2/3+o(1)}$ qubits.
Open: Analyze for $b = 2048$.

Actually have man
Lower cost for *som*
Lower cost for *ma*

ns

> 1.9,

$)^{2/3}$.

easy!):

s.

ca:

ts).

3.

Counting operations is an oversimplified cost model: ignores communication costs, parallelism. See, e.g., 1981 Brent–Kung *AT* theorem for realistic chip model.

NFS suffers somewhat from communication costs inside big linear-algebra subroutine.

2001 Bernstein:
$AT = L^{p'+o(1)}$ with $p' \approx 1.976$.

2017 Bernstein–Biasse–Mosca:
$AT = L^{q'+o(1)}$ with $q' \approx 1.456$ using $b^{2/3+o(1)}$ qubits.
Open: Analyze for $b = 2048$.

Actually have many inputs.
Lower cost for *some* output
Lower cost for *many* output

Counting operations is an oversimplified cost model: ignores communication costs, parallelism. See, e.g., 1981 Brent–Kung *AT* theorem for realistic chip model.

NFS suffers somewhat from communication costs inside big linear-algebra subroutine.

2001 Bernstein:
$AT = L^{p'+o(1)}$ with $p' \approx 1.976$.

2017 Bernstein–Biasse–Mosca:
$AT = L^{q'+o(1)}$ with $q' \approx 1.456$ using $b^{2/3+o(1)}$ qubits.
Open: Analyze for $b = 2048$.

Actually have many inputs.
Lower cost for *some* output?
Lower cost for *many* outputs?

Counting operations is an oversimplified cost model: ignores communication costs, parallelism. See, e.g., 1981 Brent–Kung AT theorem for realistic chip model.

NFS suffers somewhat from communication costs inside big linear-algebra subroutine.

2001 Bernstein:
$AT = L^{p'+o(1)}$ with $p' \approx 1.976$.

2017 Bernstein–Biasse–Mosca:
$AT = L^{q'+o(1)}$ with $q' \approx 1.456$
using $b^{2/3+o(1)}$ qubits.
Open: Analyze for $b = 2048$.

Actually have many inputs.
Lower cost for *some* output?
Lower cost for *many* outputs?

1993 Coppersmith:
$L^{1.638\ldots+o(1)}$ operations
after precomp(b) involving
$L^{2.006\ldots+o(1)}$ operations.

Counting operations is an oversimplified cost model: ignores communication costs, parallelism. See, e.g., 1981 Brent–Kung *AT* theorem for realistic chip model.

NFS suffers somewhat from communication costs inside big linear-algebra subroutine.

2001 Bernstein:
$AT = L^{p'+o(1)}$ with $p' \approx 1.976$.

2017 Bernstein–Biasse–Mosca:
$AT = L^{q'+o(1)}$ with $q' \approx 1.456$ using $b^{2/3+o(1)}$ qubits.
Open: Analyze for $b = 2048$.

Actually have many inputs.
Lower cost for *some* output?
Lower cost for *many* outputs?

1993 Coppersmith:
$L^{1.638\ldots+o(1)}$ operations after precomp(b) involving $L^{2.006\ldots+o(1)}$ operations.

2014 Bernstein–Lange:
$AT = L^{2.204\ldots+o(1)}$ to factor $L^{0.5+o(1)}$ inputs; $L^{1.704\ldots+o(1)}$ per input.

Counting operations is an oversimplified cost model: ignores communication costs, parallelism. See, e.g., 1981 Brent–Kung *AT* theorem for realistic chip model.

NFS suffers somewhat from communication costs inside big linear-algebra subroutine.

2001 Bernstein:
$AT = L^{p'+o(1)}$ with $p' \approx 1.976$.

2017 Bernstein–Biasse–Mosca:
$AT = L^{q'+o(1)}$ with $q' \approx 1.456$ using $b^{2/3+o(1)}$ qubits.
Open: Analyze for $b = 2048$.

Actually have many inputs.
Lower cost for *some* output?
Lower cost for *many* outputs?

1993 Coppersmith:
$L^{1.638\ldots+o(1)}$ operations after precomp(b) involving $L^{2.006\ldots+o(1)}$ operations.

2014 Bernstein–Lange:
$AT = L^{2.204\ldots+o(1)}$ to factor $L^{0.5+o(1)}$ inputs; $L^{1.704\ldots+o(1)}$ per input.

Open: Any quantum speedups for factoring many integers?

g operations is an
plified cost model: ignores
nication costs, parallelism.
., 1981 Brent–Kung *AT*
for realistic chip model.

fers somewhat from
nication costs inside
r-algebra subroutine.

rnstein:
$^{p'+o(1)}$ with $p' \approx 1.976$.

rnstein–Biasse–Mosca:
$^{q'+o(1)}$ with $q' \approx 1.456$
$^{/3+o(1)}$ qubits.

Analyze for $b = 2048$.

Actually have many inputs.
Lower cost for *some* output?
Lower cost for *many* outputs?

1993 Coppersmith:
$L^{1.638\ldots+o(1)}$ operations
after precomp(b) involving
$L^{2.006\ldots+o(1)}$ operations.

2014 Bernstein–Lange:
$AT = L^{2.204\ldots+o(1)}$
to factor $L^{0.5+o(1)}$ inputs;
$L^{1.704\ldots+o(1)}$ per input.

Open: Any quantum speedups
for factoring many integers?

Long-ter

Long his
in intege

Long his
switchin
not far b

ns is an
t model: ignores
sts, parallelism.
ent–Kung AT
tic chip model.

what from
sts inside
subroutine.

th $p' \approx 1.976$.

asse–Mosca:
th $q' \approx 1.456$
ubits.
r $b = 2048$.

Actually have many inputs.
Lower cost for *some* output?
Lower cost for *many* outputs?

1993 Coppersmith:
$L^{1.638\ldots+o(1)}$ operations
after precomp(b) involving
$L^{2.006\ldots+o(1)}$ operations.

2014 Bernstein–Lange:
$AT = L^{2.204\ldots+o(1)}$
to factor $L^{0.5+o(1)}$ inputs;
$L^{1.704\ldots+o(1)}$ per input.

Open: Any quantum speedups
for factoring many integers?

Long-term RSA se

Long history of ad
in integer factoriza

Long history of RS
switching to large
not far beyond bro

gnores
elism.
 AT
odel.

.

976.

ca:
456

3.

Actually have many inputs.
Lower cost for *some* output?
Lower cost for *many* outputs?

1993 Coppersmith:
$L^{1.638\ldots+o(1)}$ operations
after precomp(b) involving
$L^{2.006\ldots+o(1)}$ operations.

2014 Bernstein–Lange:
$AT = L^{2.204\ldots+o(1)}$
to factor $L^{0.5+o(1)}$ inputs;
$L^{1.704\ldots+o(1)}$ per input.

Open: Any quantum speedups
for factoring many integers?

Long-term RSA security

Long history of advances
in integer factorization.

Long history of RSA users
switching to larger key sizes
not far beyond broken sizes.

Actually have many inputs.
Lower cost for *some* output?
Lower cost for *many* outputs?

1993 Coppersmith:
$L^{1.638...+o(1)}$ operations
after precomp(b) involving
$L^{2.006...+o(1)}$ operations.

2014 Bernstein–Lange:
$AT = L^{2.204...+o(1)}$
to factor $L^{0.5+o(1)}$ inputs;
$L^{1.704...+o(1)}$ per input.

Open: Any quantum speedups
for factoring many integers?

Long-term RSA security

Long history of advances
in integer factorization.

Long history of RSA users
switching to larger key sizes,
not far beyond broken sizes.

Actually have many inputs.
Lower cost for *some* output?
Lower cost for *many* outputs?

1993 Coppersmith:
$L^{1.638\ldots+o(1)}$ operations
after precomp(b) involving
$L^{2.006\ldots+o(1)}$ operations.

2014 Bernstein–Lange:
$AT = L^{2.204\ldots+o(1)}$
to factor $L^{0.5+o(1)}$ inputs;
$L^{1.704\ldots+o(1)}$ per input.

Open: Any quantum speedups
for factoring many integers?

Long-term RSA security

Long history of advances
in integer factorization.

Long history of RSA users
switching to larger key sizes,
not far beyond broken sizes.

"Expert" cryptographers:
"Obviously they won't react to
Shor's algorithm this way! They'll
switch to codes, lattices, etc. long
before quantum computers break
RSA-2048! We don't need to
analyze the security of RSA-4096,
RSA-8192, RSA-16384, etc.!"

have many inputs.
ost for *some* output?
ost for *many* outputs?

ppersmith:
$-o(1)$ operations
ecomp(b) involving
$-o(1)$ operations.

rnstein–Lange:
$2.204\ldots+o(1)$
$L^{0.5+o(1)}$ inputs;
$-o(1)$ per input.

Any quantum speedups
oring many integers?

Long-term RSA security

Long history of advances in integer factorization.

Long history of RSA users switching to larger key sizes, not far beyond broken sizes.

"Expert" cryptographers: "Obviously they won't react to Shor's algorithm this way! They'll switch to codes, lattices, etc. long before quantum computers break RSA-2048! We don't need to analyze the security of RSA-4096, RSA-8192, RSA-16384, etc.!"

We cons
quantum
we also
of users

ny inputs.
me output?
ny outputs?

:

ations
involving
ations.

ange:
)

' inputs;
nput.

um speedups
y integers?

Long-term RSA security

Long history of advances in integer factorization.

Long history of RSA users switching to larger key sizes, not far beyond broken sizes.

"Expert" cryptographers: "Obviously they won't react to Shor's algorithm this way! They'll switch to codes, lattices, etc. long before quantum computers break RSA-2048! We don't need to analyze the security of RSA-4096, RSA-8192, RSA-16384, etc.!"

We consider possil
quantum compute
we also consider p
of users wanting t

?

s?

ups

Long-term RSA security

Long history of advances in integer factorization.

Long history of RSA users switching to larger key sizes, not far beyond broken sizes.

“Expert” cryptographers: “Obviously they won’t react to Shor’s algorithm this way! They’ll switch to codes, lattices, etc. long before quantum computers break RSA-2048! We don’t need to analyze the security of RSA-4096, RSA-8192, RSA-16384, etc.!”

We consider possible impact
quantum computers. Shoulc
we also consider possible im
of users wanting to stick to

Long-term RSA security

Long history of advances in integer factorization.

Long history of RSA users switching to larger key sizes, not far beyond broken sizes.

"Expert" cryptographers: "Obviously they won't react to Shor's algorithm this way! They'll switch to codes, lattices, etc. long before quantum computers break RSA-2048! We don't need to analyze the security of RSA-4096, RSA-8192, RSA-16384, etc.!"

We consider possible impact of quantum computers. Shouldn't we also consider possible impact of users wanting to stick to RSA?

Long-term RSA security

Long history of advances in integer factorization.

Long history of RSA users switching to larger key sizes, not far beyond broken sizes.

"Expert" cryptographers: "Obviously they won't react to Shor's algorithm this way! They'll switch to codes, lattices, etc. long before quantum computers break RSA-2048! We don't need to analyze the security of RSA-4096, RSA-8192, RSA-16384, etc.!"

We consider possible impact of quantum computers. Shouldn't we also consider possible impact of users wanting to stick to RSA?

2017 Bernstein–Heninger–Lou–Valenta "Post-quantum RSA" (pqRSA): Generated 1-terabyte RSA key; 2000000 core-hours. Shor's algorithm: $>2^{100}$ gates.

Long-term RSA security

Long history of advances in integer factorization.

Long history of RSA users switching to larger key sizes, not far beyond broken sizes.

"Expert" cryptographers: "Obviously they won't react to Shor's algorithm this way! They'll switch to codes, lattices, etc. long before quantum computers break RSA-2048! We don't need to analyze the security of RSA-4096, RSA-8192, RSA-16384, etc.!"

We consider possible impact of quantum computers. Shouldn't we also consider possible impact of users wanting to stick to RSA?

2017 Bernstein–Heninger–Lou–Valenta "Post-quantum RSA" (pqRSA): Generated 1-terabyte RSA key; 2000000 core-hours. Shor's algorithm: $>2^{100}$ gates.

Bernstein–Fried–Heninger–Lou–Valenta: Draft NIST submission proposing 1-gigabyte RSA keys. Much faster to generate.

rm RSA security

story of advances
er factorization.

story of RSA users
g to larger key sizes,
beyond broken sizes.

' cryptographers:
sly they won't react to
lgorithm this way! They'll
o codes, lattices, etc. long
uantum computers break
48! We don't need to
the security of RSA-4096,
92, RSA-16384, etc.!"

We consider possible impact of quantum computers. Shouldn't we also consider possible impact of users wanting to stick to RSA?

2017 Bernstein–Heninger–Lou–Valenta "Post-quantum RSA" (pqRSA): Generated 1-terabyte RSA key; 2000000 core-hours. Shor's algorithm: $>2^{100}$ gates.

Bernstein–Fried–Heninger–Lou–Valenta: Draft NIST submission proposing 1-gigabyte RSA keys. Much faster to generate.

The secr
4096 bit
1024 bit
Importa
keygen,

Is this a

ECM fin
using $L^{\sqrt{}}$
where lo
Beats Sh
(log log r

Public E
274-bit f

ecurity

vances
ation.

SA users
r key sizes,
oken sizes.

aphers:
von't react to
his way! They'll
attices, etc. long
omputers break
on't need to
ty of RSA-4096,
6384, etc.!"

We consider possible impact of quantum computers. Shouldn't we also consider possible impact of users wanting to stick to RSA?

2017 Bernstein–Heninger–Lou–Valenta "Post-quantum RSA" (pqRSA): Generated 1-terabyte RSA key; 2000000 core-hours. Shor's algorithm: $>2^{100}$ gates.

Bernstein–Fried–Heninger–Lou–Valenta: Draft NIST submission proposing 1-gigabyte RSA keys. Much faster to generate.

The secret primes
4096 bits in teraby
1024 bits in gigaby
Important time-sa
keygen, signing, de

Is this a weakness

ECM finds any pri
using $L^{\sqrt{2}+o(1)}$ mu
where $\log L =$ (log
Beats Shor for log
(log log modulus)^{2}

Public ECM record
274-bit factor of 7

’

to
They’ll
c. long
break
to
-4096,
!”

We consider possible impact of quantum computers. Shouldn’t we also consider possible impact of users wanting to stick to RSA?

2017 Bernstein–Heninger–Lou–Valenta “Post-quantum RSA” (pqRSA): Generated 1-terabyte RSA key; 2000000 core-hours. Shor’s algorithm: $>2^{100}$ gates.

Bernstein–Fried–Heninger–Lou–Valenta: Draft NIST submission proposing 1-gigabyte RSA keys. Much faster to generate.

The secret primes are small: 4096 bits in terabyte key; 1024 bits in gigabyte key. Important time-saver in keygen, signing, decryption.

Is this a weakness?

ECM finds any prime $<y$ using $L^{\sqrt{2}+o(1)}$ mulmods, where $\log L = (\log y \log\log y$
Beats Shor for $\log y$ below $(\log\log \text{modulus})^{2+o(1)}$.

Public ECM record: 274-bit factor of $7^{337}+1$.

We consider possible impact of quantum computers. Shouldn't we also consider possible impact of users wanting to stick to RSA?

2017 Bernstein–Heninger–Lou–Valenta "Post-quantum RSA" (pqRSA): Generated 1-terabyte RSA key; 2000000 core-hours. Shor's algorithm: $>2^{100}$ gates.

Bernstein–Fried–Heninger–Lou–Valenta: Draft NIST submission proposing 1-gigabyte RSA keys. Much faster to generate.

The secret primes are small:
4096 bits in terabyte key;
1024 bits in gigabyte key.
Important time-saver in keygen, signing, decryption.

Is this a weakness?

ECM finds any prime $<y$ using $L^{\sqrt{2}+o(1)}$ mulmods, where $\log L = (\log y \log\log y)^{1/2}$. Beats Shor for $\log y$ below $(\log\log \text{modulus})^{2+o(1)}$.

Public ECM record:
274-bit factor of $7^{337}+1$.

sider possible impact of
n computers. Shouldn't
consider possible impact
wanting to stick to RSA?

rnstein–Heninger–Lou–
"Post-quantum RSA"
): Generated 1-terabyte
; 2000000 core-hours.
lgorithm: $>2^{100}$ gates.

n–Fried–Heninger–Lou–
Draft NIST submission
g 1-gigabyte RSA keys.
ster to generate.

The secret primes are small:
4096 bits in terabyte key;
1024 bits in gigabyte key.
Important time-saver in
keygen, signing, decryption.

Is this a weakness?

ECM finds any prime $<y$
using $L^{\sqrt{2}+o(1)}$ mulmods,
where $\log L = (\log y \log\log y)^{1/2}$.
Beats Shor for $\log y$ below
$(\log\log \text{modulus})^{2+o(1)}$.

Public ECM record:
274-bit factor of $7^{337}+1$.

Analysis
$>2^{125}$ m
and 2^{33}-

2^{23} targ
finding j

ble impact of
rs. Shouldn't
ossible impact
o stick to RSA?

eninger–Lou–
ntum RSA"
ed 1-terabyte
) core-hours.
$>2^{100}$ gates.

leninger–Lou–
ST submission
yte RSA keys.
nerate.

The secret primes are small:
4096 bits in terabyte key;
1024 bits in gigabyte key.
Important time-saver in
keygen, signing, decryption.

Is this a weakness?

ECM finds any prime $<y$
using $L^{\sqrt{2}+o(1)}$ mulmods,
where $\log L = (\log y \log\log y)^{1/2}$.
Beats Shor for $\log y$ below
$(\log\log \text{modulus})^{2+o(1)}$.

Public ECM record:
274-bit factor of $7^{337}+1$.

Analysis for $y \approx 2$
$>2^{125}$ mulmods, h
and 2^{33}-bit mulmo

2^{23} target primes,
finding just one isn

t of
dn't
pact
RSA?

ou–
A"
yte
rs.
es.

ou–
ssion
eys.

The secret primes are small:
4096 bits in terabyte key;
1024 bits in gigabyte key.
Important time-saver in
keygen, signing, decryption.

Is this a weakness?

ECM finds any prime $<y$
using $L^{\sqrt{2}+o(1)}$ mulmods,
where $\log L = (\log y \log\log y)^{1/2}$.
Beats Shor for $\log y$ below
$(\log\log \text{modulus})^{2+o(1)}$.

Public ECM record:
274-bit factor of $7^{337}+1$.

Analysis for $y \approx 2^{1024}$:
$>2^{125}$ mulmods, huge depth
and 2^{33}-bit mulmod is slow.

2^{23} target primes, but
finding just one isn't enough

The secret primes are small:
4096 bits in terabyte key;
1024 bits in gigabyte key.
Important time-saver in
keygen, signing, decryption.

Is this a weakness?

ECM finds any prime $<y$
using $L^{\sqrt{2}+o(1)}$ mulmods,
where $\log L = (\log y \log\log y)^{1/2}$.
Beats Shor for $\log y$ below
$(\log\log \text{modulus})^{2+o(1)}$.

Public ECM record:
274-bit factor of $7^{337}+1$.

Analysis for $y \approx 2^{1024}$:
$>2^{125}$ mulmods, huge depth;
and 2^{33}-bit mulmod is slow.

2^{23} target primes, but
finding just one isn't enough.

The secret primes are small:
4096 bits in terabyte key;
1024 bits in gigabyte key.
Important time-saver in
keygen, signing, decryption.

Is this a weakness?

ECM finds any prime $<y$
using $L^{\sqrt{2}+o(1)}$ mulmods,
where $\log L = (\log y \log\log y)^{1/2}$.
Beats Shor for $\log y$ below
$(\log\log \text{modulus})^{2+o(1)}$.

Public ECM record:
274-bit factor of $7^{337}+1$.

Analysis for $y \approx 2^{1024}$:
$>2^{125}$ mulmods, huge depth;
and 2^{33}-bit mulmod is slow.

2^{23} target primes, but
finding just one isn't enough.

2017 Bernstein–Heninger–Lou–
Valenta: Grover+ECM
finds any prime $<y$
using $L^{1+o(1)}$ mulmods.

The secret primes are small:
4096 bits in terabyte key;
1024 bits in gigabyte key.
Important time-saver in
keygen, signing, decryption.

Is this a weakness?

ECM finds any prime $<y$
using $L^{\sqrt{2}+o(1)}$ mulmods,
where $\log L = (\log y \log\log y)^{1/2}$.
Beats Shor for $\log y$ below
$(\log\log \text{modulus})^{2+o(1)}$.

Public ECM record:
274-bit factor of $7^{337}+1$.

Analysis for $y \approx 2^{1024}$:
$>2^{125}$ mulmods, huge depth;
and 2^{33}-bit mulmod is slow.

2^{23} target primes, but
finding just one isn't enough.

2017 Bernstein–Heninger–Lou–
Valenta: Grover+ECM
finds any prime $<y$
using $L^{1+o(1)}$ mulmods.

Seems swamped by overhead.

Open: Better ways for quantum
algorithms to find small factors?

ret primes are small:
s in terabyte key;
s in gigabyte key.
nt time-saver in
signing, decryption.

weakness?

ds any prime $<y$
$^{\sqrt{2}+o(1)}$ mulmods,
g $L = (\log y \log\log y)^{1/2}$.
hor for $\log y$ below
modulus)$^{2+o(1)}$.

ECM record:
factor of $7^{337} + 1$.

Analysis for $y \approx 2^{1024}$:
$>2^{125}$ mulmods, huge depth;
and 2^{33}-bit mulmod is slow.

2^{23} target primes, but
finding just one isn't enough.

2017 Bernstein–Heninger–Lou–Valenta: Grover+ECM
finds any prime $<y$
using $L^{1+o(1)}$ mulmods.

Seems swamped by overhead.

Open: Better ways for quantum algorithms to find small factors?

Minimur
NIST all
submissi
search fo

Is a giga
Shor's a

are small:
yte key;
yte key.
ver in
ecryption.

?

me $<y$
ulmods,
$g\, y \log\log y)^{1/2}$.
y below
$+o(1)$.

d:
$337 + 1$.

Analysis for $y \approx 2^{1024}$:
$>2^{125}$ mulmods, huge depth;
and 2^{33}-bit mulmod is slow.

2^{23} target primes, but
finding just one isn't enough.

2017 Bernstein–Heninger–Lou–Valenta: Grover+ECM
finds any prime $<y$
using $L^{1+o(1)}$ mulmods.

Seems swamped by overhead.

Open: Better ways for quantum algorithms to find small factors?

Minimum security
NIST allows for po
submissions: brute
search for a 128-b

Is a gigabyte key s
Shor's algorithm t

$y)^{1/2}$.

Analysis for $y \approx 2^{1024}$:
$>2^{125}$ mulmods, huge depth;
and 2^{33}-bit mulmod is slow.

2^{23} target primes, but
finding just one isn't enough.

2017 Bernstein–Heninger–Lou–Valenta: Grover+ECM
finds any prime $<y$
using $L^{1+o(1)}$ mulmods.

Seems swamped by overhead.

Open: Better ways for quantum algorithms to find small factors?

Minimum security level that
NIST allows for post-quantu
submissions: brute-force/Gr
search for a 128-bit AES key

Is a gigabyte key so difficult
Shor's algorithm to break?

Analysis for $y \approx 2^{1024}$:
$>2^{125}$ mulmods, huge depth;
and 2^{33}-bit mulmod is slow.

2^{23} target primes, but
finding just one isn't enough.

2017 Bernstein–Heninger–Lou–Valenta: Grover+ECM
finds any prime $<y$
using $L^{1+o(1)}$ mulmods.

Seems swamped by overhead.

Open: Better ways for quantum algorithms to find small factors?

Minimum security level that NIST allows for post-quantum submissions: brute-force/Grover search for a 128-bit AES key.

Is a gigabyte key so difficult for Shor's algorithm to break?

Analysis for $y \approx 2^{1024}$:
$>2^{125}$ mulmods, huge depth;
and 2^{33}-bit mulmod is slow.

2^{23} target primes, but
finding just one isn't enough.

2017 Bernstein–Heninger–Lou–Valenta: Grover+ECM
finds any prime $<y$
using $L^{1+o(1)}$ mulmods.

Seems swamped by overhead.

Open: Better ways for quantum algorithms to find small factors?

Minimum security level that NIST allows for post-quantum submissions: brute-force/Grover search for a 128-bit AES key.

Is a gigabyte key so difficult for Shor's algorithm to break?

$64b^3 \lg b \approx 2^{110}$ for $b = 2^{33}$.

Not totally implausible to argue that Grover's algorithm could break AES-128 faster than this.

Analysis for $y \approx 2^{1024}$:
$>2^{125}$ mulmods, huge depth;
and 2^{33}-bit mulmod is slow.

2^{23} target primes, but
finding just one isn't enough.

2017 Bernstein–Heninger–Lou–Valenta: Grover+ECM
finds any prime $<y$
using $L^{1+o(1)}$ mulmods.

Seems swamped by overhead.

Open: Better ways for quantum algorithms to find small factors?

Minimum security level that NIST allows for post-quantum submissions: brute-force/Grover search for a 128-bit AES key.

Is a gigabyte key so difficult for Shor's algorithm to break?

$64b^3 \lg b \approx 2^{110}$ for $b = 2^{33}$.

Not totally implausible to argue that Grover's algorithm could break AES-128 faster than this.

But Shor's algorithm can (with more qubits) use faster mulmods.

for $y \approx 2^{1024}$:
ulmods, huge depth;
-bit mulmod is slow.

et primes, but
ust one isn't enough.

rnstein–Heninger–Lou–
Grover+ECM
y prime $<y$
$^{+o(1)}$ mulmods.

wamped by overhead.

Better ways for quantum
ms to find small factors?

Minimum security level that NIST allows for post-quantum submissions: brute-force/Grover search for a 128-bit AES key.

Is a gigabyte key so difficult for Shor's algorithm to break?

$64b^3 \lg b \approx 2^{110}$ for $b = 2^{33}$.

Not totally implausible to argue that Grover's algorithm could break AES-128 faster than this.

But Shor's algorithm can (with more qubits) use faster mulmods.

NIST all
assume

“Plausib
range fr
approxin
presently
computi
expected
a year) t
(the app
that cur
architect
in a deca
logical g

^{1024}:
huge depth;
od is slow.

but
n't enough.

eninger–Lou–
ECM
y
mods.

y overhead.

s for quantum
small factors?

Minimum security level that NIST allows for post-quantum submissions: brute-force/Grover search for a 128-bit AES key.

Is a gigabyte key so difficult for Shor's algorithm to break?

$64b^3 \lg b \approx 2^{110}$ for $b = 2^{33}$.

Not totally implausible to argue that Grover's algorithm could break AES-128 faster than this.

But Shor's algorithm can (with more qubits) use faster mulmods.

NIST allows subm
assume reasonable

"Plausible values f
range from 2^{40} log
approximate numb
presently envisione
computing archite
expected to seriall
a year) through 2^6
(the approximate r
that current classi
architectures can p
in a decade), to no
logical gates . . . "

n;

n.

ou–

d.

tum
tors?

Minimum security level that NIST allows for post-quantum submissions: brute-force/Grover search for a 128-bit AES key.

Is a gigabyte key so difficult for Shor's algorithm to break?

$64 b^3 \lg b \approx 2^{110}$ for $b = 2^{33}$.

Not totally implausible to argue that Grover's algorithm could break AES-128 faster than this.

But Shor's algorithm can (with more qubits) use faster mulmods.

NIST allows submissions to
assume reasonable time limi

"Plausible values for MAXD
range from 2^{40} logical gates
approximate number of gate
presently envisioned quantu
computing architectures are
expected to serially perform
a year) through 2^{64} logical g
(the approximate number of
that current classical compu
architectures can perform se
in a decade), to no more tha
logical gates . . ."

Minimum security level that NIST allows for post-quantum submissions: brute-force/Grover search for a 128-bit AES key.

Is a gigabyte key so difficult for Shor's algorithm to break?

$64b^3 \lg b \approx 2^{110}$ for $b = 2^{33}$.

Not totally implausible to argue that Grover's algorithm could break AES-128 faster than this.

But Shor's algorithm can (with more qubits) use faster mulmods.

NIST allows submissions to assume reasonable time limits:

"Plausible values for MAXDEPTH range from 2^{40} logical gates (the approximate number of gates that presently envisioned quantum computing architectures are expected to serially perform in a year) through 2^{64} logical gates (the approximate number of gates that current classical computing architectures can perform serially in a decade), to no more than 2^{96} logical gates . . . "

m security level that
ows for post-quantum
ons: brute-force/Grover
or a 128-bit AES key.

byte key so difficult for
lgorithm to break?

$\approx 2^{110}$ for $b = 2^{33}$.

ally implausible to argue
over's algorithm could
ES-128 faster than this.

r's algorithm can (with
bits) use faster mulmods.

NIST allows submissions to assume reasonable time limits:

"Plausible values for MAXDEPTH range from 2^{40} logical gates (the approximate number of gates that presently envisioned quantum computing architectures are expected to serially perform in a year) through 2^{64} logical gates (the approximate number of gates that current classical computing architectures can perform serially in a decade), to no more than 2^{96} logical gates . . ."

What is
for b-bit

Light tal
to cross

1981 Br
$AT \geq$ sr
even if w

(Work a
faster-th
through
Haven't
even if r
avoids F

level that
ost-quantum
e-force/Grover
it AES key.

so difficult for
o break?

or $b = 2^{33}$.

sible to argue
rithm could
ster than this.

nm can (with
faster mulmods.

NIST allows submissions to assume reasonable time limits:

“Plausible values for MAXDEPTH range from 2^{40} logical gates (the approximate number of gates that presently envisioned quantum computing architectures are expected to serially perform in a year) through 2^{64} logical gates (the approximate number of gates that current classical computing architectures can perform serially in a decade), to no more than 2^{96} logical gates . . . ”

What is the minim
for b-bit integer m

Light takes time Ω
to cross a $b^{1/2} \times$

1981 Brent–Kung
$AT \geq$ small consta
even if wire latenc

(Work around obs
faster-than-light c
through long-dista
Haven’t seen plaus
even if reversible c
avoids FTL imposs

um
over
y.
for
.
rgue
d
his.
ith
mods.

NIST allows submissions to assume reasonable time limits:

"Plausible values for MAXDEPTH range from 2^{40} logical gates (the approximate number of gates that presently envisioned quantum computing architectures are expected to serially perform in a year) through 2^{64} logical gates (the approximate number of gates that current classical computing architectures can perform serially in a decade), to no more than 2^{96} logical gates . . . "

What is the minimum time
for b-bit integer multiplicati

Light takes time $\Omega(b^{1/2})$
to cross a $b^{1/2} \times b^{1/2}$ chip.

1981 Brent–Kung AT theore
$AT \geq$ small constant $\cdot\, b^{3/2}$,
even if wire latency is 0.

(Work around obstacles usin
faster-than-light communica
through long-distance EPR
Haven't seen plausible desig
even if reversible computatio
avoids FTL impossibility pro

NIST allows submissions to assume reasonable time limits:

"Plausible values for MAXDEPTH range from 2^{40} logical gates (the approximate number of gates that presently envisioned quantum computing architectures are expected to serially perform in a year) through 2^{64} logical gates (the approximate number of gates that current classical computing architectures can perform serially in a decade), to no more than 2^{96} logical gates . . ."

What is the minimum time for b-bit integer multiplication?

Light takes time $\Omega(b^{1/2})$ to cross a $b^{1/2} \times b^{1/2}$ chip.

1981 Brent–Kung AT theorem: $AT \geq$ small constant $\cdot\, b^{3/2}$, even if wire latency is 0.

(Work around obstacles using faster-than-light communication through long-distance EPR pairs? Haven't seen plausible designs, even if reversible computation avoids FTL impossibility proofs.)

ows submissions to
reasonable time limits:

le values for MAXDEPTH
om 2^{40} logical gates (the
nate number of gates that
y envisioned quantum
ng architectures are
d to serially perform in
through 2^{64} logical gates
proximate number of gates
rent classical computing
tures can perform serially
ade), to no more than 2^{96}
ates . . . ”

What is the minimum time
for b-bit integer multiplication?

Light takes time $\Omega(b^{1/2})$
to cross a $b^{1/2} \times b^{1/2}$ chip.

1981 Brent–Kung AT theorem:
$AT \geq$ small constant $\cdot\, b^{3/2}$,
even if wire latency is 0.

(Work around obstacles using
faster-than-light communication
through long-distance EPR pairs?
Haven’t seen plausible designs,
even if reversible computation
avoids FTL impossibility proofs.)

What is
for Shor

Main bo
for $2b$-b

Traditio
controlle
a and 1/
a^2 mod
a^4 mod

Can mul
using ma
but hard
computa

issions to
e time limits:

for MAXDEPTH
gical gates (the
ber of gates that
ed quantum
ctures are
y perform in
^{64} logical gates
number of gates
cal computing
perform serially
o more than 2^{96}

What is the minimum time
for b-bit integer multiplication?

Light takes time $\Omega(b^{1/2})$
to cross a $b^{1/2} \times b^{1/2}$ chip.

1981 Brent–Kung AT theorem:
$AT \geq$ small constant $\cdot\, b^{3/2}$,
even if wire latency is 0.

(Work around obstacles using
faster-than-light communication
through long-distance EPR pairs?
Haven't seen plausible designs,
even if reversible computation
avoids FTL impossibility proofs.)

What is the minim
for Shor's algorith

Main bottleneck:
for $2b$-bit superpo

Traditional approa
controlled multipli
a and $1/a$ mod N;
a^2 mod N and $1/a$
a^4 mod N and $1/a$

Can multiply these
using many more
but hard to paralle
computation of a^2

ts:

EPTH
(the
es that
m

in
gates
f gates
ting
erially
an 2^{96}

What is the minimum time
for b-bit integer multiplication?

Light takes time $\Omega(b^{1/2})$
to cross a $b^{1/2} \times b^{1/2}$ chip.

1981 Brent–Kung AT theorem:
$AT \geq$ small constant $\cdot\, b^{3/2}$,
even if wire latency is 0.

(Work around obstacles using
faster-than-light communication
through long-distance EPR pairs?
Haven't seen plausible designs,
even if reversible computation
avoids FTL impossibility proofs.)

What is the minimum time
for Shor's algorithm?

Main bottleneck: a^e mod N
for $2b$-bit superposition e.

Traditional approach: series
controlled multiplications by
a and $1/a$ mod N;
a^2 mod N and $1/a^2$ mod N;
a^4 mod N and $1/a^4$ mod N;

Can multiply these in paralle
using many more qubits;
but hard to parallelize initial
computation of a^{2^i} mod N.

What is the minimum time for b-bit integer multiplication?

Light takes time $\Omega(b^{1/2})$ to cross a $b^{1/2} \times b^{1/2}$ chip.

1981 Brent–Kung AT theorem: $AT \geq$ small constant $\cdot\, b^{3/2}$, even if wire latency is 0.

(Work around obstacles using faster-than-light communication through long-distance EPR pairs? Haven't seen plausible designs, even if reversible computation avoids FTL impossibility proofs.)

What is the minimum time for Shor's algorithm?

Main bottleneck: $a^e \bmod N$ for $2b$-bit superposition e.

Traditional approach: series of controlled multiplications by
a and $1/a \bmod N$;
$a^2 \bmod N$ and $1/a^2 \bmod N$;
$a^4 \bmod N$ and $1/a^4 \bmod N$; etc.

Can multiply these in parallel, using many more qubits; but hard to parallelize initial computation of $a^{2^i} \bmod N$.

the minimum time
integer multiplication?

kes time $\Omega(b^{1/2})$
a $b^{1/2} \times b^{1/2}$ chip.

ent–Kung AT theorem:
mall constant $\cdot\, b^{3/2}$,
vire latency is 0.

round obstacles using
an-light communication
long-distance EPR pairs?
seen plausible designs,
eversible computation
TL impossibility proofs.)

What is the minimum time
for Shor's algorithm?

Main bottleneck: $a^e \bmod N$
for $2b$-bit superposition e.

Traditional approach: series of
controlled multiplications by
a and $1/a \bmod N$;
$a^2 \bmod N$ and $1/a^2 \bmod N$;
$a^4 \bmod N$ and $1/a^4 \bmod N$; etc.

Can multiply these in parallel,
using many more qubits;
but hard to parallelize initial
computation of $a^{2^i} \bmod N$.

Why gig
big enou
beyond t
under re

Gigabyte
millions
than 204
These al
billions o
More co

num time
ultiplication?

$2(b^{1/2})$
$b^{1/2}$ chip.

AT theorem:
ant $\cdot$ $b^{3/2}$,
y is 0.

tacles using
ommunication
nce EPR pairs?
sible designs,
computation
sibility proofs.)

What is the minimum time
for Shor's algorithm?

Main bottleneck: a^e mod N
for $2b$-bit superposition e.

Traditional approach: series of
controlled multiplications by
a and $1/a$ mod N;
a^2 mod N and $1/a^2$ mod N;
a^4 mod N and $1/a^4$ mod N; etc.

Can multiply these in parallel,
using many more qubits;
but hard to parallelize initial
computation of a^{2^i} mod N.

Why gigabyte keys
big enough to pus
beyond the 2^{64} lim
under reasonable a

Gigabyte inputs ar
millions of times la
than 2048-bit inpu
These algorithms
billions of times lo
More cost to find

on?

em:

g
tion
pairs?
ns,
on
ofs.)

What is the minimum time
for Shor's algorithm?

Main bottleneck: $a^e \bmod N$
for $2b$-bit superposition e.

Traditional approach: series of
controlled multiplications by
a and $1/a \bmod N$;
$a^2 \bmod N$ and $1/a^2 \bmod N$;
$a^4 \bmod N$ and $1/a^4 \bmod N$; etc.

Can multiply these in parallel,
using many more qubits;
but hard to parallelize initial
computation of $a^{2^i} \bmod N$.

Why gigabyte keys are reaso
big enough to push latency
beyond the 2^{64} limit,
under reasonable assumptio

Gigabyte inputs are
millions of times larger
than 2048-bit inputs.
These algorithms will take
billions of times longer.
More cost to find *all* primes

What is the minimum time for Shor's algorithm?

Main bottleneck: $a^e \bmod N$ for $2b$-bit superposition e.

Traditional approach: series of controlled multiplications by
a and $1/a \bmod N$;
$a^2 \bmod N$ and $1/a^2 \bmod N$;
$a^4 \bmod N$ and $1/a^4 \bmod N$; etc.

Can multiply these in parallel, using many more qubits; but hard to parallelize initial computation of $a^{2^i} \bmod N$.

Why gigabyte keys are reasonable: big enough to push latency beyond the 2^{64} limit, under reasonable assumptions.

Gigabyte inputs are millions of times larger than 2048-bit inputs.
These algorithms will take billions of times longer.
More cost to find *all* primes.

What is the minimum time for Shor's algorithm?

Main bottleneck: $a^e \bmod N$ for $2b$-bit superposition e.

Traditional approach: series of controlled multiplications by
a and $1/a \bmod N$;
$a^2 \bmod N$ and $1/a^2 \bmod N$;
$a^4 \bmod N$ and $1/a^4 \bmod N$; etc.

Can multiply these in parallel, using many more qubits; but hard to parallelize initial computation of $a^{2^i} \bmod N$.

Why gigabyte keys are reasonable: big enough to push latency beyond the 2^{64} limit, under reasonable assumptions.

Gigabyte inputs are millions of times larger than 2048-bit inputs.
These algorithms will take billions of times longer.
More cost to find *all* primes.

Open: What is minimum time for integer factorization?

the minimum time
's algorithm?

ttleneck: a^e mod N
it superposition e.

nal approach: series of
ed multiplications by
$/a$ mod N;
N and $1/a^2$ mod N;
N and $1/a^4$ mod N; etc.

ltiply these in parallel,
any more qubits;
l to parallelize initial
ation of a^{2^i} mod N.

Why gigabyte keys are reasonable:
big enough to push latency
beyond the 2^{64} limit,
under reasonable assumptions.

Gigabyte inputs are
millions of times larger
than 2048-bit inputs.
These algorithms will take
billions of times longer.
More cost to find *all* primes.

Open: What is minimum time
for integer factorization?

NIST's r
is define

num time
m?

a^e mod N
sition e.

ch: series of
cations by

a^2 mod N;
a^4 mod N; etc.

e in parallel,
qubits;
elize initial
^{i} mod N.

Why gigabyte keys are reasonable:
big enough to push latency
beyond the 2^{64} limit,
under reasonable assumptions.

Gigabyte inputs are
millions of times larger
than 2048-bit inputs.
These algorithms will take
billions of times longer.
More cost to find *all* primes.

Open: What is minimum time
for integer factorization?

NIST's middle sec
is defined by an A

of

etc.

el,

Why gigabyte keys are reasonable:
big enough to push latency
beyond the 2^{64} limit,
under reasonable assumptions.

Gigabyte inputs are
millions of times larger
than 2048-bit inputs.
These algorithms will take
billions of times longer.
More cost to find *all* primes.

Open: What is minimum time
for integer factorization?

NIST's middle security level
is defined by an AES-192 ke

Why gigabyte keys are reasonable: big enough to push latency beyond the 2^{64} limit, under reasonable assumptions.

Gigabyte inputs are millions of times larger than 2048-bit inputs. These algorithms will take billions of times longer. More cost to find *all* primes.

Open: What is minimum time for integer factorization?

NIST's middle security level is defined by an AES-192 key.

Why gigabyte keys are reasonable: big enough to push latency beyond the 2^{64} limit, under reasonable assumptions.

Gigabyte inputs are millions of times larger than 2048-bit inputs. These algorithms will take billions of times longer. More cost to find *all* primes.

Open: What is minimum time for integer factorization?

NIST's middle security level is defined by an AES-192 key.

With maximum depth 2^{64}, finding an AES-192 key requires $\approx 2^{144}$ cores.

Why gigabyte keys are reasonable:
big enough to push latency
beyond the 2^{64} limit,
under reasonable assumptions.

Gigabyte inputs are
millions of times larger
than 2048-bit inputs.
These algorithms will take
billions of times longer.
More cost to find *all* primes.

Open: What is minimum time
for integer factorization?

NIST's middle security level
is defined by an AES-192 key.

With maximum depth 2^{64},
finding an AES-192 key
requires $\approx 2^{144}$ cores.

This is nonsense! There is
not enough time to broadcast
the input to 2^{144} parallel
computations, and not enough
time to collect the results.

Why gigabyte keys are reasonable: big enough to push latency beyond the 2^{64} limit, under reasonable assumptions.

Gigabyte inputs are millions of times larger than 2048-bit inputs. These algorithms will take billions of times longer. More cost to find *all* primes.

Open: What is minimum time for integer factorization?

NIST's middle security level is defined by an AES-192 key.

With maximum depth 2^{64}, finding an AES-192 key requires $\approx 2^{144}$ cores.

This is nonsense! There is not enough time to broadcast the input to 2^{144} parallel computations, and not enough time to collect the results.

Is NIST implicitly assuming a higher latency limit?

gabyte keys are reasonable:
ugh to push latency
the 2^{64} limit,
easonable assumptions.

e inputs are
of times larger
48-bit inputs.
lgorithms will take
of times longer.
st to find *all* primes.

What is minimum time
ger factorization?

NIST's middle security level
is defined by an AES-192 key.

With maximum depth 2^{64},
finding an AES-192 key
requires $\approx 2^{144}$ cores.

This is nonsense! There is
not enough time to broadcast
the input to 2^{144} parallel
computations, and not enough
time to collect the results.

Is NIST implicitly assuming
a higher latency limit?

Some im

(2017 B

Conside
factoring
$(p_j - 1)p$

Unit gro
$\mathbf{Z}/2^{t_1} \times$

s are reasonable:
h latency
nit,
assumptions.

e
arger
uts.
will take
nger.
all primes.

nimum time
zation?

NIST's middle security level is defined by an AES-192 key.

With maximum depth 2^{64}, finding an AES-192 key requires $\approx 2^{144}$ cores.

This is nonsense! There is not enough time to broadcast the input to 2^{144} parallel computations, and not enough time to collect the results.

Is NIST implicitly assuming a higher latency limit?

Some improvemen

(2017 Bernstein–B

Consider Shor's al
factoring $N = p_1^{e_1}$
$(p_j - 1)p_j^{e_j-1}$ as 2

Unit group is isom
$\mathbf{Z}/2^{t_1} \times \cdots \times \mathbf{Z}/2$

onable:

ns.

.

me

NIST's middle security level is defined by an AES-192 key.

With maximum depth 2^{64}, finding an AES-192 key requires $\approx 2^{144}$ cores.

This is nonsense! There is not enough time to broadcast the input to 2^{144} parallel computations, and not enough time to collect the results.

Is NIST implicitly assuming a higher latency limit?

Some improvements to Shor

(2017 Bernstein–Biasse–Mos

Consider Shor's algorithm

factoring $N = p_1^{e_1} \cdots p_f^{e_f}$. W

$(p_j - 1)p_j^{e_j - 1}$ as $2^{t_j} u_j$ with u

Unit group is isomorphic to

$\mathbf{Z}/2^{t_1} \times \cdots \times \mathbf{Z}/2^{t_f} \times \mathbf{Z}/u_1$

NIST's middle security level is defined by an AES-192 key.

With maximum depth 2^{64}, finding an AES-192 key requires $\approx 2^{144}$ cores.

This is nonsense! There is not enough time to broadcast the input to 2^{144} parallel computations, and not enough time to collect the results.

Is NIST implicitly assuming a higher latency limit?

Some improvements to Shor

(2017 Bernstein–Biasse–Mosca)

Consider Shor's algorithm factoring $N = p_1^{e_1} \cdots p_f^{e_f}$. Write $(p_j - 1)p_j^{e_j - 1}$ as $2^{t_j} u_j$ with u_j odd.

Unit group is isomorphic to $\mathbf{Z}/2^{t_1} \times \cdots \times \mathbf{Z}/2^{t_f} \times \mathbf{Z}/u_1 \times \cdots$.

NIST's middle security level is defined by an AES-192 key.

With maximum depth 2^{64}, finding an AES-192 key requires $\approx 2^{144}$ cores.

This is nonsense! There is not enough time to broadcast the input to 2^{144} parallel computations, and not enough time to collect the results.

Is NIST implicitly assuming a higher latency limit?

Some improvements to Shor

(2017 Bernstein–Biasse–Mosca)

Consider Shor's algorithm factoring $N = p_1^{e_1} \cdots p_f^{e_f}$. Write $(p_j - 1)p_j^{e_j - 1}$ as $2^{t_j} u_j$ with u_j odd.

Unit group is isomorphic to $\mathbf{Z}/2^{t_1} \times \cdots \times \mathbf{Z}/2^{t_f} \times \mathbf{Z}/u_1 \times \cdots$.

Shor's algorithm (hopefully) computes order r of random unit. Order 2^{c_j} in $\mathbf{Z}/2^{t_j}$ is 2^{t_j} with probability $1/2$; $2^{t_j - 1}$ with probability $1/4$; etc.

middle security level
d by an AES-192 key.

aximum depth 2^{64},
an AES-192 key
$\approx 2^{144}$ cores.

nonsense! There is
ugh time to broadcast
t to 2^{144} parallel
ations, and not enough
collect the results.

implicitly assuming
latency limit?

Some improvements to Shor

(2017 Bernstein–Biasse–Mosca)

Consider Shor's algorithm factoring $N = p_1^{e_1} \cdots p_f^{e_f}$. Write $(p_j - 1)p_j^{e_j - 1}$ as $2^{t_j} u_j$ with u_j odd.

Unit group is isomorphic to $\mathbf{Z}/2^{t_1} \times \cdots \times \mathbf{Z}/2^{t_f} \times \mathbf{Z}/u_1 \times \cdots$.

Shor's algorithm (hopefully) computes order r of random unit.
Order 2^{c_j} in $\mathbf{Z}/2^{t_j}$ is
2^{t_j} with probability $1/2$;
$2^{t_j - 1}$ with probability $1/4$; etc.

Shor cor
Divisible
$c_j <$ ma
Factoriz
equal. C

urity level
ES-192 key.

epth 2^{64},
2 key
res.

There is
o broadcast
parallel
l not enough
e results.

assuming
mit?

Some improvements to Shor

(2017 Bernstein–Biasse–Mosca)

Consider Shor's algorithm factoring $N = p_1^{e_1} \cdots p_f^{e_f}$. Write $(p_j - 1)p_j^{e_j - 1}$ as $2^{t_j} u_j$ with u_j odd.

Unit group is isomorphic to $\mathbf{Z}/2^{t_1} \times \cdots \times \mathbf{Z}/2^{t_f} \times \mathbf{Z}/u_1 \times \cdots$.

Shor's algorithm (hopefully) computes order r of random unit. Order 2^{c_j} in $\mathbf{Z}/2^{t_j}$ is
2^{t_j} with probability $1/2$;
$2^{t_j - 1}$ with probability $1/4$; etc.

Shor computes gc
Divisible by p_j exa
$c_j < \max\{c_1, \ldots,$

Factorization fails
equal. Chance ≤ 1

y.

st

gh

Some improvements to Shor

(2017 Bernstein–Biasse–Mosca)

Consider Shor's algorithm factoring $N = p_1^{e_1} \cdots p_f^{e_f}$. Write $(p_j - 1)p_j^{e_j - 1}$ as $2^{t_j} u_j$ with u_j odd.

Unit group is isomorphic to $\mathbf{Z}/2^{t_1} \times \cdots \times \mathbf{Z}/2^{t_f} \times \mathbf{Z}/u_1 \times \cdots$.

Shor's algorithm (hopefully) computes order r of random unit. Order 2^{c_j} in $\mathbf{Z}/2^{t_j}$ is
2^{t_j} with probability $1/2$;
$2^{t_j - 1}$ with probability $1/4$; etc.

Shor computes gcd$\{N, a^{r/2}$

Divisible by p_j exactly when $c_j < \max\{c_1, \ldots, c_f\}$.

Factorization fails iff all c_j a
equal. Chance $\leq 1/2^{f-1}$.

Some improvements to Shor

(2017 Bernstein–Biasse–Mosca)

Consider Shor's algorithm factoring $N = p_1^{e_1} \cdots p_f^{e_f}$. Write $(p_j - 1)p_j^{e_j - 1}$ as $2^{t_j} u_j$ with u_j odd.

Unit group is isomorphic to $\mathbf{Z}/2^{t_1} \times \cdots \times \mathbf{Z}/2^{t_f} \times \mathbf{Z}/u_1 \times \cdots$.

Shor's algorithm (hopefully) computes order r of random unit. Order 2^{c_j} in $\mathbf{Z}/2^{t_j}$ is
2^{t_j} with probability $1/2$;
$2^{t_j - 1}$ with probability $1/4$; etc.

Shor computes $\gcd\{N, a^{r/2} - 1\}$. Divisible by p_j exactly when $c_j < \max\{c_1, \ldots, c_f\}$.

Factorization fails iff all c_j are equal. Chance $\le 1/2^{f-1}$.

Some improvements to Shor

(2017 Bernstein–Biasse–Mosca)

Consider Shor's algorithm factoring $N = p_1^{e_1} \cdots p_f^{e_f}$. Write $(p_j - 1)p_j^{e_j - 1}$ as $2^{t_j} u_j$ with u_j odd.

Unit group is isomorphic to $\mathbf{Z}/2^{t_1} \times \cdots \times \mathbf{Z}/2^{t_f} \times \mathbf{Z}/u_1 \times \cdots$.

Shor's algorithm (hopefully) computes order r of random unit. Order 2^{c_j} in $\mathbf{Z}/2^{t_j}$ is
2^{t_j} with probability $1/2$;
$2^{t_j - 1}$ with probability $1/4$; etc.

Shor computes $\gcd\{N, a^{r/2} - 1\}$. Divisible by p_j exactly when $c_j < \max\{c_1, \ldots, c_f\}$.

Factorization fails iff all c_j are equal. Chance $\le 1/2^{f-1}$.

More subtle problem: Factorization is likely to split off some of the primes with maximum t_j.

Can iterate Shor's algorithm enough times to completely factor. Many full-size iterations; many more for adversarial inputs.

nprovements to Shor

ernstein–Biasse–Mosca)

r Shor's algorithm
g $N = p_1^{e_1} \cdots p_f^{e_f}$. Write
$p_j^{e_j-1}$ as $2^{t_j} u_j$ with u_j odd.

up is isomorphic to
$\cdots \times \mathbf{Z}/2^{t_f} \times \mathbf{Z}/u_1 \times \cdots$.

lgorithm (hopefully)
es order r of random unit.
c_j in $\mathbf{Z}/2^{t_j}$ is
probability $1/2$;
th probability $1/4$; etc.

Shor computes $\gcd\{N, a^{r/2} - 1\}$.
Divisible by p_j exactly when
$c_j < \max\{c_1, \ldots, c_f\}$.

Factorization fails iff all c_j are
equal. Chance $\le 1/2^{f-1}$.

More subtle problem:
Factorization is likely to
split off some of the
primes with maximum t_j.

Can iterate Shor's algorithm
enough times to completely
factor. Many full-size iterations;
many more for adversarial inputs.

Better m
primality
with $a^{r/}$
$\ldots, a^d$

This spli
Any two
$\ge 1/2$ of

Factors
Much le

Also "pa
Run seve
giving se
Then fac

ts to Shor

Biasse–Mosca)

gorithm
$\cdots p_f^{e_f}$. Write
$^{t_j} u_j$ with u_j odd.

orphic to
$^{t_f} \times \mathbf{Z}/u_1 \times \cdots$.

hopefully)
of random unit.
is
y $1/2$;
ility $1/4$; etc.

Shor computes $\gcd\{N, a^{r/2} - 1\}$.
Divisible by p_j exactly when
$c_j < \max\{c_1, \ldots, c_f\}$.

Factorization fails iff all c_j are equal. Chance $\le 1/2^{f-1}$.

More subtle problem:
Factorization is likely to split off some of the primes with maximum t_j.

Can iterate Shor's algorithm enough times to completely factor. Many full-size iterations; many more for adversarial inputs.

Better method, ins
primality testing:
with $a^{r/2} + 1$, $a^{r/}$
$\ldots$, $a^d + 1$, $a^d -$

This splits p_j acco
Any two primes ha
$\ge 1/2$ of being spl

Factors are around
Much less overhea

Also "parallel cons
Run several times
giving several fact
Then factor into c

t

sca)

Vrite

u_j odd.

$\times \cdots$.

unit.

etc.

Shor computes $\gcd\{N, a^{r/2} - 1\}$.
Divisible by p_j exactly when
$c_j < \max\{c_1, \ldots, c_f\}$.

Factorization fails iff all c_j are
equal. Chance $\le 1/2^{f-1}$.

More subtle problem:
Factorization is likely to
split off some of the
primes with maximum t_j.

Can iterate Shor's algorithm
enough times to completely
factor. Many full-size iterations;
many more for adversarial inputs.

Better method, inspired by
primality testing: compute g
with $a^{r/2} + 1$, $a^{r/4} + 1$, $a^{r/8}$
$\ldots$, $a^d + 1$, $a^d - 1$, with od

This splits p_j according to c_j
Any two primes have chance
$\ge 1/2$ of being split.

Factors are around half size.
Much less overhead for recu

Also "parallel construction":
Run several times in parallel
giving several factorizations.
Then factor into coprimes.

Shor computes $\gcd\{N, a^{r/2} - 1\}$.
Divisible by p_j exactly when $c_j < \max\{c_1, \ldots, c_f\}$.

Factorization fails iff all c_j are equal. Chance $\le 1/2^{f-1}$.

More subtle problem: Factorization is likely to split off some of the primes with maximum t_j.

Can iterate Shor's algorithm enough times to completely factor. Many full-size iterations; many more for adversarial inputs.

Better method, inspired by primality testing: compute gcd with $a^{r/2} + 1$, $a^{r/4} + 1$, $a^{r/8} + 1$, $\ldots$, $a^d + 1$, $a^d - 1$, with odd d.

This splits p_j according to c_j.
Any two primes have chance $\ge 1/2$ of being split.

Factors are around half size.
Much less overhead for recursion.

Also "parallel construction":
Run several times in parallel, giving several factorizations.
Then factor into coprimes.

mputes gcd$\{N, a^{r/2} - 1\}$.
by p_j exactly when
x$\{c_1, \ldots, c_f\}$.

ation fails iff all c_j are
Chance $\le 1/2^{f-1}$.

btle problem:
ation is likely to
some of the
vith maximum t_j.

ate Shor's algorithm
times to completely
Many full-size iterations;
ore for adversarial inputs.

Better method, inspired by primality testing: compute gcd with $a^{r/2} + 1$, $a^{r/4} + 1$, $a^{r/8} + 1$, $\ldots$, $a^d + 1$, $a^d - 1$, with odd d.

This splits p_j according to c_j. Any two primes have chance $\ge 1/2$ of being split.

Factors are around half size. Much less overhead for recursion.

Also "parallel construction": Run several times in parallel, giving several factorizations. Then factor into coprimes.

These m
Didn't w

We actu
to search
numbers

Oracle fo
factor th
to recog

We twea
work in
with qub
fractions

d$\{N, a^{r/2} - 1\}$.
ctly when
$c_f\}$.

iff all c_j are
$/2^{f-1}$.

em:
ely to
he
num t_j.

algorithm
ompletely
size iterations;
versarial inputs.

Better method, inspired by primality testing: compute gcd with $a^{r/2}+1$, $a^{r/4}+1$, $a^{r/8}+1$, $\ldots$, a^d+1, a^d-1, with odd d.

This splits p_j according to c_j. Any two primes have chance $\ge 1/2$ of being split.

Factors are around half size. Much less overhead for recursion.

Also "parallel construction": Run several times in parallel, giving several factorizations. Then factor into coprimes.

These methods us
Didn't we claim b

We actually use G
to search for smoo
numbers in NFS.

Oracle for Grover'
factor thoroughly
to recognize smoo

We tweak (improv
work in superposit
with qubit budget
fractions, power d

$- 1\}$.

re

ons;

nputs.

Better method, inspired by primality testing: compute gcd with $a^{r/2} + 1$, $a^{r/4} + 1$, $a^{r/8} + 1$, $\ldots$, $a^d + 1$, $a^d - 1$, with odd d.

This splits p_j according to c_j. Any two primes have chance $\geq 1/2$ of being split.

Factors are around half size. Much less overhead for recursion.

Also "parallel construction": Run several times in parallel, giving several factorizations. Then factor into coprimes.

These methods use $>b$ qubi
Didn't we claim $b^{2/3+o(1)}$ q

We actually use Grover's me
to search for smooth $b^{2/3+o}$
numbers in NFS.

Oracle for Grover's method:
factor thoroughly enough
to recognize smooth inputs.

We tweak (improved) Shor t
work in superposition. Caref
with qubit budget for contin
fractions, power detection, e

Better method, inspired by primality testing: compute gcd with $a^{r/2}+1$, $a^{r/4}+1$, $a^{r/8}+1$, ..., a^d+1, a^d-1, with odd d.

This splits p_j according to c_j. Any two primes have chance $\ge 1/2$ of being split.

Factors are around half size. Much less overhead for recursion.

Also "parallel construction": Run several times in parallel, giving several factorizations. Then factor into coprimes.

These methods use $>b$ qubits. Didn't we claim $b^{2/3+o(1)}$ qubits?

We actually use Grover's method to search for smooth $b^{2/3+o(1)}$-bit numbers in NFS.

Oracle for Grover's method: factor thoroughly enough to recognize smooth inputs.

We tweak (improved) Shor to work in superposition. Careful with qubit budget for continued fractions, power detection, etc.

nethod, inspired by
/ testing: compute gcd
$^2+1$, $a^{r/4}+1$, $a^{r/8}+1$,
$+1$, a^d-1, with odd d.

its p_j according to c_j.
primes have chance
being split.

are around half size.
ss overhead for recursion.

arallel construction":
eral times in parallel,
everal factorizations.
ctor into coprimes.

These methods use $>b$ qubits.
Didn't we claim $b^{2/3+o(1)}$ qubits?

We actually use Grover's method to search for smooth $b^{2/3+o(1)}$-bit numbers in NFS.

Oracle for Grover's method: factor thoroughly enough to recognize smooth inputs.

We tweak (improved) Shor to work in superposition. Careful with qubit budget for continued fractions, power detection, etc.

A differe
randomn
Shor's a
$(\mathbf{Z}/N)^*$
for a ran

spired by
compute gcd
$^4 + 1$, $a^{r/8} + 1$,
1, with odd d.

rding to c_j.
ave chance
it.

half size.
d for recursion.

struction”:
in parallel,
orizations.
oprimes.

These methods use $>b$ qubits.
Didn’t we claim $b^{2/3+o(1)}$ qubits?

We actually use Grover’s method to search for smooth $b^{2/3+o(1)}$-bit numbers in NFS.

Oracle for Grover’s method: factor thoroughly enough to recognize smooth inputs.

We tweak (improved) Shor to work in superposition. Careful with qubit budget for continued fractions, power detection, etc.

A different way to
randomness of fac
Shor’s algorithm:
$(\mathbf{Z}/N)^*$ with $E(\mathbf{Z}/$
for a random ellipt

gcd

$^3 + 1$,

ld d.

j.

e

rsion.

,

These methods use $>b$ qubits.
Didn't we claim $b^{2/3+o(1)}$ qubits?

We actually use Grover's method to search for smooth $b^{2/3+o(1)}$-bit numbers in NFS.

Oracle for Grover's method: factor thoroughly enough to recognize smooth inputs.

We tweak (improved) Shor to work in superposition. Careful with qubit budget for continued fractions, power detection, etc.

A different way to improve randomness of factorizations
Shor's algorithm: replace gr
$(\mathbf{Z}/N)^*$ with $E(\mathbf{Z}/N)$
for a random elliptic curve E

These methods use $>b$ qubits.
Didn't we claim $b^{2/3+o(1)}$ qubits?

We actually use Grover's method to search for smooth $b^{2/3+o(1)}$-bit numbers in NFS.

Oracle for Grover's method: factor thoroughly enough to recognize smooth inputs.

We tweak (improved) Shor to work in superposition. Careful with qubit budget for continued fractions, power detection, etc.

A different way to improve randomness of factorizations in Shor's algorithm: replace group $(\mathbf{Z}/N)^*$ with $E(\mathbf{Z}/N)$ for a random elliptic curve E.

These methods use $>b$ qubits.
Didn't we claim $b^{2/3+o(1)}$ qubits?

We actually use Grover's method to search for smooth $b^{2/3+o(1)}$-bit numbers in NFS.

Oracle for Grover's method: factor thoroughly enough to recognize smooth inputs.

We tweak (improved) Shor to work in superposition. Careful with qubit budget for continued fractions, power detection, etc.

A different way to improve randomness of factorizations in Shor's algorithm: replace group $(\mathbf{Z}/N)^*$ with $E(\mathbf{Z}/N)$ for a random elliptic curve E.

Gal Dor suggests unifying Grover+ECM with Shor: e.g., compute esP on $E(\mathbf{Z}/N)$ where
e is superposition of scalars,
s is smooth scalar,
E is superposition of curves.

These methods use $>b$ qubits. Didn't we claim $b^{2/3+o(1)}$ qubits?

We actually use Grover's method to search for smooth $b^{2/3+o(1)}$-bit numbers in NFS.

Oracle for Grover's method: factor thoroughly enough to recognize smooth inputs.

We tweak (improved) Shor to work in superposition. Careful with qubit budget for continued fractions, power detection, etc.

A different way to improve randomness of factorizations in Shor's algorithm: replace group $(\mathbf{Z}/N)^*$ with $E(\mathbf{Z}/N)$ for a random elliptic curve E.

Gal Dor suggests unifying Grover+ECM with Shor: e.g., compute esP on $E(\mathbf{Z}/N)$ where e is superposition of scalars, s is smooth scalar, E is superposition of curves.

Open: What are minimum costs for this unification?