Post-quantum RSA

We built a great, great 1-terabyte RSA wall,
and we had the university pay for the electricity

Daniel J. Bernstein

Joint work with:
Nadia Heninger
Paul Lou
Luke Valenta

The referees are questioning applicability . . .

- Reviewer 1: “The cryptosystem is an interesting thought experiment, but I cannot believe it will be actually used”

The referees are questioning applicability . . .

- Reviewer 1: "The cryptosystem is an interesting thought experiment, but I cannot believe it will be actually used"
- Reviewer 2: "I can't see it ever being used. . . . the main result is almost comical"

The referees are questioning applicability . . .

- Reviewer 1: “The cryptosystem is an interesting thought experiment, but I cannot believe it will be actually used”
- Reviewer 2: “I can’t see it ever being used. . . . the main result is almost comical”
- Reviewer 3: “I’m not really convinced by the practicality of the scheme. . . . I can’t imagine exchanging 1 terabyte keys to secure communications”

The referees are questioning applicability . . .

- Reviewer 1: “The cryptosystem is an interesting thought experiment, but I cannot believe it will be actually used”
- Reviewer 2: “I can’t see it ever being used. . . . the main result is almost comical”
- Reviewer 3: “I’m not really convinced by the practicality of the scheme. . . . I can’t imagine exchanging 1 terabyte keys to secure communications”
- Reviewer 4: “a paranoid post-quantum solution may be sought at the great expense of performance”

The referees are questioning applicability . . .

- Reviewer 1: "The cryptosystem is an interesting thought experiment, but I cannot believe it will be actually used"
- Reviewer 2: "I can't see it ever being used. . . . the main result is almost comical"
- Reviewer 3: "I'm not really convinced by the practicality of the scheme. . . . I can't imagine exchanging 1 terabyte keys to secure communications"
- Reviewer 4: "a paranoid post-quantum solution may be sought at the great expense of performance"
- Reviewer 5: "not cheap"

. . . so how do we respond?

- Appeal to the past:

. . . so how do we respond?

- Appeal to the past: “Make RSA Great Again!”

. . . so how do we respond?

- Appeal to the past: "Make RSA Great Again!"
- Appeal to the future:

. . . so how do we respond?

- Appeal to the past: “Make RSA Great Again!”
- Appeal to the future: “Moore’s law says it’ll be okay!”

. . . so how do we respond?

- Appeal to the past: “Make RSA Great Again!”
- Appeal to the future: “Moore’s law says it’ll be okay!”
- Double down:

. . . so how do we respond?

- Appeal to the past: “Make RSA Great Again!”
- Appeal to the future: “Moore’s law says it’ll be okay!”
- Double down: “Digital cash with RSA blind signatures!”

. . . so how do we respond?

- Appeal to the past: “Make RSA Great Again!”
- Appeal to the future: “Moore’s law says it’ll be okay!”
- Double down: “Digital cash with RSA blind signatures!”
- FUD:

. . . so how do we respond?

- Appeal to the past: “Make RSA Great Again!”
- Appeal to the future: “Moore’s law says it’ll be okay!”
- Double down: “Digital cash with RSA blind signatures!”
- FUD: “Maybe all the alternatives will be broken!”

. . . so how do we respond?

- Appeal to the past: "Make RSA Great Again!"
- Appeal to the future: "Moore's law says it'll be okay!"
- Double down: "Digital cash with RSA blind signatures!"
- FUD: "Maybe all the alternatives will be broken!"
- Put it in perspective:

. . . so how do we respond?

- Appeal to the past: “Make RSA Great Again!”
- Appeal to the future: “Moore’s law says it’ll be okay!”
- Double down: “Digital cash with RSA blind signatures!”
- FUD: “Maybe all the alternatives will be broken!”
- Put it in perspective: “It’s better than QKD!”

. . . so how do we respond?

- Appeal to the past: “Make RSA Great Again!”
- Appeal to the future: “Moore’s law says it’ll be okay!”
- Double down: “Digital cash with RSA blind signatures!”
- FUD: “Maybe all the alternatives will be broken!”
- Put it in perspective: “It’s better than QKD!”
- The real answer:

. . . so how do we respond?

- Appeal to the past: "Make RSA Great Again!"
- Appeal to the future: "Moore's law says it'll be okay!"
- Double down: "Digital cash with RSA blind signatures!"
- FUD: "Maybe all the alternatives will be broken!"
- Put it in perspective: "It's better than QKD!"
- The real answer: "Someone is wrong on the Internet."

RSA scalability vs. Shor scalability

Conventional wisdom:

- Shor's algorithm has the same scalability as legitimate usage of RSA.
- "there's not going to be a larger key-size where a classical user of RSA gains [a] significant advantage over a quantum computing attacker"
- "If you increase the key size, it'd *still* be just as easy to break it as it is to encrypt"

What is the actual scalability of integer factorization?
What is the actual scalability of legitimate usage of RSA?

What is the fastest sorting algorithm?

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

What is the fastest sorting algorithm?

```
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`!

What is the fastest sorting algorithm?

```
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.

What is the fastest factoring algorithm?

Shor's algorithm?

- 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.

What is the fastest factoring algorithm?

Shor's algorithm?

- 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.

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

$\lfloor 10^{3009}\pi \rfloor$

314159265358979323846264338327950288419716939937510582097494459230781640628620899862803482534211706798214808651328230664
709384460955058223172535940812848111745028410270193852110555964462294895493038196442881097566593344612847564823378678316
527120190914564856692346034861045432664821339360726024914127372458700660631558817488152092096282925409171536436789259036
001133053054882046652138414695194151160943305727036575959195309218611738193261179310511854807446237996274956735188575272
489122793818301194912983367336244065664308602139494639522473719070217986094370277053921717629317675238467481846766940513
200056812714526356082778577134275778960917363717872146844090122495343014654958537105079227968925892354201995611212902196
086403441815981362977477130996051870721134999999837297804995105973173281609631859502445945534690830264252230825334468503
526193118817101000313783875288658753320838142061717766914730359825349042875546873115956286388235378759375195778185778053
217122680661300192787661119590921642019893809525720106548586327886593615338182796823030195203530185296899577362259941389
124972177528347913151557485724245415069595082953311686172785588907509838175463746493931925506040092770167113900984882401
285836160356370766010471018194295559619894676783744944825537977472684710404753464620804668425906949129331367702898915210
475216205696602405803815019351125338243003558764024749647326391419927260426992279678235478163600934172164121992458631503
028618297455570674983850549458858692699569092721079750930295532116534498720275596023648066549911988183479775356636980742
654252786255181841757467289097777279380008164706001614524919217321721477235014144197356854816136115735255213347574184946
843852332390739414333454776241686251898356948556209921922218427255025425688767179049460165346680498862723279178608578438
382796797668145410095388378636095068006422512520511739298489608412848862694560424196528502221066118630674427862203919494
504712371378696095636437191728746776465757396241389086583264599581339047802759009946576407895126946839835259570982582262
052248940772671947826848260147699090264013639443745530506820349625245174939965143142980919065925093722169646151570985838
741059788595977297549893016175392846813826868386894277415599185592524595395943104997252468084598727364469584865383673622
262609912460805124388439045124413654976278079771569143599770012961608944169486855584840635342207222582848864815845602850
601684273945226746767889525213852254995466672782398645659611635488623057745649803559363456817432411251507606947945109659
609402522887971089314566913686722874894056010150330861792868092087476091782493858900971490967598526136554978189312978482
168299894872265880485756401427047755513237964145152374623436454285844479526586782105114135473573952311342716610213596953
623144295248493718711014576540359027993440374200731057853906219838744780847848968332144571386875194350643021845319104848
100537061468067491927819119793995206141966342875444064374512371819217999839101591956181467514269123974894090718649423196
1567945208

When can we beat Shor's algorithm?

Costs for various algorithms (log factors suppressed):

- b^2 ops for Shor's algorithm.
  Assume best case: qubit ops as cheap as bit ops.

When can we beat Shor's algorithm?

Costs for various algorithms (log factors suppressed):

- b^2 ops for Shor's algorithm.
  Assume best case: qubit ops as cheap as bit ops.
- by ops for trial division to find primes $q \leq y$.
  Beats Shor's algorithm for y below b.

When can we beat Shor's algorithm?

Costs for various algorithms (log factors suppressed):

- b^2 ops for Shor's algorithm.
  Assume best case: qubit ops as cheap as bit ops.
- by ops for trial division to find primes $q \leq y$.
  Beats Shor's algorithm for y below b.
- $b + y$ ops to first compute $\prod_{q \leq y} q$, then gcd.
  Beats Shor's algorithm for y below b^2.

When can we beat Shor's algorithm?

Costs for various algorithms (log factors suppressed):

- b^2 ops for Shor's algorithm.
  Assume best case: qubit ops as cheap as bit ops.
- by ops for trial division to find primes $q \le y$.
  Beats Shor's algorithm for y below b.
- $b + y$ ops to first compute $\prod_{q \le y} q$, then gcd.
  Beats Shor's algorithm for y below b^2.
- $b\sqrt{y}$ ops for rho method to find primes $q \le y$.
  Not helpful here.

When does ECM beat Shor's algorithm?

$b \exp((\log y)^{1/2+o(1)})$ ops for ECM to find primes $q \le y$.

Beats Shor's algorithm for $\log y$ below $(\log b)^{2+o(1)}$.

When does ECM beat Shor's algorithm?

$b \exp((\log y)^{1/2+o(1)})$ ops for ECM to find primes $q \le y$.
Beats Shor's algorithm for $\log y$ below $(\log b)^{2+o(1)}$.

- Choose ECM parameter z with $z \in \exp((\alpha + o(1))\sqrt{\log y \log\log y})$.
- Each prime $q \le y$ is found by $1/C$ of all curves where $C \in \exp((1/2\alpha + o(1))\sqrt{\log y \log\log y})$.

When does ECM beat Shor's algorithm?

$b \exp((\log y)^{1/2+o(1)})$ ops for ECM to find primes $q \le y$.
Beats Shor's algorithm for $\log y$ below $(\log b)^{2+o(1)}$.

- Choose ECM parameter z with $z \in \exp((\alpha + o(1))\sqrt{\log y \log\log y})$.
- Each prime $q \le y$ is found by $1/C$ of all curves where $C \in \exp((1/2\alpha + o(1))\sqrt{\log y \log\log y})$.
- ECM searches through $C^{1+o(1)}$ curves. Choose $\alpha = 1/\sqrt{2}$. Cost: $\exp((\sqrt{2} + o(1))\sqrt{\log y \log\log y})$.

Daniel J. Bernstein, Nadia Heninger, Paul Lou, Luke Valenta

When does ECM beat Shor's algorithm?

$b \exp((\log y)^{1/2+o(1)})$ ops for ECM to find primes $q \le y$.
Beats Shor's algorithm for $\log y$ below $(\log b)^{2+o(1)}$.

- Choose ECM parameter z with $z \in \exp((\alpha + o(1))\sqrt{\log y \log\log y})$.
- Each prime $q \le y$ is found by $1/C$ of all curves where $C \in \exp((1/2\alpha + o(1))\sqrt{\log y \log\log y})$.
- ECM searches through $C^{1+o(1)}$ curves. Choose $\alpha = 1/\sqrt{2}$. Cost: $\exp((\sqrt{2} + o(1))\sqrt{\log y \log\log y})$.
- New: Use a Grover search through $C^{1+o(1)}$ curves. Choose $\alpha = 1/2$. Cost: $\exp((1 + o(1))\sqrt{\log y \log\log y})$.

Post-quantum RSA (PQRSA)

Make RSA fast again (see paper for asymptotics):

- Build public key N as product of many small primes.
- New: Batch generation of primes.
- Take exponent 3. (Could 2 be better? Not clear.)
- Use CRT for decryption.

Security $\geq 2^{100}$ qubit ops against all known attacks:

- Take $b = 2^{43}$ bits in N.
- Take 2^{12} bits in each prime.
- Use proper padding to stop chosen-ciphertext attacks.

Implementing PQRSA

OpenSSL doesn't support large key sizes

```
LUVALENT-M-L0UX:pqrsa lukevalenta$ openssl ca -cert myCert.crt -keyfile myKey.key -in 32k-request.pem -out signed.crt
Using configuration from /opt/local/etc/openssl/openssl.cnf
Check that the request matches the signature
Signature did not match the certificate request
140735170416720:error:04067069:rsa routines:RSA_EAY_PUBLIC_DECRYPT:modulus too large:rsa_eay.c:627:
140735170416720:error:0D0C5006:asn1 encoding routines:ASN1_item_verify:EVP lib:a_verify.c:218:
```

[0] http://fm4dd.com/openssl/certexamples.htm

Implementing PQRSA

OpenSSL doesn't support large key sizes

```
LUVALENT-M-L0UX:pqrsa lukevalenta$ openssl ca -cert myCert.crt -keyfile myKey.key -in 32k-request.pem -out signed.crt
Using configuration from /opt/local/etc/openssl/openssl.cnf
Check that the request matches the signature
Signature did not match the certificate request
140735170416720:error:04067069:rsa routines:RSA_EAY_PUBLIC_DECRYPT:modulus too large:rsa_eay.c:627:
140735170416720:error:0D0C5006:asn1 encoding routines:ASN1_item_verify:EVP lib:a_verify.c:218:
```

OpenSSL limits the RSA keysize per crypto/rsa/rsa.h:

```
# define OPENSSL_RSA_MAX_MODULUS_BITS   16384
```

per assumption that ultra-large keys make no sense in real world conditions.

[0] http://fm4dd.com/openssl/certexamples.htm

Implementing PQRSA

We implemented RSA key generation, encryption, decryption in C with modified version of GMP library:

- Change mpz struct internal typing from `int` to `int64_t`.
- Extend upper bound on memory allocation for `mpz_t`.
- Extend output and input functions to accomodate new mpz struct type.

Key generation

- 1TB multi-prime RSA key uses
  2 billion 4096-bit primes

Key generation

- 1TB multi-prime RSA key uses 2 billion 4096-bit primes
- Use batch algorithm to find primes

Key generation

- 1TB multi-prime RSA key uses 2 billion 4096-bit primes
- Use batch algorithm to find primes
- 1,975,000 core-hours
- Four months on 1,400-core cluster

Key generation (continued...)

- Construct multi-prime RSA modulus from generated primes
- Use product-tree algorithm

Key generation (continued...)

- Construct multi-prime RSA modulus from generated primes
- Use product-tree algorithm
- Four days multithreaded on single machine
- Max usage of 3.2TB RAM and 2.5TB swap
- First terabyte RSA key ever created! (At least in public.)

Encryption

- Use RSA-KEM
  - Generate random 1TB element with AES-256-CTR mode
    - Theoreticians might complain: “Hey, is this indifferentiable?”
    - Are there any fast alternatives with indifferentiability proofs?
    - Does indifferentiability matter?
  - Hash element to construct shared secret for key exchange
- To compute 3rd power: Use multiply-and-reduce algorithm instead of GMP’s modular exponentiation

Encryption

- Use RSA-KEM
  - Generate random 1TB element with AES-256-CTR mode
    - Theoreticians might complain: "Hey, is this indifferentiable?"
    - Are there any fast alternatives with indifferentiability proofs?
    - Does indifferentiability matter?
  - Hash element to construct shared secret for key exchange
- To compute 3rd power: Use multiply-and-reduce algorithm instead of GMP's modular exponentiation
- 256GB encryption in 100 hours on a single machine

Recent improvements

Reviewer 2: “in a world where most people are anxious to shave clock cycles, a cryptosystem with one-week encryption times isn’t going to fly”

Recent improvements

Reviewer 2: “in a world where most people are anxious to shave clock cycles, a cryptosystem with one-week encryption times isn’t going to fly”

Recent work from Josh Fried:

- Cluster-distributed parallelized FFT
- Completed 1TB encryption in about 4 hours with 896 cores
- Expect 1TB key generation (after primegen) to complete in about 5 hours with 896 cores

Ongoing work

Decryption:

- Precompute $d_i = e^{-1} \bmod (p_i - 1)$ and $m_i = C^{d_i} \bmod p_i$

Ongoing work

Decryption:

- Precompute $d_i = e^{-1} \bmod (p_i - 1)$ and $m_i = C^{d_i} \bmod p_i$
- Run CRT on pairs of $m_i, m_j, i \neq j$ and multiply p_i, p_j to create a node in the next level of the tree

Ongoing work

Decryption:

- Precompute $d_i = e^{-1} \bmod (p_i - 1)$ and $m_i = C^{d_i} \bmod p_i$
- Run CRT on pairs of $m_i, m_j, i \neq j$ and multiply p_i, p_j to create a node in the next level of the tree
- 16GB decryption in about 132 hours on single machine
- In progress: decryption with interpolator (save $\lg n$ factor over CRT-tree)

Ongoing work

Decryption:

- Precompute $d_i = e^{-1} \bmod (p_i - 1)$ and $m_i = C^{d_i} \bmod p_i$
- Run CRT on pairs of $m_i, m_j, i \neq j$ and multiply p_i, p_j to create a node in the next level of the tree
- 16GB decryption in about 132 hours on single machine
- In progress: decryption with interpolator (save $\lg n$ factor over CRT-tree)

Try to unify quantum-factoring landscape:

- Gal Dor suggests some bits of Shor, followed by Grover.