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

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

3141592653589793238462643383279502884197169399375105820974944592307816406286208998628034825342117067982148086513282306647
0938446095505822317253594081284811174502841027019385211055596446229489549303819644288109756659334461284756482337867831652
7120190914564856692346034861045432664821339360726024914127372458700660631558817488152092096282925409171536436789259036
0011330530548820466521384146951941511609433057270365759591953092186117381932611793105118548074462379962749567351885752724
8912279381830119491298336733624406566430860213949463952247371907021798609437027705392171762931767523846748184676694051320
0056812714526356082778577134275778960917363717872146844090122495343014654958537105079227968925892354201995611212902196
0864034418159813629774771309960518707211349999998372978049951059731732816096318595024459455346908302642522308253344685035
2619311881710100031378387528865875332083814206171776691473035982534904287554687311595628638823537875937519577818577805321
7122680661300192787661119590921642019893809525720106548586327886593615338182796823030195203530185296899577362259941389
1249721775283479131515574857242454150695950829533116861727855889075098381754637464939319255060400927701671139009848824012
8583616035637076601047101819429555961989467678374494482553797747268471040475346462080466842590694912933136770289891521047
5216205696602405803815019351125338243003558764024749647326391419927260426992279678235478163600934172164121992458631503
0286182974555706749838505494588586926995690927210797509302955321165344987202755960236480665499119881834797753566369807426
5425278625518184175746728909777727938000816470600161452491921732172147723501414419735685481613611573525521334757418494684
3852332390739414333454776241686251898356948556209921922218427255025425688767179049460165346680498862723279178608578438
3827967976681454100953883786360950680064225125205117392984896084128488626945604241965285022210661186306744278622039194945
0471237137869609563643719172874677646575739624138908658326459958133904780275900994657640789512694683983525957098258226205
2248940772671947826848260147699090264013639443745530506820349625245174939965143142980919065925093722169646151570985838741
0597885959772975498930161753928468138268683868942774155991855925245953959431049972524680845987273644695848653836736222626
0991246080512438843904512441365497627807977156914359977001296160894416948685558484063534220722258284886481584560285060168
4273945226746767889525213852254995466672782398645659611635488623057745649803559363456817432411251507606947945109659609402
5228879710893145669136867228748940560101503308617928680920874760917824938589009714909675985261365549781893129784821682998
9487226588048575640142704775551323796414515237462343645428584447952658678210511413547357395231134271661021359695362314429
5248493718711014576540359027993440374200731057853906219838744780847848968332144571386875194350643021845319104848100537061
4680674919278191197939952061419663428754440643745123718192179998391015919561814675142691239748940907186494231961567945208

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 $q \leq y$.
  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.
- $b\sqrt{y}$ ops for rho method to find primes $q \leq y$.
  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: $\exp((\sqrt{2} + o(1))\sqrt{\log y \log\log y})$.

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.