Compressing RSA/Rabin keys

D. J. Bernstein

Thanks to:

University of Illinois at Chicago

NSF CCR–9983950

Alfred P. Sloan Foundation

American Institute of Mathematics

Public keys

Each user publishes a key $U \in \{2^{2047}, 2^{2047}+1, \ldots, 2^{2048}-1\}$.

User knows prime factors of U.
Hopefully attacker doesn't.

RSA: also publish big exponent e; use primes allowing eth roots.
Rabin: always use exponent 2; use primes in $3 + 4\mathbf{Z}$.
Williams: $3 + 8\mathbf{Z}$ and $7 + 8\mathbf{Z}$.
Many subsequent variants; e.g., "RSA" using exponent 3, and "RSA" using exponent 65537.

/Rabin keys

is at Chicago

0

oundation

e of Mathematics

Public keys

Each user publishes a key $U \in \{2^{2047}, 2^{2047}+1, \ldots, 2^{2048}-1\}$.

User knows prime factors of U. Hopefully attacker doesn't.

RSA: also publish big exponent e; use primes allowing eth roots.
Rabin: always use exponent 2; use primes in $3 + 4\mathbf{Z}$.
Williams: $3 + 8\mathbf{Z}$ and $7 + 8\mathbf{Z}$.
Many subsequent variants; e.g., "RSA" using exponent 3, and "RSA" using exponent 65537.

The compression c

Can store U in 204
Can store $U_1, U_2, \ldots$
randomly accessibl

Can we use fewer

Knee-jerk answer:
If you can't afford
switch to 256-bit e
`http://cr.yp.to`

But elliptic-curve s
have slow verificat
Want a better ans

Public keys

Each user publishes a key $U \in \{2^{2047}, 2^{2047}+1, \ldots, 2^{2048}-1\}$.

User knows prime factors of U.
Hopefully attacker doesn't.

RSA: also publish big exponent e;
use primes allowing eth roots.
Rabin: always use exponent 2;
use primes in $3 + 4\mathbf{Z}$.
Williams: $3 + 8\mathbf{Z}$ and $7 + 8\mathbf{Z}$.
Many subsequent variants;
e.g., "RSA" using exponent 3,
and "RSA" using exponent 65537.

The compression question

Can store U in 2048 bits.
Can store $U_1, U_2, \ldots, U_n$,
randomly accessible, in $2048n$ bits.

Can we use fewer bits?

Knee-jerk answer: "No!
If you can't afford $2048n$ bits,
switch to 256-bit elliptic curves.
`http://cr.yp.to/ecdh.html`"

But elliptic-curve signatures
have slow verification.
Want a better answer.

es a key $U \in$
$\ldots, 2^{2048} - 1\}$.

factors of U.

doesn't.

big exponent e;
g eth roots.
exponent 2;
$4\mathbf{Z}$.
and $7 + 8\mathbf{Z}$.
variants;
exponent 3,
exponent 65537.

The compression question

Can store U in 2048 bits.
Can store $U_1, U_2, \ldots, U_n$,
randomly accessible, in $2048n$ bits.

Can we use fewer bits?

Knee-jerk answer: "No!
If you can't afford $2048n$ bits,
switch to 256-bit elliptic curves.
`http://cr.yp.to/ecdh.html`"

But elliptic-curve signatures
have slow verification.
Want a better answer.

Recognizing lower

$U \in \{2^{2047}, \ldots, 2$
so U has top bit 1
Don't store that b

With Rabin-Willia
Don't store botto

Better: Users neve
divisible by $3, 5, 7,$
so only 480 possib
for U mod 9240. F
bottom 13 bits wi
encoding of U mo

The compression question

Can store U in 2048 bits.
Can store $U_1, U_2, \ldots, U_n$, randomly accessible, in $2048n$ bits.

Can we use fewer bits?

Knee-jerk answer: "No! If you can't afford $2048n$ bits, switch to 256-bit elliptic curves. `http://cr.yp.to/ecdh.html`"

But elliptic-curve signatures have slow verification. Want a better answer.

Recognizing lower entropy

$U \in \{2^{2047}, \ldots, 2^{2048} - 1\}$ so U has top bit 1. Don't store that bit.

With Rabin-Williams: $U \in 5 + 8\mathbf{Z}$. Don't store bottom 3 bits.

Better: Users never generate U divisible by $3, 5, 7, 11$, so only 480 possibilities for U mod 9240. Replace bottom 13 bits with 9-bit encoding of U mod 9240.

question

48 bits.

. . . , U_n,

le, in 2048n bits.

bits?

“No!

2048n bits,

elliptic curves.

`o/ecdh.html`”

signatures

ion.

wer.

Recognizing lower entropy

$U \in \{2^{2047}, \dots, 2^{2048} - 1\}$
so U has top bit 1.
Don’t store that bit.

With Rabin-Williams: $U \in 5 + 8\mathbf{Z}$.
Don’t store bottom 3 bits.

Better: Users never generate U
divisible by $3, 5, 7, 11$,
so only 480 possibilities
for U mod 9240. Replace
bottom 13 bits with 9-bit
encoding of U mod 9240.

Have reduced 204

Can we do much b

Knee-jerk answer:

C’mon, you know

switch to elliptic c

e.g. User generate

independent unifo

$p \in \{2^{1023}, \dots, 2^{1}$

$q \in \{2^{1024}, \dots, 2^{1}$

$\approx 1/1025 \log 2$ cha

$\approx 1/1026 \log 2$ cha

$\approx 1/8$ chance of $\{$

$\approx 2 \log 2 - 1$ chan

so $> 2^{2023}$ equally

Recognizing lower entropy

$U \in \{2^{2047}, \ldots, 2^{2048} - 1\}$
so U has top bit 1.
Don't store that bit.

With Rabin-Williams: $U \in 5 + 8\mathbf{Z}$.
Don't store bottom 3 bits.

Better: Users never generate U
divisible by $3, 5, 7, 11$,
so only 480 possibilities
for U mod 9240. Replace
bottom 13 bits with 9-bit
encoding of U mod 9240.

Have reduced 2048 to 2043.
Can we do much better?

Knee-jerk answer: "No!
C'mon, you know you want to
switch to elliptic curves."

e.g. User generates $U = pq$ from
independent uniform random
$p \in \{2^{1023}, \ldots, 2^{1024} - 1\}$,
$q \in \{2^{1024}, \ldots, 2^{1025} - 1\}$:
$\approx 1/1025 \log 2$ chance of p prime,
$\approx 1/1026 \log 2$ chance of q prime,
$\approx 1/8$ chance of $\{3, 7\} + 8\mathbf{Z}$,
$\approx 2 \log 2 - 1$ chance of $pq < 2^{2048}$,
so $> 2^{2023}$ equally likely U's.

entropy

$^{2048}-1\}$

.

it.

ms: $U \in 5 + 8\mathbf{Z}$.

m 3 bits.

er generate U

11,

bilities

Replace

th 9-bit

d 9240.

Have reduced 2048 to 2043.
Can we do much better?

Knee-jerk answer: "No!
C'mon, you know you want to
switch to elliptic curves."

e.g. User generates $U = pq$ from
independent uniform random
$p \in \{2^{1023}, \ldots, 2^{1024}-1\}$,
$q \in \{2^{1024}, \ldots, 2^{1025}-1\}$:
$\approx 1/1025 \log 2$ chance of p prime,
$\approx 1/1026 \log 2$ chance of q prime,
$\approx 1/8$ chance of $\{3,7\} + 8\mathbf{Z}$,
$\approx 2\log 2 - 1$ chance of $pq < 2^{2048}$,
so $> 2^{2023}$ equally likely U's.

Reducing entropy

Define $f(U) = 50$
$g(U) = U$ with 50

Change key-genera
to produce keys U
Then can encode
saving one bit; als
top/bottom bits a

Brute-force key ge
generate U by the
if $f(U) = 1$, try ag
Conjecturally this
almost exactly 2 tr
confirmed by expe

Have reduced 2048 to 2043.

Can we do much better?

Knee-jerk answer: "No! C'mon, you know you want to switch to elliptic curves."

e.g. User generates $U = pq$ from independent uniform random $p \in \{2^{1023}, \ldots, 2^{1024} - 1\}$, $q \in \{2^{1024}, \ldots, 2^{1025} - 1\}$:
$\approx 1/1025 \log 2$ chance of p prime,
$\approx 1/1026 \log 2$ chance of q prime,
$\approx 1/8$ chance of $\{3, 7\} + 8\mathbf{Z}$,
$\approx 2 \log 2 - 1$ chance of $pq < 2^{2048}$,
so $> 2^{2023}$ equally likely U's.

<u>Reducing entropy</u>

Define $f(U) = 500$th bit of U, $g(U) = U$ with 500th bit omitted.

Change key-generation procedure to produce keys U with $f(U) = 0$. Then can encode U as $g(U)$, saving one bit; also save top/bottom bits as before.

Brute-force key generation: generate U by the old method; if $f(U) = 1$, try again. Conjecturally this takes almost exactly 2 tries on average; confirmed by experiment.

8 to 2043.

better?

"No!

you want to

urves."

es $U = pq$ from

rm random

$^{024} - 1\}$,

$^{025} - 1\}$:

ance of p prime,

ance of q prime,

$\{3, 7\} + 8\mathbf{Z}$,

ce of $pq < 2^{2048}$,

likely U's.

Reducing entropy

Define $f(U) = 500$th bit of U,
$g(U) = U$ with 500th bit omitted.

Change key-generation procedure to produce keys U with $f(U) = 0$.
Then can encode U as $g(U)$, saving one bit; also save top/bottom bits as before.

Brute-force key generation: generate U by the old method; if $f(U) = 1$, try again.
Conjecturally this takes almost exactly 2 tries on average; confirmed by experiment.

More generally, sel

f : {2048-bit strin

$\to \{k$-bit stri

g : {2048-bit string

$\to \{(2048 -$

with $f \times g$ invertib

Change key-genera

to produce keys U

Then can encode

saving k bits.

Is $f \times g$ easy to co

and easy to invert

for the functions w

Reducing entropy

Define $f(U) = 500$th bit of U,
$g(U) = U$ with 500th bit omitted.

Change key-generation procedure to produce keys U with $f(U) = 0$.
Then can encode U as $g(U)$, saving one bit; also save top/bottom bits as before.

Brute-force key generation: generate U by the old method; if $f(U) = 1$, try again.
Conjecturally this takes almost exactly 2 tries on average; confirmed by experiment.

More generally, select functions
$f : \{\text{2048-bit strings}\}$
$\quad \to \{k\text{-bit strings}\}$ and
$g : \{\text{2048-bit strings}\}$
$\quad \to \{(2048 - k)\text{-bit strings}\}$
with $f \times g$ invertible.

Change key-generation procedure to produce keys U with $f(U) = 0$.
Then can encode U as $g(U)$, saving k bits.

Is $f \times g$ easy to compute and easy to invert? Yes for the functions we'll consider.

0th bit of U,
0th bit omitted.

ation procedure
with $f(U) = 0$.
U as $g(U)$,
o save
s before.

neration:
old method;
gain.
takes
ries on average;
riment.

More generally, select functions
f : {2048-bit strings}
$\to$ {k-bit strings} and
g : {2048-bit strings}
$\to$ {$(2048 - k)$-bit strings}
with $f \times g$ invertible.

Change key-generation procedure
to produce keys U with $f(U) = 0$.
Then can encode U as $g(U)$,
saving k bits.

Is $f \times g$ easy to compute
and easy to invert? Yes
for the functions we'll consider.

Do U's exist with
Conjecturally chan
for the functions w
(Provable for f ch
from "universal" c

Brute force takes
far too slow for lar
Can we do much b
Yes. Will come ba

Are the resulting k
Not necessarily!

More generally, select functions
$f : \{2048\text{-bit strings}\}$
$\quad\to \{k\text{-bit strings}\}$ and
$g : \{2048\text{-bit strings}\}$
$\quad\to \{(2048 - k)\text{-bit strings}\}$
with $f \times g$ invertible.

Change key-generation procedure
to produce keys U with $f(U) = 0$.
Then can encode U as $g(U)$,
saving k bits.

Is $f \times g$ easy to compute
and easy to invert? Yes
for the functions we'll consider.

Do U's exist with $f(U) = 0$?
Conjecturally chance $\approx 1/2^k$
for the functions we'll consider.
(Provable for f chosen randomly
from "universal" classes.)

Brute force takes $\approx 2^k$ tries;
far too slow for large k.
Can we do much better?
Yes. Will come back to this.

Are the resulting keys secure?
Not necessarily!

lect functions
gs}
ngs} and
gs}
k)-bit strings}
ble.

ation procedure
with $f(U) = 0$.
U as $g(U)$,

ompute
? Yes
we'll consider.

Do U's exist with $f(U) = 0$?
Conjecturally chance $\approx 1/2^k$
for the functions we'll consider.
(Provable for f chosen randomly
from "universal" classes.)

Brute force takes $\approx 2^k$ tries;
far too slow for large k.
Can we do much better?
Yes. Will come back to this.

Are the resulting keys secure?
Not necessarily!

The half-special nu

1998 Lenstra: "Nu
form $2^{1024} \pm t \ldots$
1024-bit RSA secu
t is not much sma
Chance of an "unu
NFS polynomial is

Not true. Reducin
using $f(U) =$ half
reduces conjecture

Skewed NFS polyn
(1999 Murphy) tu
unusually small fo

Do U's exist with $f(U) = 0$?
Conjecturally chance $\approx 1/2^k$
for the functions we'll consider.
(Provable for f chosen randomly
from "universal" classes.)

Brute force takes $\approx 2^k$ tries;
far too slow for large k.
Can we do much better?
Yes. Will come back to this.

Are the resulting keys secure?
Not necessarily!

The half-special number-field sieve

1998 Lenstra: "Numbers of the
form $2^{1024} \pm t$ ... offer regular
1024-bit RSA security, as long as
t is not much smaller than 2^{500}."
Chance of an "unusually small"
NFS polynomial is "negligible."

Not true. Reducing entropy,
using $f(U) =$ half the bits of U,
reduces conjectured security level.

Skewed NFS polynomials
(1999 Murphy) turn out to be
unusually small for these numbers.

$f(U) = 0$?
nce $\approx 1/2^k$
we'll consider.
osen randomly
classes.)

$\approx 2^k$ tries;
rge k.
better?
ck to this.

keys secure?

The half-special number-field sieve

1998 Lenstra: "Numbers of the form $2^{1024} \pm t \ldots$ offer regular 1024-bit RSA security, as long as t is not much smaller than 2^{500}." Chance of an "unusually small" NFS polynomial is "negligible."

Not true. Reducing entropy, using $f(U) =$ half the bits of U, reduces conjectured security level.

Skewed NFS polynomials (1999 Murphy) turn out to be unusually small for these numbers.

Sharing entropy

Generate random
from set S of all p
Define $S_1 = S \cap f$

Generate random
e.g., for $f = 500\mathrm{th}$
generate random U
same 500th bit as
Similarly generate

Compress U_2 to $g($
compress U_3 to $g($
Overall $(2048 - k)$
to store $U_1, U_2, \ldots$

The half-special number-field sieve

1998 Lenstra: "Numbers of the form $2^{1024} \pm t$ . . . offer regular 1024-bit RSA security, as long as t is not much smaller than 2^{500}."
Chance of an "unusually small" NFS polynomial is "negligible."

Not true. Reducing entropy, using $f(U) =$ half the bits of U, reduces conjectured security level.

Skewed NFS polynomials (1999 Murphy) turn out to be unusually small for these numbers.

Sharing entropy

Generate random U_1 from set S of all possible keys.
Define $S_1 = S \cap f^{-1}(f(U_1))$.

Generate random $U_2 \in S_1$:
e.g., for $f =$ 500th bit, generate random U_2 having same 500th bit as U_1.
Similarly generate $U_3, U_4, \ldots$.

Compress U_2 to $g(U_2)$; compress U_3 to $g(U_3)$; etc.
Overall $(2048 - k)n + k$ bits to store $U_1, U_2, \ldots, U_n$.

umber-field sieve

umbers of the
offer regular
urity, as long as
ller than 2^{500}."
usually small"
"negligible."

g entropy,
the bits of U,
ed security level.

nomials
rn out to be
r these numbers.

Sharing entropy

Generate random U_1
from set S of all possible keys.
Define $S_1 = S \cap f^{-1}(f(U_1))$.

Generate random $U_2 \in S_1$:
e.g., for $f =$ 500th bit,
generate random U_2 having
same 500th bit as U_1.
Similarly generate $U_3, U_4, \ldots$.

Compress U_2 to $g(U_2)$;
compress U_3 to $g(U_3)$; etc.
Overall $(2048 - k)n + k$ bits
to store $U_1, U_2, \ldots, U_n$.

If distribution of U
is uniform over S,
and distribution of
is uniform over S_1
then distribution o
is uniform over S.

So attacker's chan
is *provably* identic
attacker's chance
Same comment wi
replaced by "forgi
etc.

Sharing entropy is

Sharing entropy

Generate random U_1
from set S of all possible keys.
Define $S_1 = S \cap f^{-1}(f(U_1))$.

Generate random $U_2 \in S_1$:
e.g., for $f = 500$th bit,
generate random U_2 having
same 500th bit as U_1.
Similarly generate $U_3, U_4, \ldots$.

Compress U_2 to $g(U_2)$;
compress U_3 to $g(U_3)$; etc.
Overall $(2048 - k)n + k$ bits
to store $U_1, U_2, \ldots, U_n$.

If distribution of U_1
is uniform over S,
and distribution of U_2 given U_1
is uniform over S_1,
then distribution of U_2
is uniform over S.

So attacker's chance of factoring U_2
is *provably* identical to
attacker's chance of factoring U_1.
Same comment with "factoring"
replaced by "forging signatures"
etc.

Sharing entropy is provably secure.

U_1

possible keys.

$^{-1}(f(U_1))$.

$U_2 \in S_1$:

n bit,

U_2 having

U_1.

$U_3, U_4, \ldots$.

(U_2);

$U_3)$; etc.

$)n + k$ bits

$., U_n$.

If distribution of U_1
is uniform over S,
and distribution of U_2 given U_1
is uniform over S_1,
then distribution of U_2
is uniform over S.

So attacker's chance of factoring U_2
is *provably* identical to
attacker's chance of factoring U_1.
Same comment with "factoring"
replaced by "forging signatures"
etc.

Sharing entropy is provably secure.

Time to factor U_1
can be less than d
the time for a sing
(e.g., Schnorr, Era

Analogy: brute-for
versus a secret-key
finds n target keys
as finding one targ

Problem arises wit
shared entropy.
(e.g., Coppersmith

For safety, choose
so that (conjectura
can't even do *one*

If distribution of U_1
is uniform over S,
and distribution of U_2 given U_1
is uniform over S_1,
then distribution of U_2
is uniform over S.

So attacker's chance of factoring U_2
is *provably* identical to
attacker's chance of factoring U_1.
Same comment with "factoring"
replaced by "forging signatures"
etc.

Sharing entropy is provably secure.

Time to factor U_1 *and* U_2
can be less than double
the time for a single factorization.
(e.g., Schnorr, Eratosthenes)

Analogy: brute-force search
versus a secret-key cipher
finds n target keys in same time
as finding one target key.

Problem arises with or without
shared entropy.
(e.g., Coppersmith, Bernstein)

For safety, choose key sizes
so that (conjecturally) attacker
can't even do *one* factorization.

U_1

f U_2 given U_1

,

of U_2

nce of factoring U_2

al to

of factoring U_1.

ith "factoring"

ng signatures"

provably secure.

Time to factor U_1 *and* U_2
can be less than double
the time for a single factorization.
(e.g., Schnorr, Eratosthenes)

Analogy: brute-force search
versus a secret-key cipher
finds n target keys in same time
as finding one target key.

Problem arises with or without
shared entropy.
(e.g., Coppersmith, Bernstein)

For safety, choose key sizes
so that (conjecturally) attacker
can't even do *one* factorization.

Perhaps time to fa
is below time to fa

Analogy: brute-for
finds *some* target
after $\approx 1/n$ of the

As before, problem
with or without sh

For safety, multiply
factorization succe
(e.g. ECM success
by n before choosi

Time to factor U_1 *and* U_2
can be less than double
the time for a single factorization.
(e.g., Schnorr, Eratosthenes)

Analogy: brute-force search
versus a secret-key cipher
finds n target keys in same time
as finding one target key.

Problem arises with or without
shared entropy.
(e.g., Coppersmith, Bernstein)

For safety, choose key sizes
so that (conjecturally) attacker
can't even do *one* factorization.

Perhaps time to factor U_1 *or* U_2
is below time to factor U_1.

Analogy: brute-force search
finds *some* target key out of n
after $\approx 1/n$ of the computation.

As before, problem arises
with or without shared entropy.

For safety, multiply conjectured
factorization success chance
(e.g. ECM success chance)
by n before choosing key sizes.

and U_2
ouble
gle factorization.
atosthenes)

rce search
y cipher
s in same time
get key.

th or without

n, Bernstein)

key sizes
ally) attacker
factorization.

Perhaps time to factor U_1 *or* U_2 is below time to factor U_1.

Analogy: brute-force search finds *some* target key out of n after $\approx 1/n$ of the computation.

As before, problem arises with or without shared entropy.

For safety, multiply conjectured factorization success chance (e.g. ECM success chance) by n before choosing key sizes.

Is this overkill?

Are there algorithm
U_1 or U_2 or ... or
more quickly than

For discrete logs, p
by randomized self

For factorization, n
without an extra n
Factorization litera
explicitly address n
Maybe we're overs
by considering just

Perhaps time to factor U_1 *or* U_2 is below time to factor U_1.

Analogy: brute-force search finds *some* target key out of n after $\approx 1/n$ of the computation.

As before, problem arises with or without shared entropy.

For safety, multiply conjectured factorization success chance (e.g. ECM success chance) by n before choosing key sizes.

Is this overkill?

Are there algorithms to factor U_1 or U_2 or . . . or U_n more quickly than factoring U_1?

For discrete logs, prove "no" by randomized self-reduction.

For factorization, no hope of proof without an extra n.
Factorization literature needs to explicitly address multiple inputs.
Maybe we're oversimplifying by considering just one input.

actor U_1 *or* U_2
actor U_1.

rce search
key out of n
e computation.

n arises
ared entropy.

y conjectured
ess chance
s chance)
ing key sizes.

Is this overkill?

Are there algorithms to factor
U_1 or U_2 or ... or U_n
more quickly than factoring U_1?

For discrete logs, prove "no"
by randomized self-reduction.

For factorization, no hope of proof
without an extra n.
Factorization literature needs to
explicitly address multiple inputs.
Maybe we're oversimplifying
by considering just one input.

Generating U give

Define $f(U) = U$ r

Reasonably fast ge
with $f(pq) = f(U$
Choose 1024-bit p
$q = 2^{1024} + (p^{-1} f$
If not both primes
If $pq > 2^{2048}$, try

Conjecturally $\approx 2^1$
on average.

Is this overkill?

Are there algorithms to factor U_1 or U_2 or ... or U_n more quickly than factoring U_1?

For discrete logs, prove “no” by randomized self-reduction.

For factorization, no hope of proof without an extra n.
Factorization literature needs to explicitly address multiple inputs.
Maybe we’re oversimplifying by considering just one input.

Generating U given bottom half

Define $f(U) = U \bmod 2^{1024}$.

Reasonably fast generation of p, q with $f(pq) = f(U_1)$, given $f(U_1)$:
Choose 1024-bit p. Compute $q = 2^{1024} + (p^{-1} f(U_1) \bmod 2^{1024})$.
If not both primes, try again.
If $pq > 2^{2048}$, try again.

Conjecturally $\approx 2^{17}$ tries on average.

ms to factor
U_n
factoring U_1?

prove "no"
f-reduction.

no hope of proof
n.
ature needs to
multiple inputs.
simplifying
t one input.

Generating U given bottom half

Define $f(U) = U \bmod 2^{1024}$.

Reasonably fast generation of p, q
with $f(pq) = f(U_1)$, given $f(U_1)$:
Choose 1024-bit p. Compute
$q = 2^{1024} + (p^{-1} f(U_1) \bmod 2^{1024})$.
If not both primes, try again.
If $pq > 2^{2048}$, try again.

Conjecturally $\approx 2^{17}$ tries
on average.

Analogous metho
$f(U) = \lfloor U/2^{1024} \rfloor$

Method reinvente

Published 1991 Gu
in context of redu
"Some forms of th
. . . need less stora
all of the bits of th
most significant by
valued to zero."

Generating U given bottom half

Define $f(U) = U \bmod 2^{1024}$.

Reasonably fast generation of p, q with $f(pq) = f(U_1)$, given $f(U_1)$:
Choose 1024-bit p. Compute
$q = 2^{1024} + (p^{-1} f(U_1) \bmod 2^{1024})$.
If not both primes, try again.
If $pq > 2^{2048}$, try again.

Conjecturally $\approx 2^{17}$ tries on average.

Analogous method works for $f(U) = \lfloor U/2^{1024} \rfloor$.

Method reinvented several times.

Published 1991 Guillou Quisquater, in context of reducing entropy:
"Some forms of the modulus
... need less storage. ...
all of the bits of the y
most significant bytes are
valued to zero."

n bottom half

mod 2^{1024}.

eneration of p, q

$_1$), given $f(U_1)$:

. Compute

$f(U_1) \bmod 2^{1024}$).

, try again.

again.

^{7} tries

Analogous method works for
$f(U) = \lfloor U/2^{1024} \rfloor$.

Method reinvented several times.

Published 1991 Guillou Quisquater, in context of reducing entropy: "Some forms of the modulus ... need less storage. ... all of the bits of the y most significant bytes are valued to zero."

Patent application

by Vanstone and Z

"A method of enc

selecting said publ

having a plurality

at least one set be

predetermined pat

and applying said

to encrypt the me

Includes some gen

ranging from sensi

Granted 2000: US

Analogous method works for
$f(U) = \lfloor U/2^{1024} \rfloor$.

Method reinvented several times.

Published 1991 Guillou Quisquater, in context of reducing entropy: "Some forms of the modulus . . . need less storage. . . . all of the bits of the y most significant bytes are valued to zero."

Patent application filed 1995 by Vanstone and Zuccherato: "A method of encrypting data. . . selecting said public key. . . having a plurality of sets of bits, at least one set being of a predetermined pattern of bits. . . and applying said public key to encrypt the message."

Includes some generation methods, ranging from sensible to silly.

Granted 2000: US 6134325.

d works for
.

d several times.

uillou Quisquater,
cing entropy:
ne modulus
nge. . . .
ne y
ytes are

Patent application filed 1995 by Vanstone and Zuccherato: "A method of encrypting data. . . selecting said public key. . . having a plurality of sets of bits, at least one set being of a predetermined pattern of bits. . . and applying said public key to encrypt the message."

Includes some generation methods, ranging from sensible to silly.

Granted 2000: US 6134325.

More patents filed
responding to silly
"Select a number
factor q as n'/p; c
the factor q is prim
q is prime, comput
n as the product o
determine that the
RSA modulus; and
is not prime, adjus
the check of wheth
prime."

Granted 2002: US
US 6496929.

Patent application filed 1995
by Vanstone and Zuccherato:
"A method of encrypting data...
selecting said public key...
having a plurality of sets of bits,
at least one set being of a
predetermined pattern of bits...
and applying said public key
to encrypt the message."

Includes some generation methods, ranging from sensible to silly.

Granted 2000: US 6134325.

More patents filed by Lenstra, responding to silly methods.
"Select a number p; ... obtain the factor q as n'/p; check whether the factor q is prime; if the factor q is prime, compute the number n as the product of p and q and determine that the number n is the RSA modulus; and if the factor q is not prime, adjust q and repeat the check of whether the factor q is prime."

Granted 2002: US 6404890,
US 6496929.

filed 1995
Zuccherato:
rypting data. . .
ic key. . .
of sets of bits,
eing of a
tern of bits. . .
public key
ssage."

eration methods,
ble to silly.

6134325.

More patents filed by Lenstra, responding to silly methods. "Select a number p; . . . obtain the factor q as n'/p; check whether the factor q is prime; if the factor q is prime, compute the number n as the product of p and q and determine that the number n is the RSA modulus; and if the factor q is not prime, adjust q and repeat the check of whether the factor q is prime."

Granted 2002: US 6404890, US 6496929.

These key-generat
allow compression
2048 bits to 1024

Exactly how fast i
Can we make it ev

What if $f(U) = U$
What if $f(U) = U$
Do we still have fa
key-generation me

More patents filed by Lenstra, responding to silly methods.

"Select a number p; ... obtain the factor q as n'/p; check whether the factor q is prime; if the factor q is prime, compute the number n as the product of p and q and determine that the number n is the RSA modulus; and if the factor q is not prime, adjust q and repeat the check of whether the factor q is prime."

Granted 2002: US 6404890, US 6496929.

These key-generation methods allow compression from 2048 bits to 1024 bits.

Exactly how fast is this?
Can we make it even faster?

What if $f(U) = U \bmod 2^{1280}$?
What if $f(U) = U \bmod 2^{1536}$?
Do we still have fast key-generation methods?

by Lenstra,
methods.
p; . . . obtain the
check whether
me; if the factor
te the number
of p and q and
e number n is the
d if the factor q
st q and repeat
her the factor q is

6404890,

These key-generation methods allow compression from 2048 bits to 1024 bits.

Exactly how fast is this?
Can we make it even faster?

What if $f(U) = U \bmod 2^{1280}$?
What if $f(U) = U \bmod 2^{1536}$?
Do we still have fast key-generation methods?

Unbalanced primes

Take $f(U) = U$ m

Choose 768-bit p.
$q = 2^{1280} + (p^{-1} f$
If not both primes
If $pq > 2^{2048}$, try

This allows compr
2048 bits to 768 b
with unbalanced p
(1998 Lenstra)

ECM more danger
Don't want p so s

These key-generation methods allow compression from 2048 bits to 1024 bits.

Exactly how fast is this? Can we make it even faster?

What if $f(U) = U \bmod 2^{1280}$?
What if $f(U) = U \bmod 2^{1536}$?
Do we still have fast key-generation methods?

Unbalanced primes

Take $f(U) = U \bmod 2^{1280}$.

Choose 768-bit p. Compute
$q = 2^{1280} + (p^{-1} f(U_1) \bmod 2^{1280})$.
If not both primes, try again.
If $pq > 2^{2048}$, try again.

This allows compression from 2048 bits to 768 bits, with unbalanced p, q.
(1998 Lenstra)

ECM more dangerous than NFS!
Don't want p so small.

ion methods

from

bits.

s this?

ven faster?

mod 2^{1280}?

mod 2^{1536}?

ast

thods?

Unbalanced primes

Take $f(U) = U \bmod 2^{1280}$.

Choose 768-bit p. Compute
$q = 2^{1280} + (p^{-1} f(U_1) \bmod 2^{1280})$.
If not both primes, try again.
If $pq > 2^{2048}$, try again.

This allows compression from
2048 bits to 768 bits,
with unbalanced p, q.
(1998 Lenstra)

ECM more dangerous than NFS!
Don't want p so small.

Primes in lattices

Take $f(U) = U$ m

Choose 683-bit p_0
$q_0 = p_0^{-1} f(U_1)$ mo
Idea: will take $p =$
and $q = q_0 + 2^{683}$

Use lattice reducti
to try to find p_1, q
with $(f(U_1) - p_0 q$
$p_1 q_0 + q_1 p_0$ (mo
Good chance of su

(2003 Coppersmit

Unbalanced primes

Take $f(U) = U \bmod 2^{1280}$.

Choose 768-bit p. Compute
$q = 2^{1280} + (p^{-1} f(U_1) \bmod 2^{1280})$.
If not both primes, try again.
If $pq > 2^{2048}$, try again.

This allows compression from
2048 bits to 768 bits,
with unbalanced p, q.
(1998 Lenstra)

ECM more dangerous than NFS!
Don't want p so small.

Primes in lattices

Take $f(U) = U \bmod 2^{1366}$.

Choose 683-bit p_0. Compute
$q_0 = p_0^{-1} f(U_1) \bmod 2^{683}$.
Idea: will take $p = p_0 + 2^{683} p_1$
and $q = q_0 + 2^{683} q_1$.

Use lattice reduction
to try to find $p_1, q_1 \approx 2^{341}$
with $(f(U_1) - p_0 q_0)/2^{683} \equiv$
$p_1 q_0 + q_1 p_0 \pmod{2^{683}}$.
Good chance of success.

(2003 Coppersmith)

s

od 2^{1280}.

Compute
$f(U_1) \bmod 2^{1280}$).
, try again.
again.

ession from
its,
$, q$.

ous than NFS!
mall.

Primes in lattices

Take $f(U) = U \bmod 2^{1366}$.

Choose 683-bit p_0. Compute
$q_0 = p_0^{-1} f(U_1) \bmod 2^{683}$.
Idea: will take $p = p_0 + 2^{683} p_1$
and $q = q_0 + 2^{683} q_1$.

Use lattice reduction
to try to find $p_1, q_1 \approx 2^{341}$
with $(f(U_1) - p_0 q_0)/2^{683} \equiv$
$p_1 q_0 + q_1 p_0 \pmod{2^{683}}$.
Good chance of success.

(2003 Coppersmith)

This allows compr
2048 bits to 682 b
with balanced p, q

Minor flaw: unifor
does not produce
uniform random in
But confirm exper
that each p_0 has g
of producing at lea
This implies that e
of p has probabilit
not far above unifo

Primes in lattices

Take $f(U) = U \bmod 2^{1366}$.

Choose 683-bit p_0. Compute $q_0 = p_0^{-1} f(U_1) \bmod 2^{683}$. Idea: will take $p = p_0 + 2^{683} p_1$ and $q = q_0 + 2^{683} q_1$.

Use lattice reduction to try to find $p_1, q_1 \approx 2^{341}$ with $(f(U_1) - p_0 q_0)/2^{683} \equiv p_1 q_0 + q_1 p_0 \pmod{2^{683}}$. Good chance of success.

(2003 Coppersmith)

This allows compression from 2048 bits to 682 bits, with balanced p, q.

Minor flaw: uniform random p_0 does not produce exactly uniform random integer p. But confirm experimentally that each p_0 has good chance of producing at least one p. This implies that each choice of p has probability not far above uniform.

od 2^{1366}.

. Compute
od 2^{683}.
$= p_0 + 2^{683} p_1$
q_1.

on
$_1 \approx 2^{341}$
$_0)/2^{683} \equiv$
d 2^{683}).
uccess.

h)

This allows compression from 2048 bits to 682 bits, with balanced p, q.

Minor flaw: uniform random p_0 does not produce exactly uniform random integer p. But confirm experimentally that each p_0 has good chance of producing at least one p. This implies that each choice of p has probability not far above uniform.

Some open questi

Find random p, q ≈
given pq mod 2^{150}
use higher-dimensi

Or $p \approx 2^{768}$, $q \approx$
Doesn't seem to i
lattice effectivenes

Find *three* balance
given half the bits

Do better with an

This allows compression from 2048 bits to 682 bits, with balanced p, q.

Minor flaw: uniform random p_0 does not produce exactly uniform random integer p. But confirm experimentally that each p_0 has good chance of producing at least one p. This implies that each choice of p has probability not far above uniform.

Some open questions:

Find random $p, q \approx 2^{1024}$ given $pq \bmod 2^{1500}$? Maybe use higher-dimensional lattices.

Or $p \approx 2^{768}$, $q \approx 2^{1280}$? Doesn't seem to improve lattice effectiveness.

Find *three* balanced integers given half the bits of product?

Do better with another f shape?

ession from
bits,
.

m random p_0
exactly
nteger p.
imentally
good chance
ast one p.
each choice
y
orm.

Some open questions:

Find random $p, q \approx 2^{1024}$ given $pq \bmod 2^{1500}$? Maybe use higher-dimensional lattices.

Or $p \approx 2^{768}$, $q \approx 2^{1280}$? Doesn't seem to improve lattice effectiveness.

Find *three* balanced integers given half the bits of product?

Do better with another f shape?

Key-generation sp

Start with many p
Use trial division e
Then try 2^{p-1} mo
$\approx 2^6$ exponentiatio
to find one prime.

Traditional key ge
chooses p, q indep
$\approx 2^7$ exponentiatio
Faster, slightly no
build visible prime

If p determines q:
$\approx 2^{12}$ exponentiat

Some open questions:

Find random $p, q \approx 2^{1024}$ given $pq \bmod 2^{1500}$? Maybe use higher-dimensional lattices.

Or $p \approx 2^{768}$, $q \approx 2^{1280}$? Doesn't seem to improve lattice effectiveness.

Find *three* balanced integers given half the bits of product?

Do better with another f shape?

Key-generation speed

Start with many p's. Use trial division etc. Then try $2^{p-1} \bmod p$. $\approx 2^6$ exponentiations to find one prime.

Traditional key generation chooses p, q independently. $\approx 2^7$ exponentiations. Faster, slightly non-uniform: build visible primes (Maurer).

If p determines q: $\approx 2^{12}$ exponentiations.

ons:

$\approx 2^{1024}$

^{0}? Maybe

ional lattices.

2^{1280}?

mprove

s.

ed integers

of product?

other f shape?

Key-generation speed

Start with many p's.
Use trial division etc.
Then try $2^{p-1} \bmod p$.
$\approx 2^6$ exponentiations
to find one prime.

Traditional key generation
chooses p, q independently.
$\approx 2^7$ exponentiations.
Faster, slightly non-uniform:
build visible primes (Maurer).

If p determines q:
$\approx 2^{12}$ exponentiations.

For $f(U) = U$ mod

Each p determines
pool of 2^{16} possib
Select randomly fr
until finding a prin
$\approx 2^7$ exponentiatio

For $f(U) = U$ mod
Obtain pool of pai
with all different p
and all different q'
$\approx 2^{12}$ exponentiat

Key-generation speed

Start with many p's.
Use trial division etc.
Then try $2^{p-1} \bmod p$.
$\approx 2^6$ exponentiations
to find one prime.

Traditional key generation
chooses p, q independently.
$\approx 2^7$ exponentiations.
Faster, slightly non-uniform:
build visible primes (Maurer).

If p determines q:
$\approx 2^{12}$ exponentiations.

For $f(U) = U \bmod 2^{1008}$:
Each p determines
pool of 2^{16} possible q's.
Select randomly from pool
until finding a prime.
$\approx 2^7$ exponentiations.

For $f(U) = U \bmod 2^{1350}$:
Obtain pool of pairs (p, q)
with all different p's
and all different q's.
$\approx 2^{12}$ exponentiations.

eed

's.

etc.

d p.

ons

neration

endently.

ons.

n-uniform:

s (Maurer).

ions.

For $f(U) = U \bmod 2^{1008}$:
Each p determines
pool of 2^{16} possible q's.
Select randomly from pool
until finding a prime.
$\approx 2^7$ exponentiations.

For $f(U) = U \bmod 2^{1350}$:
Obtain pool of pairs (p, q)
with all different p's
and all different q's.
$\approx 2^{12}$ exponentiations.

Can use lattice str
to share trial divisi
Or use batch facto
Still many expone

Is there a better m

If not, might as we
take opposite appr
compress slightly
Lattice reduction i
so can afford many
before each expon

For $f(U) = U \bmod 2^{1008}$:
Each p determines
pool of 2^{16} possible q's.
Select randomly from pool
until finding a prime.
$\approx 2^7$ exponentiations.

For $f(U) = U \bmod 2^{1350}$:
Obtain pool of pairs (p, q)
with all different p's
and all different q's.
$\approx 2^{12}$ exponentiations.

Can use lattice structure
to share trial divisions.
Or use batch factorization.
Still many exponentiations.

Is there a better method?

If not, might as well
take opposite approach:
compress slightly *more*.
Lattice reduction is fast,
so can afford many p_0's
before each exponentiation.

d 2^{1008}:
s
le q's.
om pool
ne.
ons.

d 2^{1350}:
irs (p, q)
's
s.
ions.

Can use lattice structure
to share trial divisions.
Or use batch factorization.
Still many exponentiations.

Is there a better method?

If not, might as well
take opposite approach:
compress slightly *more*.
Lattice reduction is fast,
so can afford many p_0's
before each exponentiation.

Protocol violations

One user generate
Second user sees p
and generates U_2.

Security of U_2 was
assuming uniform

What if first user
and doesn't genera
uniform random U

Recall half-special
can construct rare
allowing easier fac

Can use lattice structure
to share trial divisions.
Or use batch factorization.
Still many exponentiations.

Is there a better method?

If not, might as well
take opposite approach:
compress slightly *more*.
Lattice reduction is fast,
so can afford many p_0's
before each exponentiation.

Protocol violations

One user generates U_1.
Second user sees $f(U_1)$
and generates U_2.

Security of U_2 was proven
assuming uniform random U_1.

What if first user cheats,
and doesn't generate
uniform random U_1?

Recall half-special NFS:
can construct rare f values
allowing easier factorization.

ucture
ions.
orization.
ntiations.

nethod?

ell
roach:
more.
s fast,
y p_0's
entiation.

Protocol violations

One user generates U_1.
Second user sees $f(U_1)$
and generates U_2.

Security of U_2 was proven
assuming uniform random U_1.

What if first user cheats,
and doesn't generate
uniform random U_1?

Recall half-special NFS:
can construct rare f values
allowing easier factorization.

One solution is to
generate p_1, q_1, U_1
from digits of π:
10th, 20th, 30th, e
Not random, but c
Variant: U_1 witho

Another solution is
generate p_1, q_1, U_1
from SHA-256 out

Another solution is
generate p_1, q_1, U_1
from 1955 RAND

Protocol violations

One user generates U_1.
Second user sees $f(U_1)$
and generates U_2.

Security of U_2 was proven
assuming uniform random U_1.

What if first user cheats,
and doesn't generate
uniform random U_1?

Recall half-special NFS:
can construct rare f values
allowing easier factorization.

One solution is to
generate p_1, q_1, U_1 publicly
from digits of π:
10th, 20th, 30th, etc.
Not random, but conjecturally safe.
Variant: U_1 without p_1, q_1.

Another solution is to
generate p_1, q_1, U_1 publicly
from SHA-256 output.

Another solution is to
generate p_1, q_1, U_1 publicly
from 1955 RAND tables.