Lattice Attacks on RSA

Nadia Heninger

University of Pennsylvania

September 19, 2017

Reminder: Textbook RSA

[Rivest Shamir Adleman 1977]

Public Key

$N = pq$ modulus

e encryption exponent

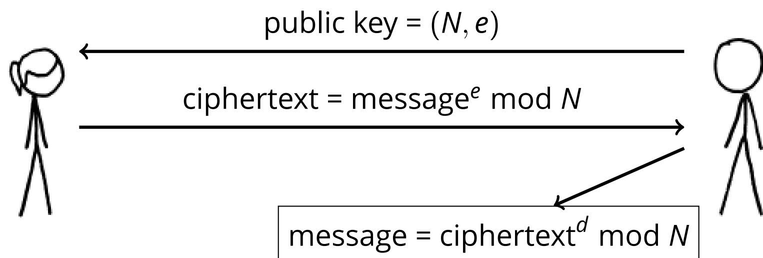
Private Key

p, q primes

d decryption exponent

$(d = e^{-1} \bmod (p-1)(q-1))$

Encryption



What's wrong with this RSA example?

```
message = Integer('squeamishossifrage',base=35)
N = random_prime(2^512)*random_prime(2^512)
c = message^3 % N
```

What’s wrong with this RSA example?

```
message = Integer('squeamishossifrage',base=35)
N = random_prime(2^512)*random_prime(2^512)
c = message^3 % N
```

```
sage: Integer(c^(1/3)).str(base=35)
'squeamishossifrage'
```

What’s wrong with this RSA example?

```
message = Integer('squeamishossifrage',base=35)
N = random_prime(2^512)*random_prime(2^512)
c = message^3 % N

sage: Integer(c^(1/3)).str(base=35)
'squeamishossifrage'
```

The message is too small.

This is why we use padding.

```
N = random_prime(2^150)*random_prime(2^150)
message = Integer('thepasswordfortodayisswordfish',base=35)
c = message^3 % N
```

```
N = random_prime(2^150)*random_prime(2^150)
message = Integer('thepasswordfortodayisswordfish',base=35)
c = message^3 % N

sage: int(c^(1/3))==message
False
```

```
N = random_prime(2^150)*random_prime(2^150)
message = Integer('thepasswordfortodayisswordfish',base=35)
c = message^3 % N
```

This is a stereotyped message. We might be able to guess the format.

```
N = random_prime(2^150)*random_prime(2^150)
message = Integer('thepasswordfortodayisswordfish',base=35)
c = message^3 % N
```

```
a = Integer('thepasswordfortodayis000000000',base=35)
```

```
N = random_prime(2^150)*random_prime(2^150)
message = Integer('thepasswordfortodayisswordfish',base=35)
c = message^3 % N
```

```
a = Integer('thepasswordfortodayis000000000',base=35)

X = Integer('xxxxxxxxx',base=35)
M = matrix([[X^3, 3*X^2*a, 3*X*a^2, a^3-c],
            [0,N*X^2,0,0],[0,0,N*X,0],[0,0,0,N]])
```

```
N = random_prime(2^150)*random_prime(2^150)
message = Integer('thepasswordfortodayisswordfish',base=35)
c = message^3 % N
```

```
a = Integer('thepasswordfortodayis000000000',base=35)

X = Integer('xxxxxxxxx',base=35)
M = matrix([[X^3, 3*X^2*a, 3*X*a^2, a^3-c],
            [0,N*X^2,0,0],[0,0,N*X,0],[0,0,0,N]])

B = M.LLL()
Q = B[0][0]*x^3/X^3+B[0][1]*x^2/X^2+B[0][2]*x/X+B[0][3]
```

```
N = random_prime(2^150)*random_prime(2^150)
message = Integer('thepasswordfortodayisswordfish',base=35)
c = message^3 % N
```

```
a = Integer('thepasswordfortodayis000000000',base=35)

X = Integer('xxxxxxxxx',base=35)
M = matrix([[X^3, 3*X^2*a, 3*X*a^2, a^3-c],
            [0,N*X^2,0,0],[0,0,N*X,0],[0,0,0,N]])

B = M.LLL()
Q = B[0][0]*x^3/X^3+B[0][1]*x^2/X^2+B[0][2]*x/X+B[0][3]

sage: Q.roots(ring=ZZ)[0][0].str(base=35)
'swordfish'
```

What's going on here? Coppersmith's method.

Theorem (Coppersmith)

We can efficiently compute up to $1/e$-fraction of the bits of an RSA-encrypted message with public exponent e if we know the rest of the plaintext.

```
sage: N.nbits()
296
sage: Integer('swordfish',base=35).nbits()
46
```

What's going on here? Coppersmith's method.

Theorem (Coppersmith)

Given a polynomial f of degree d and N, we can efficiently find all roots r_i satisfying

$$f(r_i) \equiv 0 \bmod N$$

when $|r_i| < N^{1/d}$.

In our case, our input polynomial looks like

$$f(x) = (a + x)^3 - c \equiv 0 \bmod N$$

We are looking for a root $r = \texttt{swordfish}$ satisfying

$$f(r) = (a + \texttt{swordfish})^3 - c \equiv 0 \bmod N$$

Why is this an interesting theorem?

1. A general method to solve polynomials mod N would break RSA: If c is a ciphertext,

$$x^e - c \equiv 0 \bmod N$$

   has a root $x = m$ for m our original message.

2. There is an efficient algorithm to solve equations mod primes.
   - For a composite, factor into primes, solve mod each prime, and use Chinese remainder theorem to lift solution mod N.

3. By accepting a bound on solution size, Coppersmith's method lets us solve equations without factoring N.

Coppersmith's Algorithm Outline

Input: polynomial f, modulus N.
Output: a root r modulo N.

In our example, we have $f(x) = (x+a)^3 - c$.

We will construct a new polynomial $Q(x)$ so that

$$Q(r) = 0 \qquad \text{over the integers.}$$

If we construct $Q(x)$ as

$$Q(x) = s(x)f(x) + t(x)N$$

with $s(x), t(x) \in \mathbb{Z}[x]$, then by construction

$$Q(r) \equiv 0 \bmod N$$

(In other words, $Q(x) \in \langle f(x), N \rangle$ over $\mathbb{Z}[x]$.)

Manipulating polynomials

Input: $f(x) = x^3 + f_2x^2 + f_1x + f_0$,N
Output: $Q(x) \in \langle f(x), N \rangle$ over $\mathbb{Z}[x]$.

If we only care about polynomials Q of degree 3, then

$$Q(x) = c_3 f(x) + c_2 N x^2 + c_1 N x + c_0 N$$

with $c_3, c_2, c_1, c_0 \in \mathbb{Z}$.

$$\begin{array}{rlccccccc} & c_3 & (x^3 & + & f_2x^2 & + & f_1x & + & f_0) \\ + & c_2 & & & Nx^2 & & & & \\ + & c_1 & & & & & Nx & & \\ + & c_0 & & & & & & & N \\ \hline & & Q_3x^3 & + & Q_2x^2 & + & Q_1x & + & Q_0 \end{array}$$

Manipulating polynomials as coefficient vectors

We can represent elements of $\mathbb{Z}[x]$ as coefficient vectors:

$$g_d x^d + g_{d-1} x^{d-1} + \cdots + g_0 \qquad \leftrightarrow \qquad (g_d, g_{d-1}, \ldots, g_0)$$

If we construct the matrix

$$\begin{bmatrix} 1 & f_2 & f_1 & f_0 \\ & N & & \\ & & N & \\ & & & N \end{bmatrix}$$

Then the coefficient vector representing our polynomial

$$Q(x) = c_3 f(x) + c_2 N x^2 + c_1 N x + c_0 N$$

is an integer combination of the rows of this matrix.

Polynomial coefficient vectors and lattices

The set of vectors generated by integer combinations of the rows of our matrix

$$\begin{bmatrix} 1 & f_2 & f_1 & f_0 \\ & N & & \\ & & N & \\ & & & N \end{bmatrix}$$

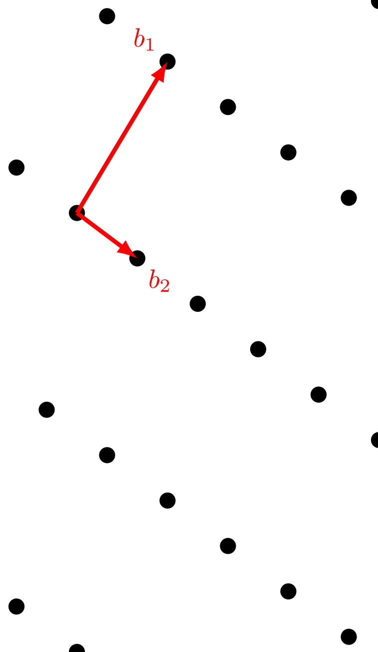
is a *lattice*.

What is a lattice?

Definition
A **lattice** is a discrete additive subgroup of $\mathbb{R}^n$.

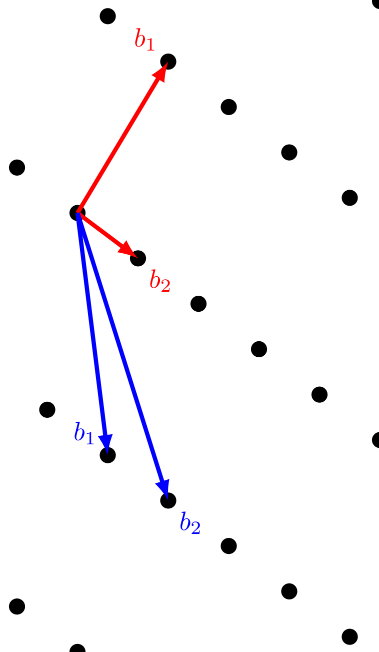
Definition
A **lattice** is a subset of $\mathbb{R}^n$ generated by integer linear combinations of some linearly independent basis $\{b_1, \ldots, b_n\}$.

- Has algebraic properties (it's a group under addition).
- Has geometric properties (it lives in $\mathbb{R}^n$ so has dot product, distance).

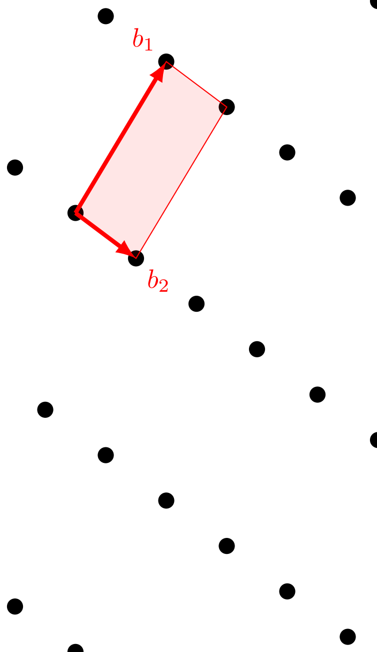


Properties of lattices: Bases

- In n dimensions a lattice has a basis of size at most n.
- The basis is not unique.



Properties of lattices: Determinant



Definition

The **determinant** of a lattice with a basis matrix B is $|\det B|$.

- The determinant is invariant for a given lattice.
- Gives volume of fundamental parallelepiped.

Properties of lattices: Minima

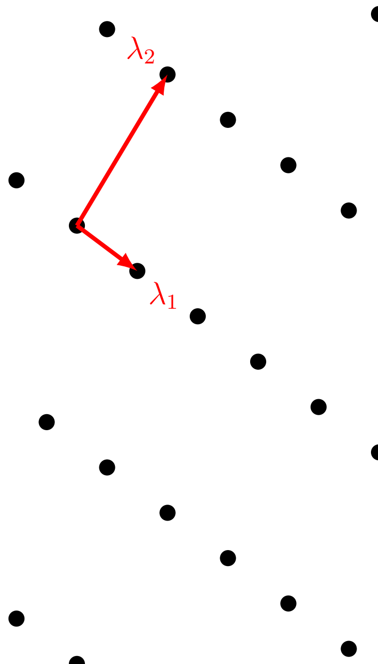
Let $\lambda_1 > 0$ be the length of the shortest vector in the lattice.

Definition

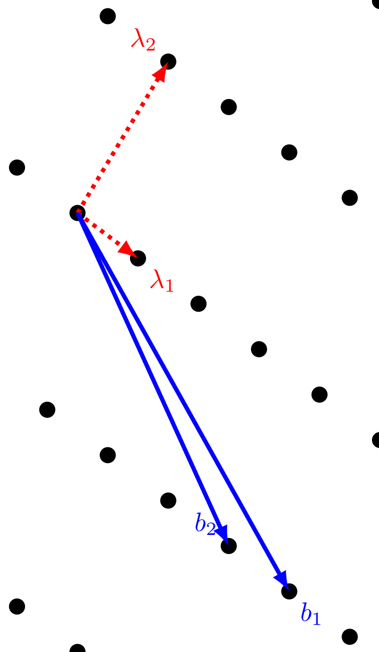
The *ith successive minimum* λ_i is the smallest radius of a ball containing *i* linearly independent lattice vectors.

Theorem (Minkowski)

$\lambda_1(L) < \sqrt{n} \det L^{1/n}$



Computational problems on lattices: SVP



Shortest Vector Problem (SVP)

Given an arbitrary basis for *L*, find the shortest vector in *L*.

- SVP is NP-hard.

Shortest Independent Vectors Problem (SIVP)

Find the *n* shortest linearly independent vectors

Computational problems on lattices: CVP

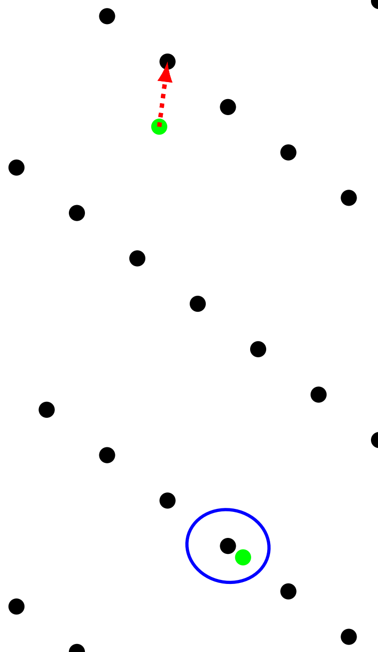
Closest Vector Problem (CVP)

Given an arbitrary basis for *L*, and a point *x* find the vector in *L* closest to *x*.

- CVP is NP-hard.

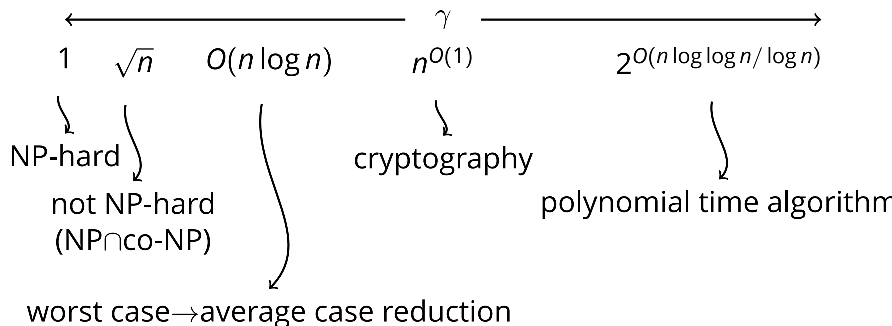
Bounded Distance Decoding (BDD)

Given an arbitrary point *x* and radius *r*, find a vector in *L* within distance *r* of *x*.



Approximation results

Search for vectors of length $\gamma\lambda_1$.



Algorithmic results

LLL

Given a basis for a lattice can in polynomial time find a *reduced* basis $\{b_i\}$ s.t.

$$|b_i| \leq 2^{(n-1)/2} \lambda_i$$

Theorem (LLL (Simplified Version))

We can find a vector of length

$$|v| < 2^{\dim L} (\det L)^{1/\dim L}$$

- In practice on random lattices, LLL finds $v = 1.02^n (\det L)^{1/\dim L}$. [Nguyen,Stehle]

BKZ

Given a lattice basis, can in time $2^{O(k)}$ find a reduced basis s.t. $|b_i| \leq k^{O(n/k)}$.

Coppersmith's method outline

Input: $f(x) \in \mathbb{Z}[x]$, $N \in \mathbb{Z}$. **Output:** r s.t. $f(r) \equiv 0 \bmod N$.

Intermediate output: $Q(x)$ such that $Q(r) = 0$ over $\mathbb{Z}$.

1. $Q(x) \in \langle f(x), N \rangle$ so $Q(r) \equiv 0 \bmod N$ by construction.
2. If $|r| < R$, then we can bound

$$\begin{aligned} |Q(r)| &= |Q_3 r^3 + Q_2 r^2 + Q_1 r + Q_0| \\ &\leq |Q_3| R^3 + |Q_2| R^2 + |Q_1| R + |Q_0| \end{aligned}$$

3. If $|Q(r)| < N$ and $Q(r) \equiv 0 \bmod N$ then $Q(r) = 0$.

We want a Q in our lattice with short coefficient vector!

Coppersmith's method outline

1. Construct a matrix of coefficient vectors of elements of $\langle f(x), N \rangle$.

2. Run a lattice basis reduction algorithm on this matrix.

3. Construct a polynomial Q from the shortest vector output.

4. Factor Q to find its roots.

Running Coppersmith's method on our example

Input: $f(x) = (x+a)^3 - c, N$
Output: $r < R$ such that $f(r) \equiv 0 \bmod N$.

1. Construct lattice basis

$$\begin{bmatrix} R^3 & 3aR^2 & 3a^2R & a^3 - c \\ & NR^2 & & \\ & & NR & \\ & & & N \end{bmatrix}$$

$$\dim L = 4$$

$$\det L = R^6 N^3$$

Factor of R is so that $Q(r) \leq |v|$ for $v \in L$.

Running Coppersmith's method on our example

Input: $f(x) = (x+a)^3 - c$, N
Output: $r < R$ such that $f(r) \equiv 0 \bmod N$.

1. Construct lattice basis
$$\begin{bmatrix} R^3 & 3aR^2 & 3a^2R & a^3 - c \\ & NR^2 & & \\ & & NR & \\ & & & N \end{bmatrix}$$
$$\dim L = 4$$
$$\det L = R^6 N^3$$
Factor of R is so that $Q(r) \leq |v|$ for $v \in L$.

2. Ignoring approximation factor, we can solve when
$$|Q(r)| \leq |v_1| \leq \det L^{1/\dim L} < N$$
$$(R^6 N^3)^{1/4} < N$$
$$R < N^{1/6}$$

In my example I chose $\lg N = 296$, $\lg r = 46$.

Achieving the Coppersmith bound $r < N^{1/d}$

1. Generate lattice from subset of $\langle f(x), N\rangle^k$.
2. Allow higher degree polynomials.

Theorem (CHHS 2016)

It is not possible to solve for $r > N^{1/d}$ with any method that constructs auxiliary polynomial $Q(x)$.

Countermeasures for real-world RSA

- Must use padding scheme with cryptographically secure randomized padding for RSA.
  - PKCS#1v1.5 widely used in practice, not CCA-secure.
  - OAEP is CCA-secure but not widely used.
- Current recommendation: Use RSA exponent $e \geq 65537$.

```
p = random_prime(2^512); q = random_prime(2^512)
N = p*q

a = p - (p % 2^86)
```

```
p = random_prime(2^512); q = random_prime(2^512)
N = p*q

a = p - (p % 2^86)

sage: hex(a)
'a9759e8c9fba8c0ec3e637d1e26e7b88befeb03ac199d1190
76e3294d16ffcaef629e2937a03592895b29b0ac708e79830
4330240bc000000000000000000000'
```

Key recovery from partial information.

```
p = random_prime(2^512); q = random_prime(2^512)
N = p*q

a = p - (p % 2^86)



X = 2^86
M = matrix([[X^2, 2*X*a, a^2], [0, X, a], [0, 0, N]])
B = M.LLL()
```

```
p = random_prime(2^512); q = random_prime(2^512)
N = p*q

a = p - (p % 2^86)



X = 2^86
M = matrix([[X^2, 2*X*a, a^2], [0, X, a], [0, 0, N]])
B = M.LLL()

Q = B[0][0]*x^2/X^2+B[0][1]*x/X+B[0][2]

sage: a+Q.roots(ring=ZZ)[0][0] == p
True
```

Partial key recovery and finding solutions modulo divisors

Theorem (Coppersmith)

Given half the bits (most or least significant) of p, we can factor N in polynomial time.

Partial key recovery and finding solutions modulo divisors

Theorem (Howgrave-Graham)

Given degree d polynomial f, integer N, we can find roots r modulo divisors B of N satisfying

$$f(r) \equiv 0 \bmod B$$

for $|B| > N^{\beta}$, *when* $|r| < N^{\beta^2/d}$.

For RSA partial key recovery, we have

$$f(x) = a + x$$

and we want to find a solution vanishing modulo $p \approx N^{1/2}$ for some $p \mid N$.

Partial key recovery example

Input: $f(x) = a + x, N$
Output: $r < R \quad \text{s.t.} \quad f(r) \equiv 0 \bmod p, \quad p|N, \quad p \geq N^{1/2}$

1. We chose the polynomial basis $(x+a)^2, (x+a), N$.

Partial key recovery example

Input: $f(x) = a + x, N$
Output: $r < R \quad \text{s.t.} \quad f(r) \equiv 0 \bmod p, \quad p|N, \quad p \geq N^{1/2}$

1. We chose the polynomial basis $(x+a)^2, (x+a), N$.
2. This corresponds to a lattice basis

$$\begin{bmatrix} R^2 & 2Ra & a^2 \\ 0 & R & a \\ & & N \end{bmatrix}$$

$\dim L = 3$
$\det L = R^3 N$

Partial key recovery example

Input: $f(x) = a + x, N$
Output: $r < R \quad \text{s.t.} \quad f(r) \equiv 0 \bmod p, \quad p | N, \quad p \geq N^{1/2}$

1. We chose the polynomial basis $(x + a)^2, (x + a), N$.
2. This corresponds to a lattice basis

$$\begin{bmatrix} R^2 & 2Ra & a^2 \\ 0 & R & a \\ & & N \end{bmatrix} \qquad \begin{aligned} \dim L &= 3 \\ \det L &= R^3 N \end{aligned}$$

3. LLL will find us a vector of size about $|v| \approx \det L^{1/\dim L}$.

Partial key recovery example

Input: $f(x) = a + x, N$
Output: $r < R \quad \text{s.t.} \quad f(r) \equiv 0 \bmod p, \quad p|N, \quad p \geq N^{1/2}$

1. We chose the polynomial basis $(x+a)^2, (x+a), N$.
2. This corresponds to a lattice basis

$$\begin{bmatrix} R^2 & 2Ra & a^2 \\ 0 & R & a \\ & & N \end{bmatrix} \qquad \begin{aligned} \dim L &= 3 \\ \det L &= R^3 N \end{aligned}$$

3. LLL will find us a vector of size about $|v| \approx \det L^{1/\dim L}$.
4. The algorithm will find the root when we have

$$\begin{aligned} |Q(r)| \leq |v| \approx \det L^{1/\dim L} &< p \\ (R^3 N)^{1/3} &< N^{1/2} \\ R &< N^{1/6} \end{aligned}$$

We had $\lg r = 86$ and $\lg p = 512$.

Partial key recovery and related attacks

RSA particularly susceptible to partial key recovery attacks.

- Can factor given 1/2 bits of p. [Coppersmith 96]
- Can factor given 1/4 bits of d. [Boneh Durfee Frankel 98]
- Can factor given 1/2 bits of $d \bmod (p-1)$. [Blömer May 03]

Applying partial key recovery in the wild

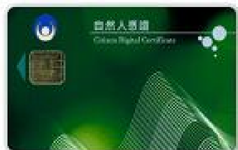
Taiwan Citizen Digital Certificate

[Bernstein, Chang, Cheng, Chou, Heninger, Lange, van Someren Asiacrypt 2013]

Many countries adopting national PKI.

Taiwan’s smart card IDs allow citizens to

- file income taxes,
- update car registrations,
- transact with government agencies,
- interact with companies (e.g. Chunghwa Telecom) online.



Taiwan Citizen Digital Certificate

- Smart cards are issued by the government.
- FIPS-140 and Common Criteria Level 4+ certified.
- RSA keys are generated on card.
- Certificates stored on national LDAP directory. This is publicly accessible to enable citizen-to-citizen and citizen-to-commerce interactions.

Certificate of Chen-Mou Cheng

```
Data: Version: 3 (0x2)
Serial Number: d7:15:33:8e:79:a7:02:11:7d:4f:25:b5:47:e8:ad:38
Signature Algorithm: sha1WithRSAEncryption
Issuer: C=TW, O=XXX
Validity
     Not Before: Feb 24 03:20:49 2012 GMT
     Not After : Feb 24 03:20:49 2017 GMT
Subject: C=TW, CN=YYY serialNumber=0000000112831644
Subject Public Key Info:
     Public Key Algorithm: rsaEncryption
Public-Key: (2048 bit) Modulus:
     00:bf:e7:7c:28:1d:c8:78:a7:13:1f:cd:2b:f7:63:
     2c:89:0a:74:ab:62:c9:1d:7c:62:eb:e8:fc:51:89:
     b3:45:0e:a4:fa:b6:06:de:b3:24:c0:da:43:44:16:
     e5:21:cd:20:f0:58:34:2a:12:f9:89:62:75:e0:55:
     8c:6f:2b:0f:44:c2:06:6c:4c:93:cc:6f:98:e4:4e:
     3a:79:d9:91:87:45:cd:85:8c:33:7f:51:83:39:a6:
     9a:60:98:e5:4a:85:c1:d1:27:bb:1e:b2:b4:e3:86:
     a3:21:cc:4c:36:08:96:90:cb:f4:7e:01:12:16:25:
     90:f2:4d:e4:11:7d:13:17:44:cb:3e:49:4a:f8:a9:
     a0:72:fc:4a:58:0b:66:a0:27:e0:84:eb:3e:f3:5d:
     5f:b4:86:1e:d2:42:a3:0e:96:7c:75:43:6a:34:3d:
     6b:96:4d:ca:f0:de:f2:bf:5c:ac:f6:41:f5:e5:bc:
     fc:95:ee:b1:f9:c1:a8:6c:82:3a:dd:60:ba:24:a1:
     eb:32:54:f7:20:51:e7:c0:95:c2:ed:56:c8:03:31:
     96:c1:b6:6f:b7:4e:c4:18:8f:50:6a:86:1b:a5:99:
     d9:3f:ad:41:00:d4:2b:e4:e7:39:08:55:7a:ff:08:
     30:9e:df:9d:65:e5:0d:13:5c:8d:a6:f8:82:0c:61:
     c8:6b
Exponent: 65537 (0x10001)

        .
        .
        .
```

Factoring public RSA keys

April 2012: Downloaded certificates from LDAP server:

- 2,300,000 1024-bit RSA public keys
- 740,000 2048-bit RSA public keys

Factoring public RSA keys

April 2012: Downloaded certificates from LDAP server:

- 2,300,000 1024-bit RSA public keys
- 740,000 2048-bit RSA public keys

- Factored 103 RSA-1024 public keys with GCD algorithm

  If we have two RSA moduli

  $$N_1 = pq_1 \qquad N_2 = pq_2$$

  Then $\gcd(N_1, N_2) = p$ and we can factor both moduli.

This should *never* happen to properly generated keys.

Investigating the factors...

Most common factor appears 46 times

```
c0000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
000000000000000000000000000002f9
```

Investigating the factors...

Most common factor appears 46 times

```
c0000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
000000000000000000000000000002f9
```

which is the next prime after $2^{511} + 2^{510}$.

Investigating the factors...

Most common factor appears 46 times

```
c0000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
000000000000000000000000000002f9
```

which is the next prime after $2^{511} + 2^{510}$.
The next most common factor, repeated 7 times, is

```
c9242492249292499249492449242492
24929249924949244924249224929249
92494924492424922492924992494924
492424922492924992494924492424e5
```

Several other factors exhibit such a pattern.

How is this pattern generated?

1100100100100100001001001001000100100100100101001001001001001
1001001001001001010010010010001001001001000010010010010010
0010010010010010100100100100110010010010010010100100100100100
0100100100100100001001001001000100100100100101001001001001001
1001001001001001010010010010001001001001000010010010010010
0010010010010010100100100100110010010010010010100100100100100
0100100100100100001001001001000100100100100101001001001001001
1001001001001001010010010010001001001001000010010011100101

How is this pattern generated?

Swap every 16 bits in a 32 bit word

0010010010010010 1100100100100100 1001001001001001 0010010010010010
0100100100100100 1001001001001001 0010010010010010 0100100100100100
1001001001001001 0010010010010010 0100100100100100 1001001001001001
0010010010010010 0100100100100100 1001001001001001 0010010010010010
0100100100100100 1001001001001001 0010010010010010 0100100100100100
1001001001001001 0010010010010010 0100100100100100 1001001001001001
0010010010010010 0100100100100100 1001001001001001 0010010010010010
0100100100100100 1001001001001001 0010010011100101 0100100100100100

How is this pattern generated?

Realign

001001001001001011001001001001001001001001001001001001001001001001001001001001001
001001001001001001001001001001001001001001001001001001001001001001001001001001001
001001001001001001001001001001001001001001001001001001001001001001001001001001001
001001001001001001001001001001001001001001001001001001001001001001001001001001001
001001001001001001001001001001001001001001001001001001001001001001001001001001001
001001001001001001001001001001001001001001001001001001001001001001001001001001001
001001001001001001001001001001001001001001001001001001001001001001001001001001001
00100100100100100100100111001010100100100100100

How is this pattern generated?

Realign

001001001001001011001001001001001001001001001001001001001001001001
001001001001001001001001001001001001001001001001001001001001001001
001001001001001001001001001001001001001001001001001001001001001001
001001001001001001001001001001001001001001001001001001001001001001
001001001001001001001001001001001001001001001001001001001001001001
001001001001001001001001001001001001001001001001001001001001001001
001001001001001001001001001001001001001001001001001001001001001001
00100100100100100100100111001010100100100100100

The 119 factors had patterns of period 1, 3, 5, and 7.

Shared prime generation

Hypothesized key generation process for weak primes:

1. Choose a bit pattern of length 1, 3, 5, or 7 bits.
2. Repeat it to cover 512 bits.
3. For every 32-bit word, swap the lower and upper 16 bits.
4. Fix the most significant two bits to 11.
5. Find the next prime greater than or equal to this number.

Shared prime generation

Hypothesized key generation process for weak primes:

1. Choose a bit pattern of length 1, 3, 5, or 7 bits.
2. Repeat it to cover 512 bits.
3. For every 32-bit word, swap the lower and upper 16 bits.
4. Fix the most significant two bits to 11.
5. Find the next prime greater than or equal to this number.

Factoring by trial division

Enumerating all patterns of this form factored 18 more keys.

Extending to patterns of length 9 gave us 4 more keys.

Some more prime factors

```
c0000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
000000000000000000000000000101ff

c0000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000100000177
```

Some more prime factors

```
c0000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
000000000000000000000000000101ff

c0000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000100000177
```

Hypothesis: There might be more prime factors of the form

$$p = 2^{511} + 2^{510} + x$$

where x is “small”.

Factoring Taiwanese keys from partial information

We observed RNG getting “stuck”. What if RNG becomes unstuck in least significant bits?

Exactly our RSA key recovery example:

- A 3-dimensional lattice can let us find errors as big as $N^{1/6}$.
- Ran 3-dimensional lattice with every pattern against every key (1 hour per pattern, 164 patterns).
- Factored 39 new keys (and all but 2 of keys factored via GCD).

First example we know of applying Coppersmith’s method to RSA keys in the wild.

```
p = random_prime(2^512); q = random_prime(2^512)
N = p*q

d = random_prime(2^254)
e = inverse_mod(d,(p-1)*(q-1))
```

d is relatively small. (But not that small.)

```
p = random_prime(2^512); q = random_prime(2^512)
N = p*q

d = random_prime(2^254)
e = inverse_mod(d,(p-1)*(q-1))



X = 2^764; Y = 2^254
M = matrix([[X, e*Y, -1], [0, Y*(N+1), 0], [0, 0, N+1]])

B = M.LLL()
```

```
p = random_prime(2^512); q = random_prime(2^512)
N = p*q

d = random_prime(2^254)
e = inverse_mod(d,(p-1)*(q-1))



X = 2^764; Y = 2^254
M = matrix([[X, e*Y, -1], [0, Y*(N+1), 0], [0, 0, N+1]])

B = M.LLL()

sage: abs(B[0][0]/X) == d
True
```

Small RSA private exponent with lattices

Theorem (Wiener)

We can efficiently compute d when $d < N^{1/4}$.

The *RSA equation* is

$$ed \equiv 1 \bmod (p-1)(q-1)$$

$$ed = 1 + k(N - (p+q) + 1)$$

Small RSA private exponent with lattices

Theorem (Wiener)

We can efficiently compute d when $d < N^{1/4}$.

The *RSA equation* is

$$ed \equiv 1 \bmod (p-1)(q-1)$$
$$ed = 1 + k(N - (p+q) + 1)$$

Let $s = p + q$.

We would like to solve

$$ed = 1 - ks + k(N+1)$$

for d, k, s unknown.

We know $k \leq d$ and $s \approx \sqrt{N}$.

Small RSA private exponent with lattices

We would like to solve

$$ed = 1 - ks + k(N+1)$$

for d, k, s unknown.

Can write as

$$ks + ed - 1 \equiv 0 \bmod (N+1)$$

We would like to find small solutions $x = ks, y = d$ for

$$f(x,y) = x + ey - 1 \equiv 0 \bmod (N+1).$$

Small RSA private exponent with lattices

Would like to solve equation

$$f(x,y) = x + ey - 1 \equiv 0 \bmod (N+1)$$

for solution $x = ks, y = d$. Bound $|d| < X$, $|ks| < Y$.

Create lattice basis

$$\begin{bmatrix} X & eY & -1 \\ & Y(N+1) & \\ & & (N+1) \end{bmatrix}$$

$\dim L = 3$
$\det L = XY(N+1)^2$

Corresponds to $x + ey - 1$, $y(N+1)$, $(N+1)$.

Small RSA private exponent with lattices

We will find a coefficient vector for a polynomial $Q(x, y)$ satisfying $Q(d, ks) = 0$ over $\mathbb{Z}$ when

$$\begin{aligned} |Q(d, ks)| \leq \dim L^{1/\det L} &< N + 1 \\ (XY(N+1)^2)^{1/3} &< N + 1 \\ XY &< N + 1 \end{aligned}$$

We want to find solutions $d < X$, $ks < Y$, and we know

$$k \leq d \qquad s \approx \sqrt{N}$$

So when $d < X = N^{1/4}$, we can set $Y \approx N^{3/4}$ and $X \approx N^{1/4}$ and guarantee

$$Q(d, ks) = 0.$$

Small RSA private exponent with lattices

We will find a coefficient vector for a polynomial $Q(x, y)$ satisfying $Q(d, ks) = 0$ over $\mathbb{Z}$ when

$$
\begin{aligned}
|Q(d, ks)| \leq \dim L^{1/\det L} &< N + 1 \\
(XY(N+1)^2)^{1/3} &< N + 1 \\
XY &< N + 1
\end{aligned}
$$

We want to find solutions $d < X$, $ks < Y$, and we know

$$k \leq d \qquad s \approx \sqrt{N}$$

So when $d < X = N^{1/4}$, we can set $Y \approx N^{3/4}$ and $X \approx N^{1/4}$ and guarantee

$$Q(d, ks) = 0.$$

Lattice is actually finding equation

$$dx + (ks - 1)y - d = 0$$

Theorem (Boneh Durfee)

We can efficiently compute d when $d < N^{0.292}$.

Boneh and Durfee use Coppersmith's method to find small solutions $x = k, y = (p + q)$ to

$$xy - (N + 1)x - 1 \equiv 0 \bmod e$$

Improvements: Use higher multiplicities and degree, examine sublattice.

Multivariate Coppersmith

Input: Multivariate polynomial $f(x_1, \ldots, x_m)$

Output: Integers $r_1, \ldots, r_m$ such that

$$f(r_1, \ldots, r_m) \equiv 0 \bmod N$$

Same approach works in this case, with some tweaks:

- To find solutions we solve a system of m equations taken from the short vectors in our lattice.
- Theorems are generally heuristic because we don't know solution set is finite.
- Results are more ad hoc in general.

Open problem: Give a useful characterization of when multivariate Coppersmith method works.

Some more Taiwanese factors

```
ffffaa55ffffffffff3cd9fe3ffff676
fffffffffffe00000000000000000000
00000000000000000000000000000000
000000000000000000000000000000009d
```

```
c000b800000000000000000000000000
00000000000000000000000000000000
00000680000000000000000000000000
00000000000000000000000000000251
```

Factoring Taiwanese keys with lattices

Returning to Taiwan example. What if RNG becomes stuck after most significant bits?

Want to find solutions to the equation

$$a + 2^t x + y \equiv 0 \bmod p \qquad p | N$$

This lets us factor keys with pattern errors in most, least, or middle bits by setting t.

Ran on 20 most common patterns and factored 13 more keys.

Factoring with Bivariate Coppersmith

Search for prime factors p of N of the form

$$p = a + 2^t x + y$$

Factoring with Bivariate Coppersmith

Search for prime factors p of N of the form

$$p = a + 2^t x + y$$

Algorithm (Expected Algorithm)

1. *Generate lattice from multiples of* $f(x, y) = a + 2^t x + y$, N.
2. *Run LLL and take two short polynomials* $Q_1(x, y)$, $Q_2(x, y)$.
3. *Solve for* r_1, r_2 *satisfying* $Q_1(r_1, r_2) = Q_2(r_1, r_2) = 0$.
4. *Check if* $\gcd(a + 2^t r_1 + r_2, N)$ *is nontrivial.*

Factoring with Bivariate Coppersmith

Search for prime factors p of N of the form

$$p = a + 2^t x + y$$

Algorithm (Expected Algorithm)

1. *Generate lattice from multiples of* $f(x, y) = a + 2^t x + y$, N.
2. *Run LLL and take two short polynomials* $Q_1(x, y)$, $Q_2(x, y)$.
3. *Solve for* r_1, r_2 *satisfying* $Q_1(r_1, r_2) = Q_2(r_1, r_2) = 0$.
4. *Check if* $\gcd(a + 2^t r_1 + r_2, N)$ *is nontrivial.*

- Analysis says 10-dimensional lattices let us solve for

$$|r_1 r_2| < N^{1/10}.$$

- For 1024-bit N, should have $|r_1 r_2| < 2^{102}$.

Tricky Details: Algebraic Dependence

- Need *two* equations $Q_1(x, y)$, $Q_2(x, y)$.
- Coefficient vectors in lattice are linearly independent, but polynomials might have algebraic relation.

Tricky Details: Algebraic Dependence

- Need *two* equations $Q_1(x, y)$, $Q_2(x, y)$.
- Coefficient vectors in lattice are linearly independent, but polynomials might have algebraic relation.

Assumption

The short vectors of the LLL-reduced basis correspond to algebraically independent polynomials.

Tricky Details: Algebraic Dependence

- Need *two* equations $Q_1(x, y)$, $Q_2(x, y)$.
- Coefficient vectors in lattice are linearly independent, but polynomials might have algebraic relation.

Assumption

The short vectors of the LLL reduced basis correspond to algebraically independent polynomials.

This assumption **failed** in our experiments.

Tricky Details: Algebraic Dependence

- Need *two* equations $Q_1(x, y)$, $Q_2(x, y)$.
- Coefficient vectors in lattice are linearly independent, but polynomials might have algebraic relation.

Assumption

The short vectors of the LLL reduced basis correspond to algebraically independent polynomials.

This assumption **failed** in our experiments.

- But we could sill solve from single equation!

Bivariate Coppersmith details

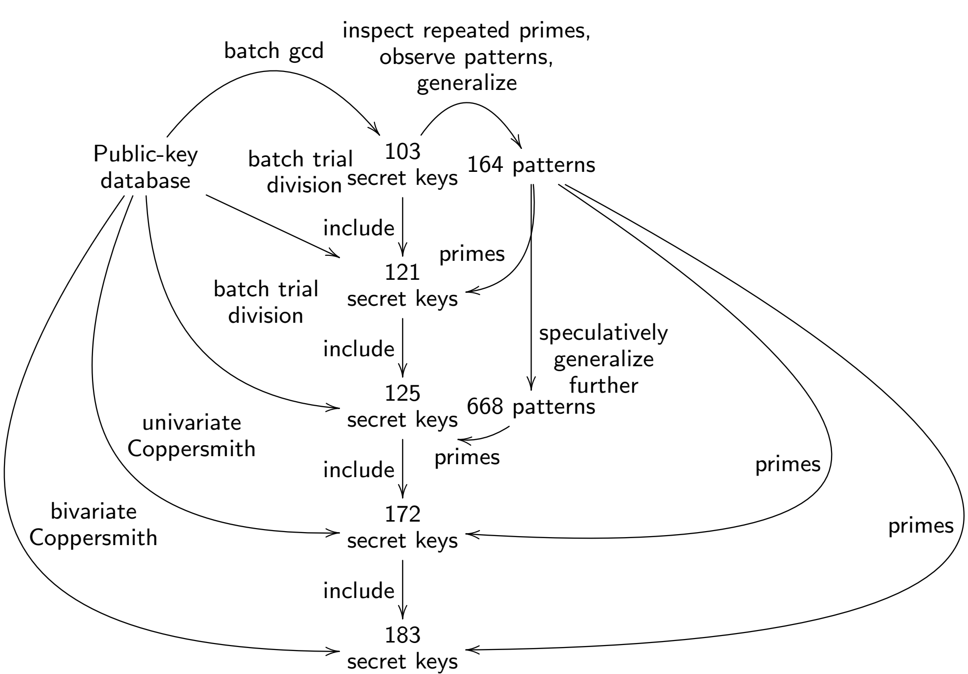
A mystery: The attack works *much* better in practice than theory guarantees.

Smallest lattice with guaranteed solution has dimension 10. But dimension 6 worked too.

Open problem: Why?

For finding solution, equations were not algebraically independent, but could find a solution anyway.

Open problem: Why?


batch gcd
inspect repeated primes,
observe patterns,
generalize
Public-key
database
batch trial
division
103
secret keys
164 patterns
include
primes
121
secret keys
batch trial
division
speculatively
generalize
further
include
125
secret keys
668 patterns
univariate
Coppersmith
primes
include
primes
172
secret keys
bivariate
Coppersmith
primes
include
183
secret keys

Why are government-certified smartcards generating weak keys?

Best practices and standards in hardware random number generation require

- designers to characterize the entropy generated by circuits
- testing of the signal from the entropy source at run time
- post-processing by running output through cryptographic hash function

Card behavior very clearly not FIPS-compliant.

Why are government-certified smartcards generating weak keys?

Best practices and standards in hardware random number generation require

- designers to characterize the entropy generated by circuits
- testing of the signal from the entropy source at run time
- post-processing by running output through cryptographic hash function

Card behavior very clearly not FIPS-compliant.

Hypothesized failure:

- Hardware RNG has underlying weakness that causes failure in some situations.
- Card software not operated in FIPS mode
  $\Longrightarrow$ no testing or post-processing RNG output.

Countermeasures against lattice attacks

- Don't use RSA for encryption.
- If you must use RSA, use RSA-OAEP padding for ciphertexts.
- If you must use RSA, don't use small e or small d.
- Use good random number generators.
- Avoid side channels in your implementations.

Conclusions

Widely used crypto can fail in interesting ways that affect real-world security.

Complex systems can experience cascading failures that make otherwise good crypto trivial to break.

Find (and fix!) cryptographic vulnerabilities in the wild by beating public keys over the head with math.