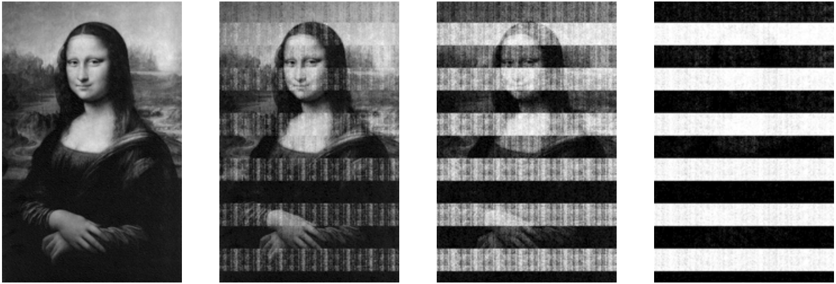
How to recover cryptographic keys from partial information

Nadia Heninger

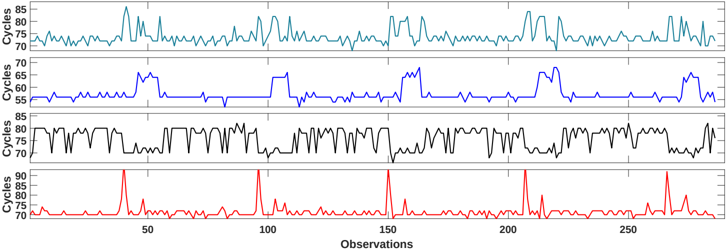
University of Pennsylvania

December 10, 2018

Motivation: Side-channel attacks



5s. 30s. 1m. 5m.



Textbook RSA

[Rivest Shamir Adleman 1977]

Public Key

$N = pq$ modulus

e encryption exponent

$e = 65537$ in practice

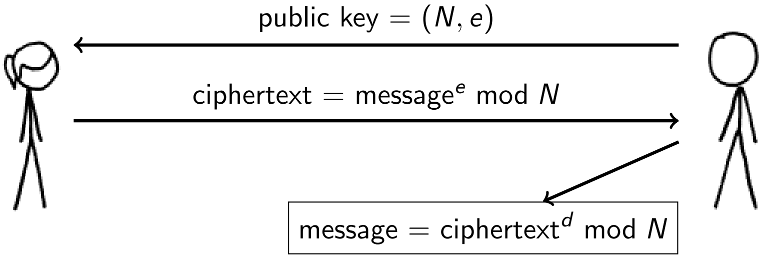
Private Key

p, q primes

d decryption exponent

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

Encryption



CRT RSA

For efficiency, RSA implementations typically precompute

$$d_p \equiv d \bmod (p-1) \qquad\qquad d_q \equiv d \bmod (q-1)$$

Then decrypt or sign by computing

$$m_p = c^{d_p} \bmod p \qquad\qquad m_q = c^{d_q} \bmod q.$$

Let $u = q^{-1} \bmod p$. Then we can reconstruct m as

$$m = m_q + qu(m_p - m_q)$$

Partial key recovery for RSA:

An attacker learns some information about p or q. Can they efficiently factor N?

More realistic scenario:

An attacker learns some information about d_p and d_q. Can they efficiently recover d?

Factoring with Partial Information

Already-factored modulus: Trivial.

Factoring with Partial Information

One factor known: Trivial. (Division)

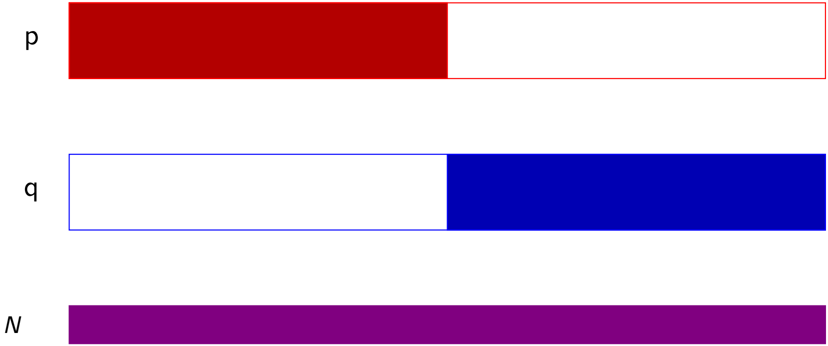
Factoring from CRT coefficients

With high probability, $\gcd(a^{ed_p - 1} - 1, N) = p$ for random a.

Factoring with Partial Information

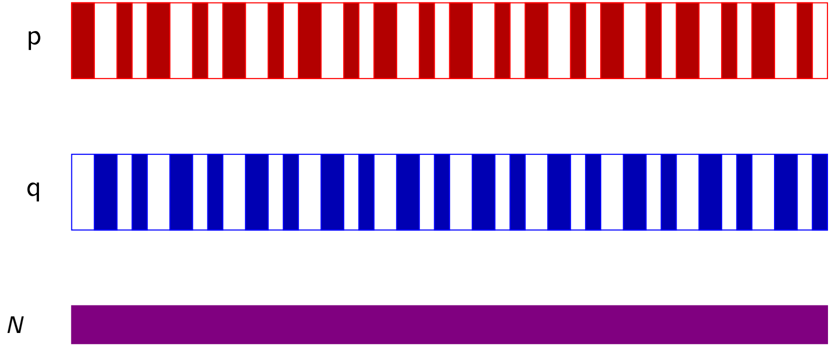
Neither factor known: Subexponential time. (Number field sieve)

Factoring with Partial Information



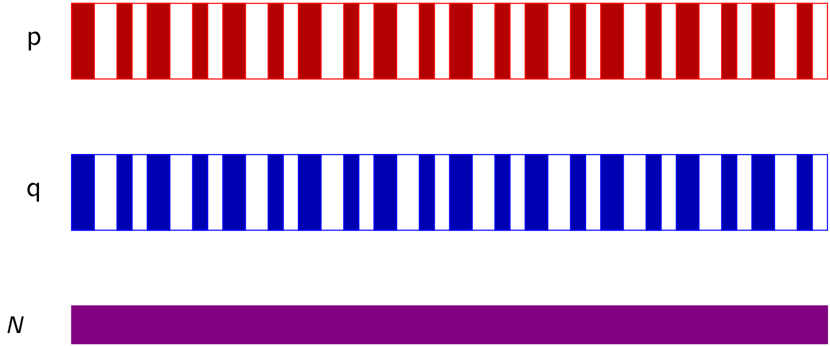
Trivial. (Division + fixing a few bits.)

Factoring with Partial Information



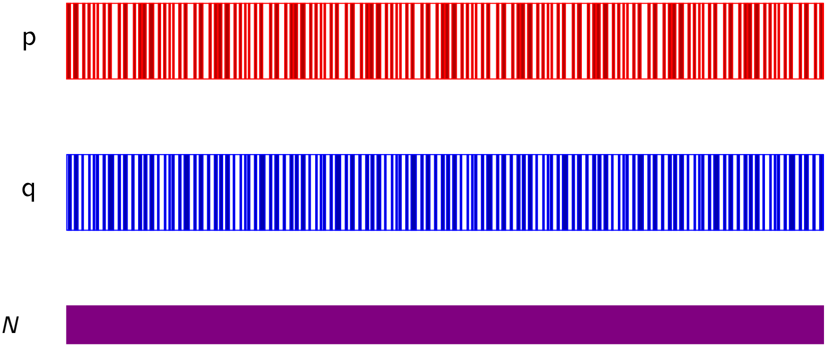
Trivial. (Branch and prune.) [Heninger Shacham 09]

Factoring with Partial Information



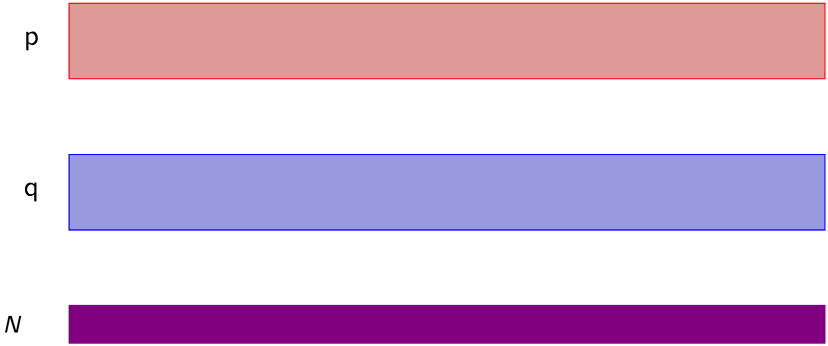
Expected polynomial time. (Branch and prune.) [Heninger Shacham 09]

Factoring with Partial Information



Expected polynomial time for $\geq 50\%$ of bits known. [Heninger Shacham 09]

Factoring with partial information



Expected polynomial time when information/bit $\geq .5$.
[Paterson Polychroniadou Sibborn 2012] [BBGGBHLvVY 2017]

Branch and prune family of algorithms.

(RSA key recovery with redundancy.)

RSA key recovery with erasures

Remove all but a δ-fraction of the bits, chosen at random, from an RSA private key.

(Flip a coin at each bit of the key. With probability δ, the attacker gets to see the bit's value.)

Simplest case

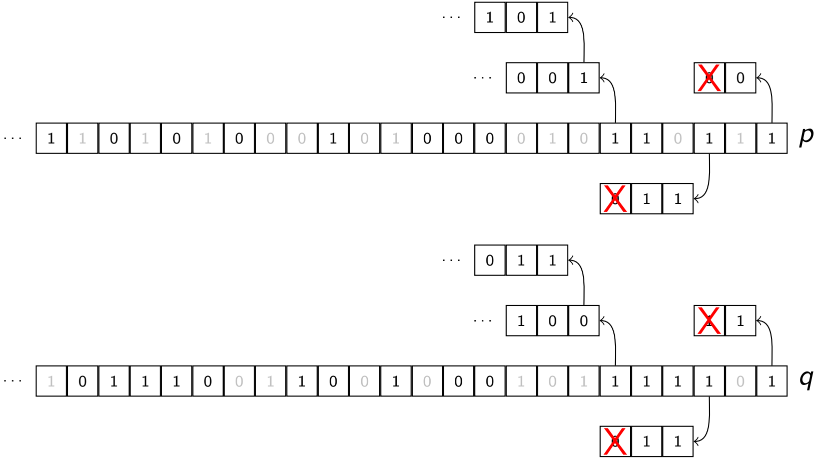
$N = pq$, get random bits of p and q.
N is known.

How to efficiently reconstruct the key?

"For example the paper tries to factor $N = pq$ by writing it as a set of binary equations over the bits of p and q."

– J.S. Coron, "Ten Reasons why a Paper is Rejected from a Crypto Conference"

Branch and Prune Algorithm



At each step, verify that

- $pq = N \bmod 2^i$ at each step i.
- bits match known information.

Prune otherwise.

Heuristic Running Time Analysis

[Heninger Shacham 2009]

Assumption:

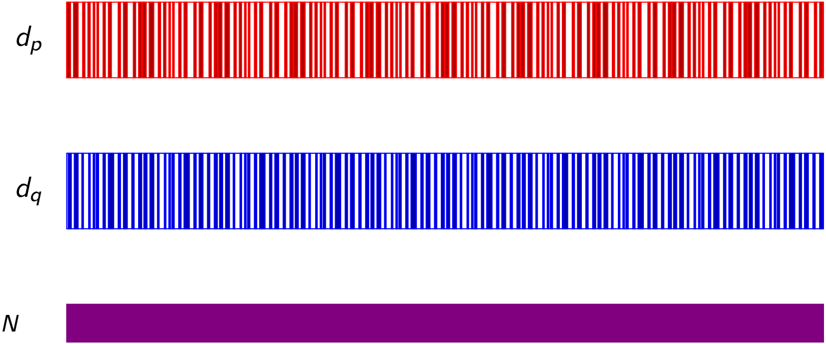
After an incorrect guess, induced bits are uniformly random.

Theorem (Heuristic, [BbGGBHLvVY 17])

When the average amount of self-information known is $> .5$ bit per bit, the algorithm runs in expected linear time.

Key recovery for CRT-RSA with missing bits

[IGIES 2015]



Branch-and-prune works the same as before. (Must brute force k_p.)

Key recovery for CRT-RSA with missing bits

RSA equations: $ed_p = 1 + k_p(p-1) \qquad ed_q = 1 + k_q(q-1)$

Rearrange: $(ed_p - 1 + k_p)(ed_q - 1 + k_q) = k_p k_q N$

Then k_p, k_q are related as:

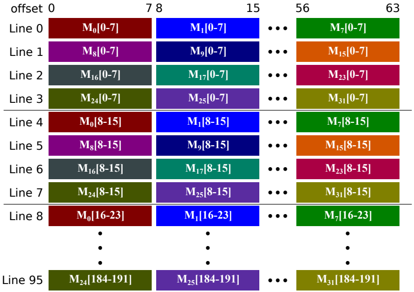
$$(k_p - 1)(k_q - 1) \equiv k_p k_q N \bmod e$$

We do not know k_p or k_q, but we need to brute force at most e possible pairs.

For each guess of k_p, k_q, apply branch and prune to RSA equations.

Application: Cachebleed attack

[Yarom Genkin Heninger 2016]



OpenSSL cache timing countermeasures:

- fixed-window exponentiation
- scatter multipliers in memory.

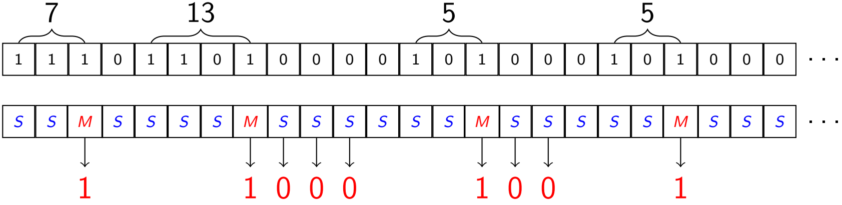
- Intel introduced cache banks to serve parts of cache.
- Cache bank conflicts produce timing differences.
- For windowed exponentiation, learn 3 LSBs of every 5 bits.
- 4096-bit key: 3.5 minutes on 36 cores, mostly brute-forcing k.

Application: Left-to-right square-and-multiply leak

[Bernstein Breitner Genkin Groot Bruinderink Heninger Lange van Vredendaal Yarom 2017]

- Libgcrypt sliding window implementation not constant time.

Flush+Reload cache attack leaks square and multiply sequence.



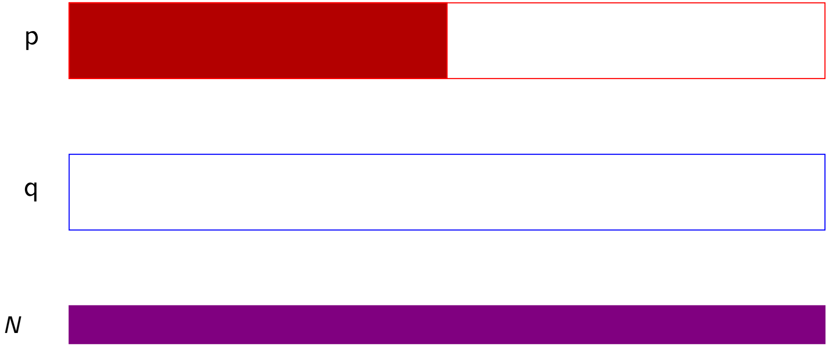
Only 40% of bits directly leaked → not enough to efficiently recover.

- We can derive implicit information from square-and-multiply sequence and efficiently recover key.

Coppersmith/lattice family of algorithms.

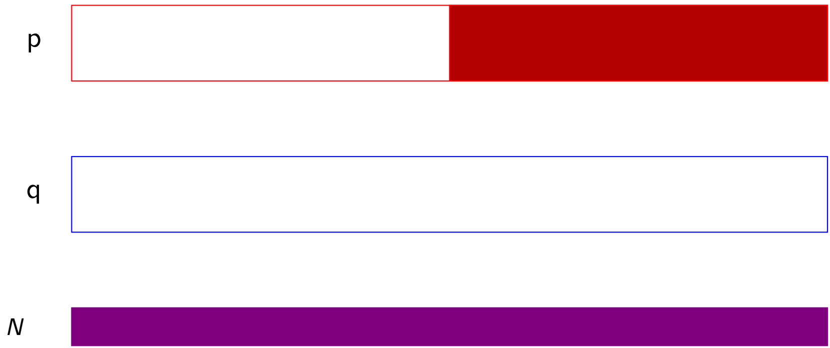
(RSA key recovery without redundancy.)

Factoring with Partial Information



Polynomial time. (Lattice basis reduction.) [Coppersmith 96]

Factoring with Partial Information



Polynomial time. (Lattice basis reduction.) [Coppersmith 1996]

Theorem (Coppersmith 1996)

Let $N = pq$ with $p, q \approx \sqrt{N}$. Given half the bits (most or least significant) of p, we can factor N in polynomial time.

```
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, X*a, 0], [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, X*a, 0], [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 (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 $p \mid N$.

Coppersmith's Algorithm Outline

Input: polynomial f, integer N, bound $0 < \beta \leq 1$.
Output: a root r modulo p, $p|N$, $p \geq N^{\beta}$.

In our example, we have $f(x) = x + a$.

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

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

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

Manipulating polynomials

Input: $f(x) = x + a$, N, β.
Output: $Q(x) \in \langle f(x), N \rangle$ over $\mathbb{Z}[x]$.

If we only care about polynomials Q of degree 2, then

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

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

$$\begin{array}{cccccccc}
 & c_2 & (x^2 & + & xa & + & 0) \\
+ & c_1 & (& & x & + & a) \\
+ & c_0 & & & & & N \\
\hline
 & & Q_2 x^2 & + & Q_1 x & + & Q_0
\end{array}$$

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

$$\begin{bmatrix} 1 & a & 0 \\ 0 & 1 & a \\ & & N \end{bmatrix}$$

Then the coefficient vector representing our polynomial

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

is an integer combination of the rows of this matrix.

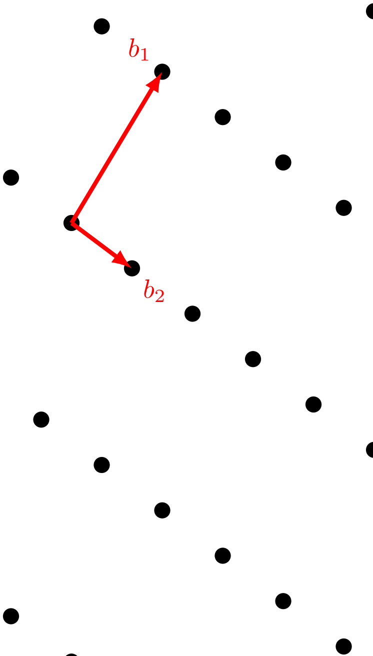
Polynomial coefficient vectors and lattices

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

$$\begin{bmatrix} 1 & a & 0 \\ 0 & 1 & a \\ & & N \end{bmatrix}$$

is a *lattice*.

What is a lattice?



Definition

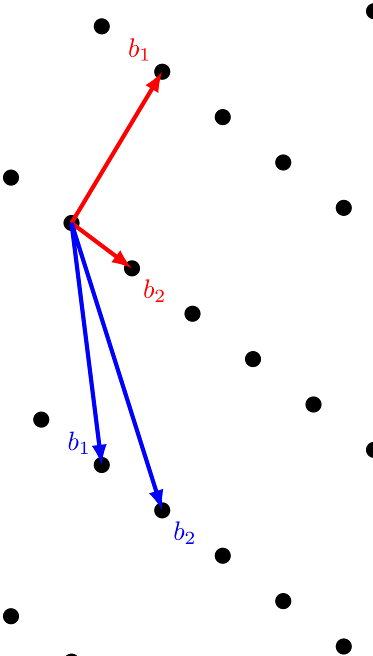
A **lattice** is a discrete additive subgroup of $\mathbb{R}^n$.

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



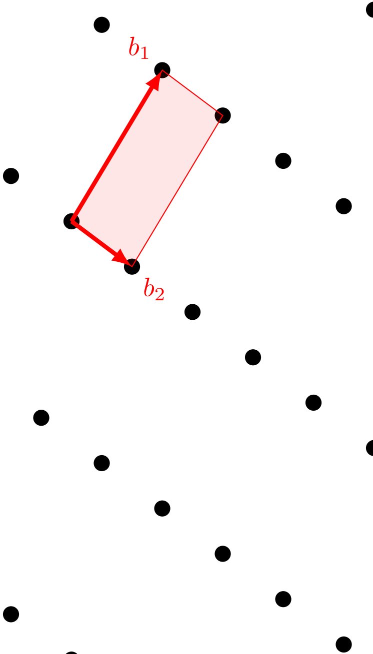
- In n dimensions a lattice has a basis of size at most n.
- The basis is not unique.

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.

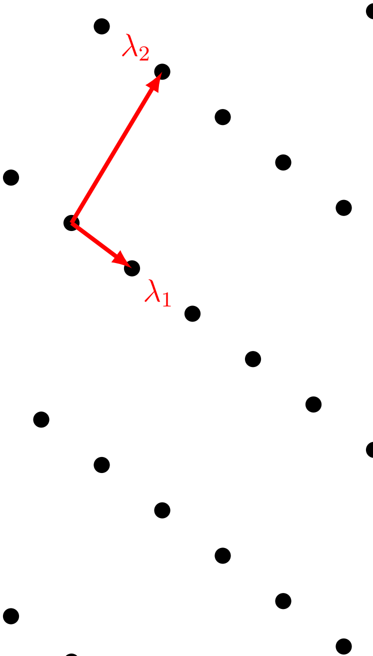


Properties of lattices: Minima

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

Theorem (Minkowski)

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

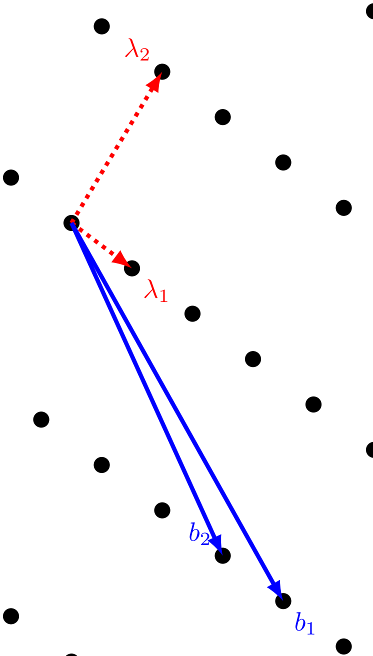


Computational problems on lattices: SVP

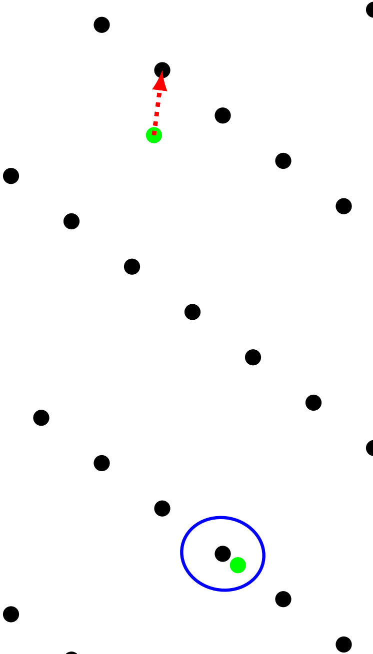
Shortest Vector Problem (SVP)

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

- SVP is NP-hard.



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.

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. $f(r) \equiv 0 \bmod p$ and $p|N$.

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

1. $Q(x) \in \langle f(x), N \rangle$ so $Q(r) \equiv 0 \bmod p$ by construction.
2. If $|r| < R$, then we can bound
$$\begin{aligned} |Q(r)| &= |Q_2 r^2 + Q_1 r + Q_0| \\ &\leq |Q_2| R^2 + |Q_1| R + |Q_0| \end{aligned}$$
3. If $|Q(r)| < N^\beta \leq p$ and $Q(r) \equiv 0 \bmod p$ 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: $f(x) = x + a$, N
Output: $r < R$ such that $f(r) \equiv 0 \bmod p$.

1. Construct lattice basis

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

$$\dim L = 3$$

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

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

Running Coppersmith's method on our example

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

1. Construct lattice basis

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

$\dim L = 3$

$\det L = R^3 N$

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

2. Ignoring approximation factor, we can solve when

$$|Q(r)| \leq |v| \approx \det L^{1/\dim L} < p$$

$$(R^3 N)^{1/3} < N^{1/2}$$

$$R < N^{1/6}$$

In the example we had $\lg r = 86$ and $\lg p = 512$.

Achieving the Howgrave-Graham bound $r < p^{1/2}$

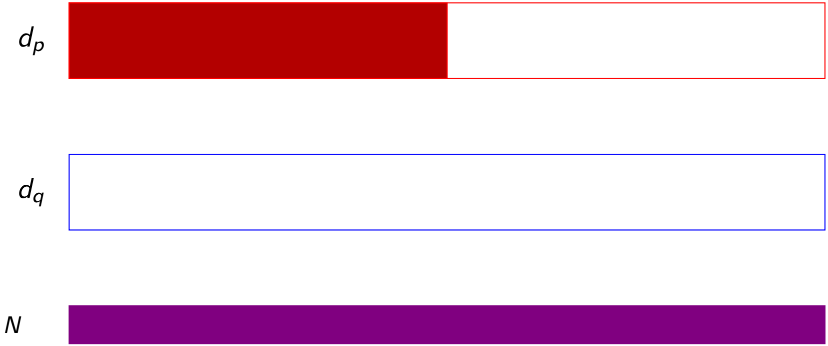
1. Generate lattice from subset of $\langle f(x), N \rangle^k$.
2. Allow higher degree polynomials.

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]

Factoring with Partial Information



Polynomial time. (Lattice basis reduction.) [Blömer May 03]

Key recovery from partial information on CRT-RSA

Assume we know some a such that $d_p = a + r$ and r small.

$$\text{RSA equation:} \qquad ed_p = 1 + k_p(p-1)$$

$$\text{Rearrange:} \qquad (ed_p - 1 + k_p) = k_p p$$

Then we would like to solve for a small solution r to:

$$x + a - e^{-1}(1 + k_p) \equiv 0 \bmod p$$

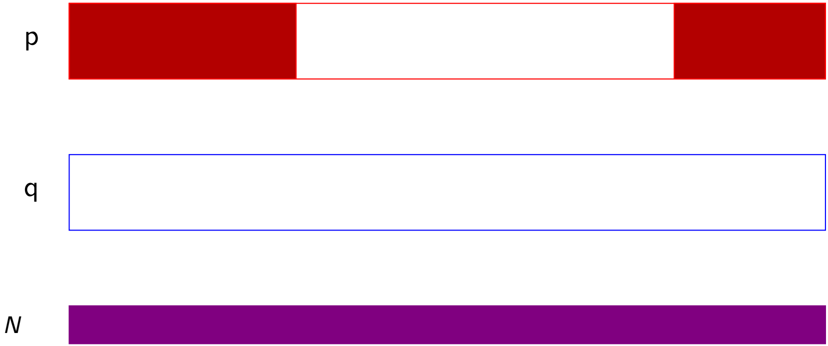
For e small, we can brute force over k_p, and we know $p|N$.

We can apply Coppersmith/Howgrave-Graham technique as before.

Factoring with Partial Information

Unknown.

Factoring with Partial Information



Polynomial time. (Lattice basis reduction.) [Coppersmith 1996] [Howgrave-Graham 2001]

Theorem (Howgrave-Graham 2001)

Let $N = pq$, with $p, q \approx \sqrt{N}$. Given a value a such that

$$a + 2^t r = p \quad \textit{for} \quad r \leq \sqrt{p},$$

we can factor N in polynomial time.

Proof.

1. Input $f(x) = a + 2^t x$.
2. Generate $f'(x) = 2^{-t} f(x)$.
3. Run the Howgrave-Graham algorithm.

$\square$

Factoring with Partial Information

Heuristic polynomial time. (Lattice basis reduction.) [Herrmann May 08]

Theorem (Herrmann May 2008)

Let $N = pq$, with $p, q \approx \sqrt{N}$. Given a value a such that

$$a + 2^{t_1} r_1 + 2^{t_2} r_2 = p \quad \textit{for} \quad r_1 r_2 \leq p^{0.41},$$

we can factor N in polynomial time.

Proof.

1. Input bivariate polynomial $f(x, y) = a + 2^{t_1} x + 2^{t_2} y$.
2. Run bivariate extension of Coppersmith/Howgrave-Graham method.

$\square$

Application: Taiwan Citizen Digital Certificate broken RNG

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

- Taiwanese RSA smartcards had broken RNG that would get “stuck”:

  ```
  c9242492249292499249492449242492
  24929249924949244924249224929249
  92494924492424922492924992494924
  492424922492924992494924492424e5
  ```

Used multivariate Coppersmith/Howgrave-Graham method to factor keys by guessing locations that RNG would “stick” and ”unstick”.

Factoring with Partial Information

Heuristic polynomial time. (Lattice basis reduction.) [Herrmann May 08]

Factoring with Partial Information

Heuristic polynomial in lg N, exponential in number of unknown chunks. (Lattice basis reduction.) [Herrmann May 08]

Theorem (Herrmann May 2008)

Let $N = pq$, with $p, q \approx \sqrt{N}$. Given a value a such that

$$a + 2^{t_1} r_1 + \cdots + 2^{t_m} r_m = p \quad \textit{for} \quad r_1 \ldots r_m \leq p^{0.3},$$

we can factor N in time polynomial in $\lg N$ and exponential in m.

Proof method.

Multivariate extension of Coppersmith/Howgrave-Graham method. $\square$

Factoring with Partial Information

Exponential in number of unknown chunks using lattices.

ECDSA signature scheme

Public Parameters

- An elliptic curve E
- A base point G of order n on E.

Private Key

- An integer d mod n.

Public Key

- $Q = dG$ in uncompressed (x, y) or compressed $(x, 1 \text{ bit of } y)$ format.

Sign

1. Input message hash h.
2. Choose integer k mod n.
3. Compute point $(r, y_r) = kG$.
4. Output $(r, s = k^{-1}(h + dr) \bmod n)$.

Partial key recovery for (EC)DSA:

An attacker learns some information about the signature nonce k.
Can they efficiently recover the secret key d?

ECDSA key recovery from nonce k

k

Sign

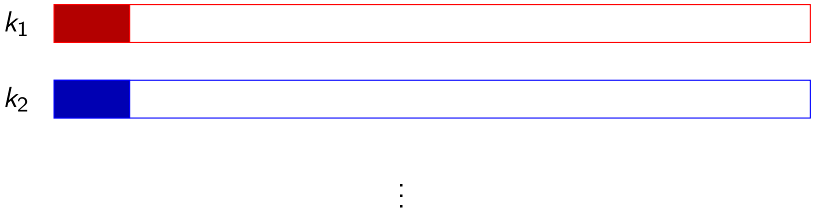
1. Input message hash h.
2. Choose integer k mod n.
3. Compute point $(r, y_r) = kG$.
4. Output $(r, s = k^{-1}(h + dr) \bmod n)$.

Fact

If an attacker learns k for a signature, the long-term secret key d is revealed.

$$d = (sk - h)r^{-1} \bmod n$$

ECDSA key recovery from partial information about nonces



Polynomial time, using lattices. [Howgrave-Graham Smart 2001], [Nguyen Shparlinski 2003]

ECDSA key recovery from partial information about nonces

Secret key d can be computed from MSBs of nonces.

Input signatures $(r_1, s_1), \ldots, (r_m, s_m)$ on messages $h_1, \ldots, h_m$.

Then we have a system of equations in unknowns $k_1, \ldots, k_m, d$:

$$k_1 - s_1^{-1} r_1 d - s_1^{-1} h_1 \equiv 0 \bmod n$$
$$k_2 - s_2^{-1} r_2 d - s_2^{-1} h_2 \equiv 0 \bmod n$$
$$\vdots$$
$$k_m - s_m^{-1} r_m d - s_m^{-1} h_m \equiv 0 \bmod n$$

ECDSA key recovery from partial information about nonces

Secret key d can be computed from MSBs of nonces.

Input signatures $(r_1, s_1), \ldots, (r_m, s_m)$ on messages $h_1, \ldots, h_m$.

Assume we have learned MSBs of k_i so that $k_i = a_i + b_i$ with $b_i < B$.

Then we have a system of equations in unknowns $b_1, \ldots, b_m, d$:

$$\begin{aligned} b_1 - s_1^{-1} r_1 d + a_1 - s_1^{-1} h_1 &\equiv 0 \bmod n \\ b_2 - s_2^{-1} r_2 d + a_2 - s_2^{-1} h_2 &\equiv 0 \bmod n \\ &\vdots \\ b_m - s_m^{-1} r_m d + a_m - s_m^{-1} h_m &\equiv 0 \bmod n \end{aligned}$$

Formulating ECDSA as a hidden number problem

[Howgrave-Graham Smart 2001], [Nguyen Shparlinski 2003]

We have a system of equations in unknowns $b_1, \ldots, b_m, d$:

$$b_1 - t_1 d - u_1 \equiv 0 \bmod n$$
$$b_2 - t_2 d - u_2 \equiv 0 \bmod n$$
$$\vdots$$
$$b_m - t_m d - u_m \equiv 0 \bmod n$$

We assume the b_i are small.

This is an instance of the *hidden number problem* [Boneh Venkatesan 96].

Solving the hidden number problem with lattices

Input:

$$
\begin{aligned}
b_1 - t_1 d - u_1 &\equiv 0 \bmod n \\
&\vdots \\
b_m - t_m d - u_m &\equiv 0 \bmod n
\end{aligned}
$$

in unknowns $b_1, \ldots, b_m, d$, where $|b_i| < B$.

Construct the lattice

$$
M = \begin{bmatrix}
n & & & & & \\
 & n & & & & \\
 & & \ddots & & & \\
 & & & n & & \\
t_1 & t_2 & \ldots & t_m & B/n & \\
u_1 & u_2 & \ldots & u_m & & B
\end{bmatrix}
$$

$v_k = (b_1, b_2, \ldots, b_m, Bd/n, B)$ is a short vector in this lattice.

Solving the hidden number problem with lattices

Construct the lattice

$$M = \begin{bmatrix} n & & & & & \\ & n & & & & \\ & & \ddots & & & \\ & & & n & & \\ t_1 & t_2 & \dots & t_m & B/n & \\ u_1 & u_2 & \dots & u_m & & B \end{bmatrix}$$

Want vector
$v_k = (b_1, b_2, \dots, b_m, Bd/n, B)$

We have:

- $\dim L = m + 2$ $\qquad$ $\det L = B^2 n^{m-1}$
- Ignoring approximation factors, LLL or BKZ will find a vector

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

- We are searching for a vector with length $|v_k| \leq \sqrt{m+2}B$.
- Thus we expect to find v_k when

$$\log B \leq \lfloor \log n(m-1)/m - (\log m)/2 \rfloor$$

Solving the hidden number problem with lattices

We expect to find v_k when

$$\log B \leq \lfloor \log n(m-1)/m - (\log m)/2 \rfloor$$

- 160-bit n: 2 bit leakage $\approx$ 100 signatures [LN 13]
- 256 bit n: 4 bit leakage easy, 3 bits 100+ signatures

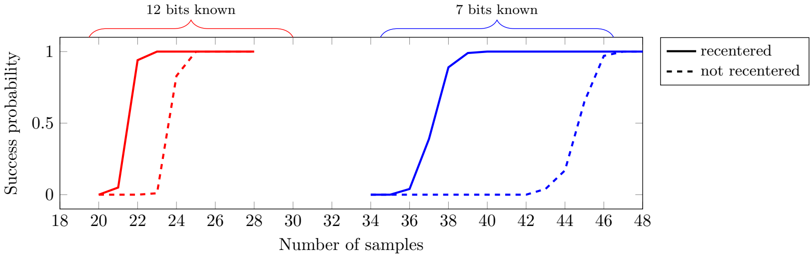
An alternative: Fourier analysis approach (Bleichenbacher)

- 160 bits: 1-bit bias, $\approx 2^{30}$ signatures, [AFGKTZ 14]
- 256 bits: 2-bit bias, $\approx 2^{37}$ signatures, [Tibouchi 18]

Application: Intel SGX EPID cache leak

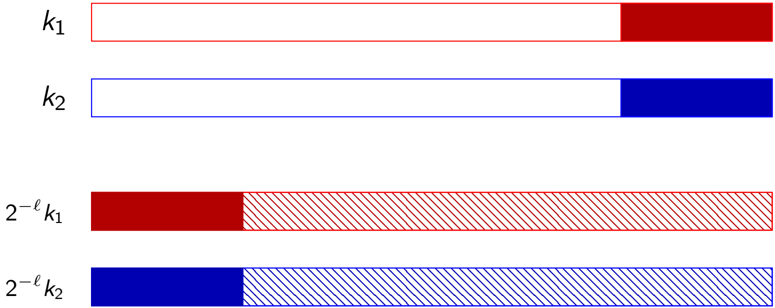
[Dall De Micheli Eisenbarth Genkin Heninger Moghimi Yarom 2018]

Intel SGX EPID attestation protocol leaked nonce MSBs via cache leak.



Can recover secret keys using a few thousand signatures with a lattice attack.

ECDSA key recovery from LSBs of nonces



Solve as before. Bounds are basically the same.

Summary of Key Recovery Techniques

RSA	**Lattice techniques** Large blocks of contiguous bits, no redundancy.	**Branch-and-prune** Non-contiguous bits, redundancy.
DSA	**Lattice techniques** Few samples, several bits known.	**Fourier analysis** Many many samples, fewer bits known.
DH	**Kangaroo** Square root time; hard/annoying.	

Open problem: Is there some way to get the best of all worlds?