RSA X

Coppersmith method

Tanja Lange

Eindhoven University of Technology

2MMC10 – Cryptology

Patterns do not find all factors

These primes

c000000000000000000000000000000
000000000000000000000000000000
000000000000000000000000000000
00000000000000000000000002030b

c000000000000000000000000000000
000000000000000000000000000000
000000000000000000000000000000
00000000000000000000001000000177

were found via GCDs, but not from the patterns.

Patterns do not find all factors

These primes

```
c0000000000000000000000000000000
0000000000000000000000000000000000
0000000000000000000000000000000000
000000000000000000000000002030b
```

```
c0000000000000000000000000000000
0000000000000000000000000000000000
0000000000000000000000000000000000
000000000000000000000001000000177
```

were found via GCDs, but not from the patterns.
Looks like base pattern 0 with some bits flipped.

Coppersmith's method of finding roots mod n

Assume that prime factor p of n has form

$$p = a + r,$$

a is one of the bit patterns, r is a small integer to account for bit errors (and incrementing to next prime).

- Define polynomial
$$f(x) = a + x$$
- Build matrix from coefficients of f.
- Use LLL algorithm (method for lattice basis reduction) on this matrix to construct a new polynomial $g(x)$ where $g(r) = 0$ over the integers.
- Factoring polynomials over $\mathbf{Z}$ is easy. For all roots r_i test if $a + r_i$ divides n.

Coppersmith's method of finding roots mod n

Assume that prime factor p of n has form

$$p = a + r,$$

a is one of the bit patterns, r is a small integer to account for bit errors (and incrementing to next prime).

- Define polynomial
$$f(x) = a + x$$
- Build matrix from coefficients of f.
- Use LLL algorithm (method for lattice basis reduction) on this matrix to construct a new polynomial $g(x)$ where $g(r) = 0$ over the integers.
- Factoring polynomials over $\mathbf{Z}$ is easy. For all roots r_i test if $a + r_i$ divides n.
- This lecture uses this method and LLL as a black box, next looks into when it works.

Find root r of $f(x) = a + x$

- Let $r \leq X$. We know or guess some bound.
  In our case only very few bottom bits changed to reach a prime.
- Construct the matrix M as

$$\begin{bmatrix} X^2 & Xa & 0 \\ 0 & X & a \\ 0 & 0 & n \end{bmatrix}$$

  corresponding to the coefficients of the polynomials
  $Xxf(Xx), f(Xx)$, and n.
- Run LLL lattice basis reduction on matrix M.
- Regard the shortest vector as coefficients of polynomial $g(Xx)$.
- Compute the roots r_i of $g(x)$ and check if $a + r_i$ divides n.
  Note: no X here.

RSA key recovery from partial information

```
p = random_prime(2^512); q = random_prime(2^512)
n = p*q                  # nothing suspicious here
a = p - (p % 2^160)        # partial info we learn
```

RSA key recovery from partial information

```
p = random_prime(2^512); q = random_prime(2^512)
n = p*q                  # nothing suspicious here
a = p - (p % 2^160)        # partial info we learn

sage: hex(a)
'5d388fc0902ebe38dcd214d13e5be1c89827c5ac8e91c8a97
3320ada8edc3365846143427abe6eb51fb3d6a00000000000
0000000000000000000000000000'
```

RSA key recovery from partial information

```
p = random_prime(2^512); q = random_prime(2^512)
n = p*q                  # nothing suspicious here
a = p - (p % 2^160)        # partial info we learn

sage: hex(a)
'5d388fc0902ebe38dcd214d13e5be1c89827c5ac8e91c8a97
3320ada8edc3365846143427abe6eb51fb3d6a00000000000
0000000000000000000000000000'

X = 2^160                # matching p % 2^160 above
M = matrix([[X^2, X*a, 0], [0, X, a], [0, 0, n]])
B = M.LLL()
```

RSA key recovery from partial information

```
p = random_prime(2^512); q = random_prime(2^512)
n = p*q                  # nothing suspicious here
a = p - (p % 2^160)        # partial info we learn

sage: hex(a)
'5d388fc0902ebe38dcd214d13e5be1c89827c5ac8e91c8a97
3320ada8edc3365846143427abe6eb51fb3d6a00000000000
0000000000000000000000000000'

X = 2^160                # matching p % 2^160 above
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: Q.roots(ring=ZZ)
[(28130990442341253511569687156172127007365979813 7, 1)]
sage: a+Q.roots(ring=ZZ)[0][0] == p
True
```

Factors!

- Ran this one all 164 patterns; about 1h/pattern.
- Factored 160 keys, including 39 previously unfactored keys.
- Found all but 2 of the 103 keys factored with the GCD method.

Factors!

- Ran this one all 164 patterns; about 1h/pattern.
- Factored 160 keys, including 39 previously unfactored keys.
- Found all but 2 of the 103 keys factored with the GCD method.
- Missing 2 keys have factor e0000...0f,
  so we included e000 as pattern, but didn't find more factors.

Factors!

- Ran this one all 164 patterns; about 1h/pattern.
- Factored 160 keys, including 39 previously unfactored keys.
- Found all but 2 of the 103 keys factored with the GCD method.
- Missing 2 keys have factor e0000...0f,
  so we included e000 as pattern, but didn't find more factors.
- Coppersmith can handle more errors than $X < p^{1/3}$ by using larger matrices.
  Works up to $X < p^{1/2}$ but gets very expensive.
- See next lecture for math details.
- Generalizations can handle more than one block of errors.
- We found more primes.
  Full story at `http://smartfacts.cr.yp.to/`