Coppersmith attacks

Tanja Lange

Eindhoven University of Technology

11 May 2026

Bad randomness

Problems with non-randomness

- 2004 Bauer–Laurie: checked 18000 PGP RSA keys; found 2 keys sharing a factor.
- 2012 Lenstra–Hughes–Augier–Bos–Kleinjung–Wachter and Heninger–Durumeric–Wustrow–Halderman,
  Factored tens of thousands of public keys on the Internet
  ... typically keys for your home router, not for your bank.
- Different devices produced same key or shared prime factors.
- Most common problem: horrifyingly bad interactions between OpenSSL key generation, `/dev/urandom` seeding, entropy sources.
- 2021 still problems with randomness, see GitKraken bug.

Problems with non-randomness

- 2004 Bauer–Laurie: checked 18000 PGP RSA keys; found 2 keys sharing a factor.
- 2012 Lenstra–Hughes–Augier–Bos–Kleinjung–Wachter and Heninger–Durumeric–Wustrow–Halderman,
  Factored tens of thousands of public keys on the Internet
  ... typically keys for your home router, not for your bank.
- Different devices produced same key or shared prime factors.
- Most common problem: horrifyingly bad interactions between OpenSSL key generation, `/dev/urandom` seeding, entropy sources.
- 2021 still problems with randomness, see GitKraken bug.
- These computations find q_2 in

$$p_1q_1, p_2q_2, p_3q_3, p_4q_2, p_5q_5, p_6q_6;$$

  and thus also p_2 and p_4.
  Obvious:GCD computation. Faster: scaled remainder trees.

Problems with non-randomness

- 2004 Bauer–Laurie: checked 18000 PGP RSA keys; found 2 keys sharing a factor.
- 2012 Lenstra–Hughes–Augier–Bos–Kleinjung–Wachter and Heninger–Durumeric–Wustrow–Halderman,
  Factored tens of thousands of public keys on the Internet
  ... typically keys for your home router, not for your bank.
- Different devices produced same key or shared prime factors.
- Most common problem: horrifyingly bad interactions between OpenSSL key generation, `/dev/urandom` seeding, entropy sources.
- 2021 still problems with randomness, see GitKraken bug.
- These computations find q_2 in

$$p_1q_1, p_2q_2, p_3q_3, p_4q_2, p_5q_5, p_6q_6;$$

  and thus also p_2 and p_4.
  Obvious:GCD computation. Faster: scaled remainder trees.
- Follow-up projects

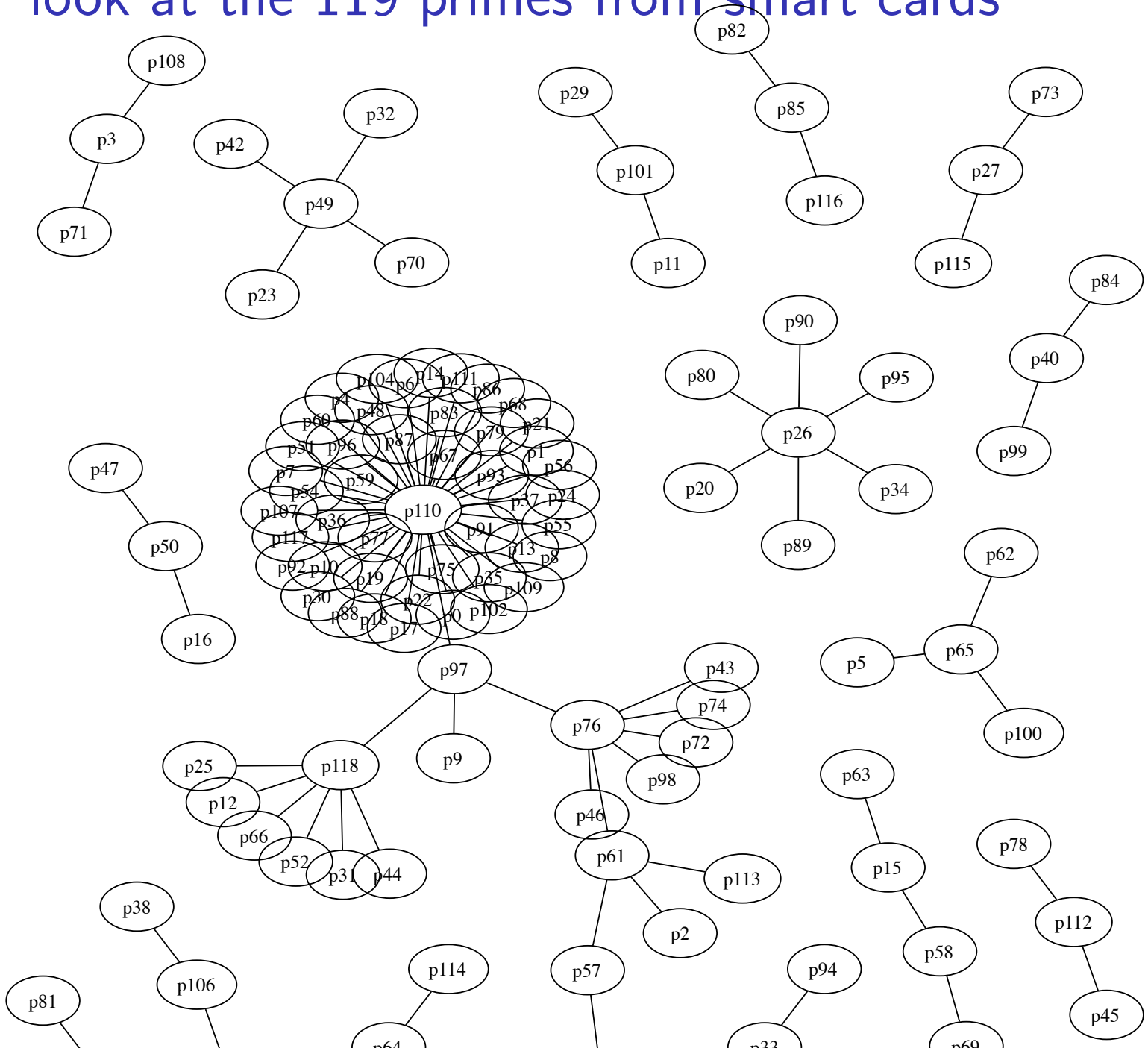
Problems with non-randomness

- 2004 Bauer–Laurie: checked 18000 PGP RSA keys; found 2 keys sharing a factor.
- 2012 Lenstra–Hughes–Augier–Bos–Kleinjung–Wachter and Heninger–Durumeric–Wustrow–Halderman,
  Factored tens of thousands of public keys on the Internet
  ... typically keys for your home router, not for your bank.
- Different devices produced same key or shared prime factors.
- Most common problem: horrifyingly bad interactions between OpenSSL key generation, `/dev/urandom` seeding, entropy sources.
- 2021 still problems with randomness, see GitKraken bug.
- These computations find q_2 in

$$p_1q_1, p_2q_2, p_3q_3, p_4q_2, p_5q_5, p_6q_6;$$

  and thus also p_2 and p_4.
  Obvious:GCD computation. Faster: scaled remainder trees.
- Follow-up projects incl shared primes from certified smart cards.

Closer look at the 119 primes from smart cards



Look at the primes!

Prime factor p110 appears 46 times

Look at the primes!

Prime factor p110 appears 46 times

```
c0000000000000000000000000000000
0000000000000000000000000000000
0000000000000000000000000000000
00000000000000000000000000002f9
```

Look at the primes!

Prime factor p110 appears 46 times

c000000000000000000000000000000
000000000000000000000000000000
000000000000000000000000000000
0000000000000000000000000002f9

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

c92424922492924992494924492424 92
24929249924949244924249224929249
92494924492424922492924992494924
4924249224929249924924492424e5

Several other factors exhibit such a pattern.

How is this pattern generated?

Prime written in binary

1100100100100100001001001001001000100100100100101001001001001001
1001001001001001010010010010010001001001001001000010010010010010
0010010010010010100100100100100110010010010010010100100100100100
0100100100100100001001001001001000100100100100101001001001001001
1001001001001001010010010010010001001001001001000010010010010010
0010010010010010100100100100100110010010010010010100100100100100
0100100100100100001001001001001000100100100100101001001001001001
1001001001001001010010010010010001001001001001000010010011100101

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

0010010010010010110010010010010010010010010010010010010010010010010001001
001001001001001001001001001001001001001001001001001001001001001001001
001001001001001001001001001001001001001001001001001001001001001001001
001001001001001001001001001001001001001001001001001001001001001001001
001001001001001001001001001001001001001001001001001001001001001001001
001001001001001001001001001001001001001001001001001001001001001001001
001001001001001001001001001001001001001001001001001001001001001001001
00100100100100100100100100111001010100100100100100

How is this pattern generated?

Realign

00100100100100101100100100100100100100100100100100100100100100
00100100100100100100100100100100100100100100100100100100100100
00100100100100100100100100100100100100100100100100100100100100
00100100100100100100100100100100100100100100100100100100100100
00100100100100100100100100100100100100100100100100100100100100
00100100100100100100100100100100100100100100100100100100100100
00100100100100100100100100100100100100100100100100100100100100
00100100100100100100100100100100100100100100100100100100100100
00100100100100100100100100110010101001001001001001001001001001

Each of the 119 keys had at least one prime factor
with patterns of period 1,3,5, or 7.

Prime generation?

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

We understand that this is not what they wanted to do.
Best guess: random number generator got stuck with some test pattern,
output was not whitened using AES.

Reply to disclosure was "human error", confirmation that TRNG output was used instead of PRNG output.

TRNG: True random-number generator.
PRNG: Pseudo random-number generator.

Factoring by trial division

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

Do this for any pattern:
0,1,001,010,011,100,101,110
00001,00010,00011,00100,00101,0011,00111,01000,01001,01010,...
00000001,0000011,0000101,0000111,0001001,...

Factoring by trial division

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

Do this for any pattern:
0,1,001,010,011,100,101,110
00001,00010,00011,00100,00101,0011,00111,01000,01001,01010,...
00000001,0000011,0000101,0000111,0001001,...
Computing GCDs factored 105 moduli, of which 18 were new.

Factoring by trial division

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

Do this for any pattern:
0,1,001,010,011,100,101,110
00001,00010,00011,00100,00101,0011,00111,01000,01001,01010,...
00000001,0000011,0000101,0000111,0001001,...
Computing GCDs factored 105 moduli, of which 18 were new.

Breaking RSA-1024 by "trial division".

Factoring by trial division

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

Do this for any pattern:
0,1,001,010,011,100,101,110
00001,00010,00011,00100,00101,0011,00111,01000,01001,01010,...
00000001,0000011,0000101,0000111,0001001,...
Computing GCDs factored 105 moduli, of which 18 were new.

Breaking RSA-1024 by "trial division".
Factored 4 more keys using patterns of length 9.

Sneak preview: We want more keys!

LLL recap

LLL – Lenstra, Lenstra, and Lovász, 1982

- On input a set of vectors $\{v_1, v_2, \ldots, v_d\}$, entered as row vectors in a matrix M, output matrix with shorter vectors v_j' so that $v_j' = \sum a_i v_i$ for some $a_i \in \mathbf{Z}$.

LLL – Lenstra, Lenstra, and Lovász, 1982

- On input a set of vectors $\{v_1, v_2, \ldots, v_d\}$, entered as row vectors in a matrix M, output matrix with shorter vectors v'_j so that $v'_j = \sum a_i v_i$ for some $a_i \in \mathbf{Z}$.
- LLL outputs d vectors which are shorter and more orthogonal. Each vector is an integer linear combination of the inputs.
- LLL uses many elements from Gram-Schmidt orthogonalization:
  - for $j = 1$ to d
  - for $i = 1$ to $j - 1$
  - $\mu_{ij} = \frac{\langle v_i^*, v_j \rangle}{\langle v_i^*, v_i^* \rangle}$
  - $v_j^* = v_j - \sum_{i=1}^{j-1} \mu_{ij} v_i^*$
- Note that the μ_{ij} are not integers, so not permitted as coefficients.
- d vectors are LLL reduced for parameter $0.25 < \delta < 1$ if
  - $|\mu_{ij}| \leq 0.5$ for all $1 \leq j < i \leq d$,
  - $(\delta - \mu_{i-1i}^2)||v_{i-1}^*||^2 \leq ||v_i^*||^2$.
- This guarantees $||v_1|| \leq (2/\sqrt{4\delta - 1})^{(d-1)/2} \det(M)^{1/d}$, where $\det(M)$ is the determinant of the matrix.

LLL algorithm (from Cohen, GTM 138, transposed)

Input: $\{v_1, v_2, \ldots, v_d\}$, $0.25 < \delta < 1$
Output: LLL reduced matrix with parameter δ

1. $k \leftarrow 2$, $k_{\max} \leftarrow 1$, $v_1^* \leftarrow v_1$, $V_1 = \langle v_1, v_1 \rangle$
2. if $k \leq k_{\max}$ go to step 3
   else $k_{\max} \leftarrow k$, $v_k^* \leftarrow v_k$, for $j = 1$ to $k-1$
   - put $\mu_{jk} \leftarrow \langle v_j^*, v_k \rangle / V_j$ and $v_k^* \leftarrow v_k^* - \mu_{jk} v_j^*$

   $V_k = \langle v_k, v_k \rangle$
3. Execute RED$(k, k-1)$. If $(\delta - \mu_{i-1i}^2) V_{k-1} > V_k$ execute SWAP(k) and $k \leftarrow \max\{2, k-1\}$; else for $= k-2$ down to 1 execute RED(k, j) and $k \leftarrow k+1$.
4. If $k \leq d$ go to step 2; else output basis $\{v_1, v_2, \ldots, v_d\}$.

- RED(k, j): If $|\mu_{jk}| \leq 0.5$ return; else $q \leftarrow \lfloor \mu_{jk} \rceil$, $v_k \leftarrow v_k - q v_j$, $\mu_{jk} \leftarrow \mu_{jk} - q$, for $i = 1$ to $j-1$ put $\mu_{ik} \leftarrow \mu_{ik} - q\mu_{ij}$ and return.
- SWAP(k): Swap v_k and v_{k-1}. If $k > 2$ for $j = 1$ to $k-2$ swap μ_{jk} and μ_{jk-1} and update all variables to match (see p.88 in Cohen)

For a nice visualization with animation see pages 61–66 of `http://thijs.com/docs/lec1.pdf`. (This might need Acroread.)

Coppersmith's method for factorization

Patterns do not find all factors

These primes

c0000000000000000000000000000000
0000000000000000000000000000000000
0000000000000000000000000000000000
00000000000000000000000002030b

c0000000000000000000000000000000
0000000000000000000000000000000000
0000000000000000000000000000000000
00000000000000000000001000000177

were found via GCDs, but not from the patterns.

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.
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/`

Theoretical background

Theorem by Howgrave-Graham

Let $g(x) = \sum_{i=0}^{d-1} g_i x^i \in \mathbf{Z}[x]$ of $\deg(g) = d - 1$.
Let $b, k \in \mathbf{Z}_{>0}$. If

1. $g(x_0) \equiv 0 \bmod b^k$ with $|x_0| \leq X$,
2. $||g(xX)|| \leq b^k/\sqrt{d}$

then $g(x_0) = 0$ over $\mathbf{Z}$.

Here $||g(xX)|| = \sqrt{g_0^2 + g_1^2 X^2 + \cdots + g_{d-1}^2 X^{2(d-1)}}$ is Euclidean norm of the coefficient vector of $g(xX)$.

Proof: Let $v = (1, x_0/X, x_0^2/X^2, \ldots, x_0^{d-1}/X^{d-1})$ and $w = (g_0, g_1 X, g_2 X^2, \ldots, g_{d-1} X^{d-1})$.
Note $v \cdot w = g(x_0)$ and each entry in v is ≤ 1.

Theorem by Howgrave-Graham

Let $g(x) = \sum_{i=0}^{d-1} g_i x^i \in \mathbf{Z}[x]$ of $\deg(g) = d - 1$.
Let $b, k \in \mathbf{Z}_{>0}$. If

1. $g(x_0) \equiv 0 \bmod b^k$ with $|x_0| \leq X$,
2. $||g(xX)|| \leq b^k/\sqrt{d}$

then $g(x_0) = 0$ over $\mathbf{Z}$.

Here $||g(xX)|| = \sqrt{g_0^2 + g_1^2 X^2 + \cdots + g_{d-1}^2 X^{2(d-1)}}$ is Euclidean norm of the coefficient vector of $g(xX)$.

Proof: Let $v = (1, x_0/X, x_0^2/X^2, \ldots, x_0^{d-1}/X^{d-1})$ and $w = (g_0, g_1 X, g_2 X^2, \ldots, g_{d-1} X^{d-1})$.
Note $v \cdot w = g(x_0)$ and each entry in v is ≤ 1.

By Cauchy-Schwarz inequality $|v \cdot w| < ||v||||w||$.

Strict inequality as they are not linearly dependent.
Here $||v|| \leq \sqrt{1 + 1 + 1 + \cdots + 1} = \sqrt{d}$ and $||w|| = ||g(xX)||$.

Theorem by Howgrave-Graham

Let $g(x) = \sum_{i=0}^{d-1} g_i x^i \in \mathbf{Z}[x]$ of $\deg(g) = d - 1$.
Let $b, k \in \mathbf{Z}_{>0}$. If

1. $g(x_0) \equiv 0 \bmod b^k$ with $|x_0| \leq X$,
2. $||g(xX)|| \leq b^k/\sqrt{d}$

then $g(x_0) = 0$ over $\mathbf{Z}$.

Here $||g(xX)|| = \sqrt{g_0^2 + g_1^2 X^2 + \cdots + g_{d-1}^2 X^{2(d-1)}}$ is Euclidean norm of the coefficient vector of $g(xX)$.

Proof: Let $v = (1, x_0/X, x_0^2/X^2, \ldots, x_0^{d-1}/X^{d-1})$ and $w = (g_0, g_1 X, g_2 X^2, \ldots, g_{d-1} X^{d-1})$.
Note $v \cdot w = g(x_0)$ and each entry in v is ≤ 1.

By Cauchy-Schwarz inequality $|v \cdot w| < ||v||||w||$.

Strict inequality as they are not linearly dependent.
Here $||v|| \leq \sqrt{1 + 1 + 1 + \cdots + 1} = \sqrt{d}$ and $||w|| = ||g(xX)||$.
Thus $|g(x_0)| = |v \cdot w| < ||v||||w|| \leq \sqrt{d} b^k/\sqrt{d} = b^k$.

Theorem by Howgrave-Graham

Let $g(x) = \sum_{i=0}^{d-1} g_i x^i \in \mathbf{Z}[x]$ of $\deg(g) = d - 1$.
Let $b, k \in \mathbf{Z}_{>0}$. If

1. $g(x_0) \equiv 0 \bmod b^k$ with $|x_0| \leq X$,
2. $||g(xX)|| \leq b^k/\sqrt{d}$

then $g(x_0) = 0$ over $\mathbf{Z}$.

Here $||g(xX)|| = \sqrt{g_0^2 + g_1^2 X^2 + \cdots + g_{d-1}^2 X^{2(d-1)}}$ is Euclidean norm of the coefficient vector of $g(xX)$.

Proof: Let $v = (1, x_0/X, x_0^2/X^2, \ldots, x_0^{d-1}/X^{d-1})$ and $w = (g_0, g_1 X, g_2 X^2, \ldots, g_{d-1} X^{d-1})$.
Note $v \cdot w = g(x_0)$ and each entry in v is ≤ 1.

By Cauchy-Schwarz inequality $|v \cdot w| < ||v||||w||$.

Strict inequality as they are not linearly dependent.

Here $||v|| \leq \sqrt{1 + 1 + 1 + \cdots + 1} = \sqrt{d}$ and $||w|| = ||g(xX)||$.

Thus $|g(x_0)| = |v \cdot w| < ||v||||w|| \leq \sqrt{d} b^k/\sqrt{d} = b^k$.

If $g(x_0) \in b^k \mathbf{Z}$ and $|g(x_0)| < b^k$ then $g(x_0) = 0$.

If $z \equiv 0 \bmod b^k$ & $|z| < b^k$ then

Theorem by Howgrave-Graham

Let $g(x) = \sum_{i=0}^{d-1} g_i x^i \in \mathbf{Z}[x]$ of $\deg(g) = d - 1$.
Let $b, k \in \mathbf{Z}_{>0}$. If

1. $g(x_0) \equiv 0 \bmod b^k$ with $|x_0| \leq X$,
2. $||g(xX)|| \leq b^k/\sqrt{d}$

then $g(x_0) = 0$ over $\mathbf{Z}$.

Here $||g(xX)|| = \sqrt{g_0^2 + g_1^2 X^2 + \cdots + g_{d-1}^2 X^{2(d-1)}}$ is Euclidean norm of the coefficient vector of $g(xX)$.

Proof: Let $v = (1, x_0/X, x_0^2/X^2, \ldots, x_0^{d-1}/X^{d-1})$ and $w = (g_0, g_1 X, g_2 X^2, \ldots, g_{d-1} X^{d-1})$.
Note $v \cdot w = g(x_0)$ and each entry in v is ≤ 1.

By Cauchy-Schwarz inequality $|v \cdot w| < ||v||||w||$.

Strict inequality as they are not linearly dependent.

Here $||v|| \leq \sqrt{1 + 1 + 1 + \cdots + 1} = \sqrt{d}$ and $||w|| = ||g(xX)||$.

Thus $|g(x_0)| = |v \cdot w| < ||v||||w|| \leq \sqrt{d} b^k/\sqrt{d} = b^k$.

If $g(x_0) \in b^k \mathbf{Z}$ and $|g(x_0)| < b^k$ then $g(x_0) = 0$.

If $z \equiv 0 \bmod b^k$ & $|z| < b^k$ then $z = 0$; using $z \equiv 0 \bmod b^k \Leftrightarrow z \in b^k \mathbf{Z}$.

What to look for?

We want to find a polynomial $g(x)$ and a root x_0 so that $g(x_0) \in b^k\mathbf{Z}$.

What to look for?

We want to find a polynomial $g(x)$ and a root x_0 so that $g(x_0) \in b^k\mathbf{Z}$.
Here are some polynomials that work:
$g(x) \in b^k\mathbf{Z}$

What to look for?

We want to find a polynomial $g(x)$ and a root x_0 so that $g(x_0) \in b^k\mathbf{Z}$.
Here are some polynomials that work:
$g(x) \in b^k\mathbf{Z} + b^k x\mathbf{Z}$

What to look for?

We want to find a polynomial $g(x)$ and a root x_0 so that $g(x_0) \in b^k\mathbf{Z}$.
Here are some polynomials that work:
$g(x) \in b^k\mathbf{Z} + b^k x\mathbf{Z} + b^k x^2\mathbf{Z}$

What to look for?

We want to find a polynomial $g(x)$ and a root x_0 so that $g(x_0) \in b^k\mathbf{Z}$.
Here are some polynomials that work:
$g(x) \in b^k\mathbf{Z} + b^k x\mathbf{Z} + b^k x^2\mathbf{Z} + b^k x^3\mathbf{Z} \cdots$.

What to look for?

We want to find a polynomial $g(x)$ and a root x_0 so that $g(x_0) \in b^k\mathbf{Z}$.
Here are some polynomials that work:
$g(x) \in b^k\mathbf{Z} + b^k x\mathbf{Z} + b^k x^2\mathbf{Z} + b^k x^3\mathbf{Z} \cdots$.

We have some polynomial $f(x)$ to start with and know that
$f(x_0) \in b^k\mathbf{Z}$ for the x_0 we're looking for.

What to look for?

We want to find a polynomial $g(x)$ and a root x_0 so that $g(x_0) \in b^k\mathbf{Z}$.
Here are some polynomials that work:
$g(x) \in b^k\mathbf{Z} + b^k x\mathbf{Z} + b^k x^2\mathbf{Z} + b^k x^3\mathbf{Z} \cdots$.

We have some polynomial $f(x)$ to start with and know that
$f(x_0) \in b^k\mathbf{Z}$ for the x_0 we're looking for.

If $\deg(f) = t$ then we're looking for
$g(x) \in b^k\mathbf{Z} + b^k x\mathbf{Z} + b^k x^2\mathbf{Z} + \cdots + b^k x^{t-1}\mathbf{Z} + f(x)\mathbf{Z}$.
The polynomial f does not need an extra b^k.

What to look for?

We want to find a polynomial $g(x)$ and a root x_0 so that $g(x_0) \in b^k\mathbf{Z}$.
Here are some polynomials that work:
$g(x) \in b^k\mathbf{Z} + b^k x\mathbf{Z} + b^k x^2\mathbf{Z} + b^k x^3\mathbf{Z} \cdots$.

We have some polynomial $f(x)$ to start with and know that
$f(x_0) \in b^k\mathbf{Z}$ for the x_0 we're looking for.

If $\deg(f) = t$ then we're looking for
$g(x) \in b^k\mathbf{Z} + b^k x\mathbf{Z} + b^k x^2\mathbf{Z} + \cdots + b^k x^{t-1}\mathbf{Z} + f(x)\mathbf{Z}$.
The polynomial f does not need an extra b^k.

If that's too restrictive we can expand to
$g(x) \in$
$b^k\mathbf{Z} + b^k x\mathbf{Z} + b^k x^2\mathbf{Z} + \cdots + b^k x^{t-1}\mathbf{Z} + f(x)\mathbf{Z} + xf(x)\mathbf{Z} + x^2 f(x)\mathbf{Z} + \cdots$.

What to look for?

We want to find a polynomial $g(x)$ and a root x_0 so that $g(x_0) \in b^k\mathbf{Z}$.
Here are some polynomials that work:
$g(x) \in b^k\mathbf{Z} + b^k x\mathbf{Z} + b^k x^2\mathbf{Z} + b^k x^3\mathbf{Z} \cdots$.

We have some polynomial $f(x)$ to start with and know that
$f(x_0) \in b^k\mathbf{Z}$ for the x_0 we're looking for.

If $\deg(f) = t$ then we're looking for
$g(x) \in b^k\mathbf{Z} + b^k x\mathbf{Z} + b^k x^2\mathbf{Z} + \cdots + b^k x^{t-1}\mathbf{Z} + f(x)\mathbf{Z}$.
The polynomial f does not need an extra b^k.

If that's too restrictive we can expand to
$g(x) \in$
$b^k\mathbf{Z} + b^k x\mathbf{Z} + b^k x^2\mathbf{Z} + \cdots + b^k x^{t-1}\mathbf{Z} + f(x)\mathbf{Z} + xf(x)\mathbf{Z} + x^2 f(x)\mathbf{Z} + \cdots$.

Can also change to congruences modulo b^{2k}:
using $b^{2k}, b^k f$ and f^2 and filling in like above.

Etc.

What to look for and how to find it?

All of these attacks start by finding some polynomial $f(x)$ for which a root modulo b^k is interesting.
Let $\deg(f) = t$ and let $|x_0| \leq X$ for some known X.

To find $g(x) \in$
$b^k\mathbf{Z} + b^k x\mathbf{Z} + b^k x^2\mathbf{Z} + \cdots + b^k x^{t-1}\mathbf{Z} + f(x)\mathbf{Z} + xf(x)\mathbf{Z} + x^2 f(x)\mathbf{Z} + \cdots$
we will use LLL, which builds integer linear combinations of the input rows of a matrix. It returns a vector that is short in the Euclidean norm. (Hence we wanted that in the Howgrave-Graham theorem).

We set up a system of equations in the coefficient vectors, one row per option. $b^k\mathbf{Z}$ turns into coefficient b^k at the x^0 column etc.

For Howgrave-Graham we need to scale the column of x^s by X^s.
So we get

$$\begin{pmatrix} X & a \\ 0 & n \end{pmatrix}$$

$b^k = p$ but we only know n.
But 2×2 likely too small.

What to look for and how to find it?

All of these attacks start by finding some polynomial $f(x)$ for which a root modulo b^k is interesting.
Let $\deg(f) = t$ and let $|x_0| \leq X$ for some known X.

To find $g(x) \in$
$b^k\mathbf{Z} + b^k x\mathbf{Z} + b^k x^2\mathbf{Z} + \cdots + b^k x^{t-1}\mathbf{Z} + f(x)\mathbf{Z} + xf(x)\mathbf{Z} + x^2 f(x)\mathbf{Z} + \cdots$
we will use LLL, which builds integer linear combinations of the input rows of a matrix. It returns a vector that is short in the Euclidean norm. (Hence we wanted that in the Howgrave-Graham theorem).

We set up a system of equations in the coefficient vectors, one row per option. $b^k\mathbf{Z}$ turns into coefficient b^k at the x^0 column etc.

For Howgrave-Graham we need to scale the column of x^s by X^s.
So we get

$$\begin{pmatrix} X^2 & aX & 0 \\ 0 & X & a \\ 0 & 0 & n \end{pmatrix}$$

LLL gives $||v_1|| \leq (2/\sqrt{4\delta - 1})^{(d-1)/2} \det(M)^{1/d}$, i.e.,
$||g(xX)|| \leq 2(X^3 n)^{1/3}$ for $\delta = 1/2$.
Then $2(X^3 n)^{1/3} \leq p/\sqrt{3}$ for $X < n^{1/6}/\sqrt{12}$ if $p \approx q$.

How to handle larger X?

Use dimension $d = 4$:

$$\begin{pmatrix} X^3 & aX^2 & 0 & 0 \\ 0 & X^2 & aX & 0 \\ 0 & 0 & X & a \\ 0 & 0 & 0 & n \end{pmatrix}$$

LLL gives $||v_1|| \leq (2/\sqrt{4\delta - 1})^{(d-1)/2} \det(M)^{1/d}$, i.e.,
$||g(xX)|| \leq 2^{3/2}(X^6 n)^{1/4}$ for $\delta = 1/2$.

Then $2^{3/2}(X^6 n)^{1/4} \leq p/\sqrt{4}$ for $X < n^{1/6}$ if $p \approx q$ (ignoring constants).

How to handle larger X?

Use dimension $d = 4$:

$$\begin{pmatrix} X^3 & aX^2 & 0 & 0 \\ 0 & X^2 & aX & 0 \\ 0 & 0 & X & a \\ 0 & 0 & 0 & n \end{pmatrix}$$

LLL gives $||v_1|| \leq (2/\sqrt{4\delta - 1})^{(d-1)/2} \det(M)^{1/d}$, i.e.,
$||g(xX)|| \leq 2^{3/2}(X^6 n)^{1/4}$ for $\delta = 1/2$.

Then $2^{3/2}(X^6 n)^{1/4} \leq p/\sqrt{4}$ for $X < n^{1/6}$ if $p \approx q$ (ignoring constants).

Same as before. Larger d does not help.

How to handle larger X?

Use dimension $d = 5$ **and** equations modulo p^2,
so $b^k = p^2$.
Again, we only know n not p.
Use $n^2, nf(x)$ and $f^2(x)$ and shifts.

$$\begin{pmatrix} X^4 & 2aX^3 & a^2X^2 & 0 & 0 \\ 0 & X^3 & 2aX^2 & a^2X & 0 \\ 0 & 0 & X^2 & 2aX & a^2 \\ 0 & 0 & 0 & nX & na \\ 0 & 0 & 0 & 0 & n^2 \end{pmatrix}$$

LLL gives $||v_1|| \leq (2/\sqrt{4\delta - 1})^{(d-1)/2} \det(M)^{1/d}$, i.e.,
$||g(xX)|| \leq 2^2(X^{10}n^3)^{1/5}$ for $\delta = 1/2$.

Then $2^2(X^{10}n^3)^{1/5} \leq p^2/\sqrt{5}$ for $X < n^{1/5}$ if $p \approx q$ (ignoring constants).

How to handle larger X?

Use dimension $d = 5$ **and** equations modulo p^2,
so $b^k = p^2$.
Again, we only know n not p.
Use $n^2, nf(x)$ and $f^2(x)$ and shifts.

$$\begin{pmatrix} X^4 & 2aX^3 & a^2X^2 & 0 & 0 \\ 0 & X^3 & 2aX^2 & a^2X & 0 \\ 0 & 0 & X^2 & 2aX & a^2 \\ 0 & 0 & 0 & nX & na \\ 0 & 0 & 0 & 0 & n^2 \end{pmatrix}$$

LLL gives $||v_1|| \leq (2/\sqrt{4\delta - 1})^{(d-1)/2} \det(M)^{1/d}$, i.e.,
$||g(xX)|| \leq 2^2(X^{10}n^3)^{1/5}$ for $\delta = 1/2$.

Then $2^2(X^{10}n^3)^{1/5} \leq p^2/\sqrt{5}$ for $X < n^{1/5}$ if $p \approx q$ (ignoring constants).
This is a better bound $(1/5 > 1/6)$.

How to handle larger X?

Use dimension $d = 5$ **and** equations modulo p^2,
so $b^k = p^2$.
Again, we only know n not p.
Use $n^2, nf(x)$ and $f^2(x)$ and shifts.

$$\begin{pmatrix} X^4 & 2aX^3 & a^2X^2 & 0 & 0 \\ 0 & X^3 & 2aX^2 & a^2X & 0 \\ 0 & 0 & X^2 & 2aX & a^2 \\ 0 & 0 & 0 & nX & na \\ 0 & 0 & 0 & 0 & n^2 \end{pmatrix}$$

LLL gives $||v_1|| \leq (2/\sqrt{4\delta - 1})^{(d-1)/2} \det(M)^{1/d}$, i.e.,
$||g(xX)|| \leq 2^2(X^{10}n^3)^{1/5}$ for $\delta = 1/2$.

Then $2^2(X^{10}n^3)^{1/5} \leq p^2/\sqrt{5}$ for $X < n^{1/5}$ if $p \approx q$ (ignoring constants).
This is a better bound $(1/5 > 1/6)$.

Next use n^3, n^2f, nf^2, and f^3 modulo p^3 and expand to $d = 7$ with X^2f^3 and X^3f^3 for $X < n^{3/14}$.

How to handle larger X?

Use dimension $d = 5$ **and** equations modulo p^2,
so $b^k = p^2$.
Again, we only know n not p.
Use $n^2, nf(x)$ and $f^2(x)$ and shifts.

$$\begin{pmatrix} X^4 & 2aX^3 & a^2X^2 & 0 & 0 \\ 0 & X^3 & 2aX^2 & a^2X & 0 \\ 0 & 0 & X^2 & 2aX & a^2 \\ 0 & 0 & 0 & nX & na \\ 0 & 0 & 0 & 0 & n^2 \end{pmatrix}$$

LLL gives $||v_1|| \leq (2/\sqrt{4\delta - 1})^{(d-1)/2} \det(M)^{1/d}$, i.e.,
$||g(xX)|| \leq 2^2(X^{10}n^3)^{1/5}$ for $\delta = 1/2$.

Then $2^2(X^{10}n^3)^{1/5} \leq p^2/\sqrt{5}$ for $X < n^{1/5}$ if $p \approx q$ (ignoring constants).
This is a better bound $(1/5 > 1/6)$.

Next use n^3, n^2f, nf^2, and f^3 modulo p^3 and expand to $d = 7$ with X^2f^3 and X^3f^3 for $X < n^{3/14}$.

Asymptotically this reaches $X < n^{1/4}$.

More interesting bad randomness

ROCA – Return of Coppersmith Attack

M. Nemec, M. Sys, P. Svenda, D. Klinec, V. Matyas

Interesting RSA key generation in Infineon chip (2017).

Normally one picks p' randomly, tests if prime

ROCA – Return of Coppersmith Attack

M. Nemec, M. Sys, P. Svenda, D. Klinec, V. Matyas

Interesting RSA key generation in Infineon chip (2017).

Normally one picks odd p' randomly, tests if prime

ROCA – Return of Coppersmith Attack

M. Nemec, M. Sys, P. Svenda, D. Klinec, V. Matyas

Interesting RSA key generation in Infineon chip (2017).

Normally one picks odd p', not divisible by 3, randomly, tests if prime

Warm-up for ROCA I

Ensure p' not divisible by any of 3,5,7,11, and 13.
Use that $\mathbf{F}_p^*$ is cyclic.
Choose p' satisfying the congruences
$p' \equiv 2^{r_1} \bmod 3$
$p' \equiv 3^{r_2} \bmod 5$
$p' \equiv 3^{r_3} \bmod 7$
$p' \equiv 2^{r_4} \bmod 11$
$p' \equiv 2^{r_5} \bmod 13$
with r_i random and p' reconstructed using CRT.

Note: 2 and 3 are generators for the respective primes.
This gives
$2 \cdot 4 \cdot 6 \cdot 10 \cdot 12 = 5760$ options.

Warm-up for ROCA II

It would have been OKi'ish but worse to choose p' as
$p' \equiv 2^{r_1} \bmod 3$
$p' \equiv 2^{r_2} \bmod 5$
$p' \equiv 2^{r_3} \bmod 7$
$p' \equiv 2^{r_4} \bmod 11$
$p' \equiv 2^{r_5} \bmod 13$
with r_i random and p' reconstructed using CRT.

Note: 2 is not a generator.mod 7
This gives only
$2 \cdot 4 \cdot 3 \cdot 10 \cdot 12 = 2880$ options.

Warm-up for ROCA III

It is really bad
to replace this by a single exponentiation and choose p' as
$p' \equiv 5477^r \bmod 3 \cdot 5 \cdot 7 \cdot 11 \cdot 13$
with r random.

Note:
The orders of 5477
modulo 3,5,7,11, and 13
are 2,4,6,2, and 6, but the powers are linked.

Instead of $2 \cdot 4 \cdot 6 \cdot 2 \cdot 6 = 576$ this gives $\mathrm{lcm}(2, 4, 6, 2, 6) = 12$ options.

Actual ROCA I

All RSA keys generated by those Infineon smart cards satisfy
$n \bmod 2 = 1$
$n \bmod 11 \in \{1, 10\}$
$n \bmod 37 \in \{1, 10, 37\}$
$n \bmod 97 \in \{1, 35, 36, 61, 62, 96\}$
$n \bmod 331 \in \{1, 330\}$

These give $1 \cdot 2 \cdot 3 \cdot 6 \cdot 2 = 72$
possibilities of $n \bmod L$, where $L = 2 \cdot 11 \cdot 37 \cdot 97 \cdot 331$, instead of
$1 \cdot 10 \cdot 36 \cdot 96 \cdot 330 = 11404800$

Actual ROCA I

All RSA keys generated by those Infineon smart cards satisfy
$n \bmod 2 = 1$
$n \bmod 11 \in \{1, 10\}$
$n \bmod 37 \in \{1, 10, 37\}$
$n \bmod 97 \in \{1, 35, 36, 61, 62, 96\}$
$n \bmod 331 \in \{1, 330\}$

These give $1 \cdot 2 \cdot 3 \cdot 6 \cdot 2 = 72$
possibilities of $n \bmod L$, where $L = 2 \cdot 11 \cdot 37 \cdot 97 \cdot 331$, instead of
$1 \cdot 10 \cdot 36 \cdot 96 \cdot 330 = 11404800$
Worse,
$n \bmod 2 \cdot 11 \cdot 37 \cdot 97 \cdot 331$
$\in \{1, 65537, 4878941, 8942297, 14367385, 24016035\}$

$n \in \{65537^i \bmod L | i \in \mathbf{Z}\}$ and 65537 has order 6 mod L.

If $n = p \cdot q = 65537^i \bmod L$
then likely $p, q \in \{65537^i \bmod L | i \in \mathbf{Z}\}$.

Actual ROCA II

There are more congruences where this holds.

Actually $L = \prod_{\ell<702,\ell\text{prime}} \ell$ with $\log_2 L \approx 971$.

Chips used for RSA-2048, thus $\log_2 p = 1024$,
so $p = p' + k \cdot L$ where $p \equiv p' \bmod L$, and k with
$\gcd(k, L) = 1$ and $\log_2 k \approx 53$ is random so that p is prime.

Same for q.

Actual ROCA II

There are more congruences where this holds.

Actually $L = \prod_{\ell<702,\ell\text{prime}} \ell$ with $\log_2 L \approx 971$.

Chips used for RSA-2048, thus $\log_2 p = 1024$,
so $p = p' + k \cdot L$ where $p \equiv p' \bmod L$, and k with
$\gcd(k, L) = 1$ and $\log_2 k \approx 53$ is random so that p is prime.

Same for q.

If L was a power of 2, this would be Coppersmith with known bottom bits.

Actual ROCA II

There are more congruences where this holds.

Actually $L = \prod_{\ell<702,\ell\text{prime}} \ell$ with $\log_2 L \approx 971$.

Chips used for RSA-2048, thus $\log_2 p = 1024$,
so $p = p' + k \cdot L$ where $p \equiv p' \bmod L$, and k with
$\gcd(k, L) = 1$ and $\log_2 k \approx 53$ is random so that p is prime.

Same for q.

If L was a power of 2, this would be Coppersmith with known bottom bits.

Exercise:
Show how to transpose the above for known bottom bits.

Actual ROCA II

There are more congruences where this holds.

Actually $L = \prod_{\ell<702,\ell\text{prime}} \ell$ with $\log_2 L \approx 971$.

Chips used for RSA-2048, thus $\log_2 p = 1024$,
so $p = p' + k \cdot L$ where $p \equiv p' \bmod L$, and k with
$\gcd(k, L) = 1$ and $\log_2 k \approx 53$ is random so that p is prime.

Same for q.

If L was a power of 2, this would be Coppersmith with known bottom bits.

Exercise:
Show how to transpose the above for known bottom bits.
Hint: use that 2^i is invertible modulo n to remove huge constant in determinant.

Full attack

Lenstra's "Divisors in Residue Classes" finds prime factors of the form $p = u + k \cdot L$ efficiently if $L \geq n^{1/3}$.
Coppersmith, Howgrave-Graham, and Nagaraj work for $L \geq n^{1/4}$.

Full attack

Lenstra's "Divisors in Residue Classes" finds prime factors of the form $p = u + k \cdot L$ efficiently if $L \geq n^{1/3}$.
Coppersmith, Howgrave-Graham, and Nagaraj work for $L \geq n^{1/4}$.

Here $\log_2 L > 970 > 683 > 2048/3$, hence this works.

Full attack

Lenstra's "Divisors in Residue Classes" finds prime factors of the form $p = u + k \cdot L$ efficiently if $L \geq n^{1/3}$.
Coppersmith, Howgrave-Graham, and Nagaraj work for $L \geq n^{1/4}$.

Here $\log_2 L > 970 > 683 > 2048/3$, hence this works.

We can do better:
Run Lenstra for all $p' \in \{65537^i \bmod L | i \in \mathbf{Z}\}$.

Each run is cheap, but there are many options for p', e.g.
$65537^i \bmod 23 \in \{\pm 1, \pm 2, \pm 3, \pm 4, \ldots, \pm 9, \pm 10, \pm 11\}$.

Full attack

Lenstra's "Divisors in Residue Classes" finds prime factors of the form $p = u + k \cdot L$ efficiently if $L \geq n^{1/3}$.
Coppersmith, Howgrave-Graham, and Nagaraj work for $L \geq n^{1/4}$.

Here $\log_2 L > 970 > 683 > 2048/3$, hence this works.

We can do better:
Run Lenstra for all $p' \in \{65537^i \bmod L | i \in \mathbf{Z}\}$.

Each run is cheap, but there are many options for p', e.g.
$65537^i \bmod 23 \in \{\pm 1, \pm 2, \pm 3, \pm 4, \ldots, \pm 9, \pm 10, \pm 11\}$.

But L is much larger than needed.
So use $L' | L$ which minimizes number of choices $\times$ runtime.

Coppersmith's method for stereotyped messages

Stereotyped message with small e in RSA

```
n = random_prime(2^160)*random_prime(2^160)
m = Integer('myfavoritesubjectiscryptology',36)
c = m^3 % n      # note small primes, reduction happens
```

Stereotyped message with small e in RSA

```
n = random_prime(2^160)*random_prime(2^160)
m = Integer('myfavoritesubjectiscryptology',36)
c = m^3 % n       # note small primes, reduction happens
```

Match this up with
$g(x) \in b^k\mathbf{Z} + b^k x\mathbf{Z} + b^k x^2\mathbf{Z} + \cdots + b^k x^{t-1}\mathbf{Z} + f(x)\mathbf{Z}$.
for $b^k = n, f(x) = (a+x)^3 - c$ with

```
a = Integer('myfavoritesubjectis0000000000',36)
```

Stereotyped message with small e in RSA

```
n = random_prime(2^160)*random_prime(2^160)
m = Integer('myfavoritesubjectiscryptology',36)
c = m^3 % n       # note small primes, reduction happens
```

Match this up with
$g(x) \in b^k\mathbf{Z} + b^k x\mathbf{Z} + b^k x^2\mathbf{Z} + \cdots + b^k x^{t-1}\mathbf{Z} + f(x)\mathbf{Z}$.
for $b^k = n, f(x) = (a+x)^3 - c$ with

```
a = Integer('myfavoritesubjectis0000000000',36)

X = Integer('zzzzzzzzzz',36)
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]])
```

Stereotyped message with small e in RSA

```
n = random_prime(2^160)*random_prime(2^160)
m = Integer('myfavoritesubjectiscryptology',36)
c = m^3 % n       # note small primes, reduction happens
```

Match this up with
$g(x) \in b^k\mathbf{Z} + b^k x\mathbf{Z} + b^k x^2\mathbf{Z} + \cdots + b^k x^{t-1}\mathbf{Z} + f(x)\mathbf{Z}$.
for $b^k = n, f(x) = (a+x)^3 - c$ with

```
a = Integer('myfavoritesubjectis0000000000',36)

X = Integer('zzzzzzzzzz',36)
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]
```

Stereotyped message with small e in RSA

```
n = random_prime(2^160)*random_prime(2^160)
m = Integer('myfavoritesubjectiscryptology',36)
c = m^3 % n       # note small primes, reduction happens
```

Match this up with
$g(x) \in b^k\mathbf{Z} + b^k x\mathbf{Z} + b^k x^2\mathbf{Z} + \cdots + b^k x^{t-1}\mathbf{Z} + f(x)\mathbf{Z}$.
for $b^k = n, f(x) = (a+x)^3 - c$ with

```
a = Integer('myfavoritesubjectis0000000000',36)

X = Integer('zzzzzzzzzz',36)
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=36)
'cryptology'
```

When does this work?

For $e = 3$ we have M as

$$\begin{pmatrix} X^3 & 3aX^2 & 3a^2X & a^3 - c \\ 0 & nX^2 & 0 & 0 \\ 0 & 0 & nX & 0 \\ 0 & 0 & 0 & n \end{pmatrix}$$

Thus $\det(M) = X^6 n^3$.

When does this work?

For $e = 3$ we have M as

$$\begin{pmatrix} X^3 & 3aX^2 & 3a^2X & a^3 - c \\ 0 & nX^2 & 0 & 0 \\ 0 & 0 & nX & 0 \\ 0 & 0 & 0 & n \end{pmatrix}$$

Thus $\det(M) = X^6 n^3$.

Howgrave-Graham guarantees solution if

$$(X^6 n^3)^{1/4} < n,$$

i.e. if

$$X < n^{1/6}.$$

When does this work?

For $e = 3$ we have M as

$$\begin{pmatrix} X^3 & 3aX^2 & 3a^2X & a^3 - c \\ 0 & nX^2 & 0 & 0 \\ 0 & 0 & nX & 0 \\ 0 & 0 & 0 & n \end{pmatrix}$$

Thus $\det(M) = X^6 n^3$.

Howgrave-Graham guarantees solution if

$$(X^6 n^3)^{1/4} < n,$$

i.e. if

$$X < n^{1/6}.$$

Exercise:
Work this out for $e = 5$, i.e. $f(x) = (x + a)^5 - c$.