CSE 291-E: Applied Cryptography

Nadia Heninger

UCSD

Fall 2020 Lecture 18

Legal Notice

The Zoom session for this class will be recorded and made available asynchronously on Canvas to registered students.

Announcements

1. HW 8 is due December 10!

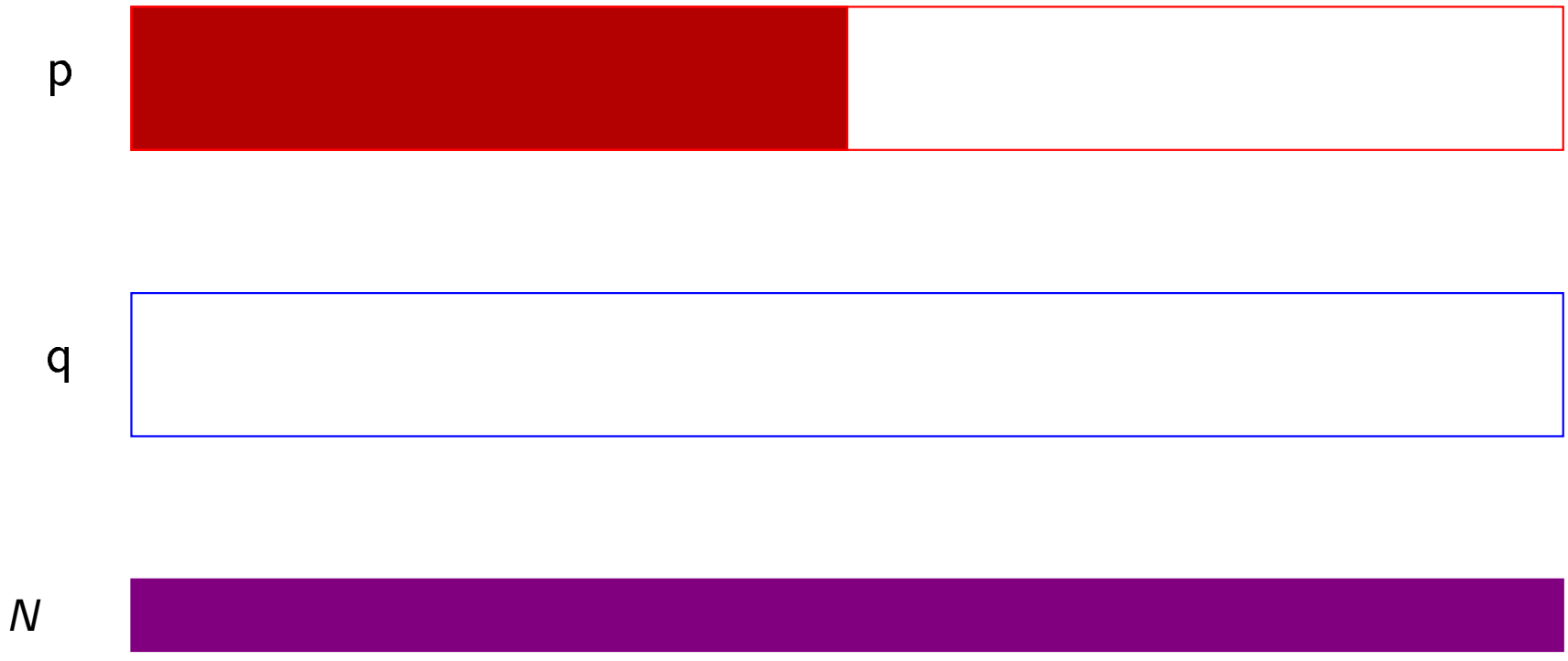
Last time:

- Lattice-based cryptanalysis

This time:

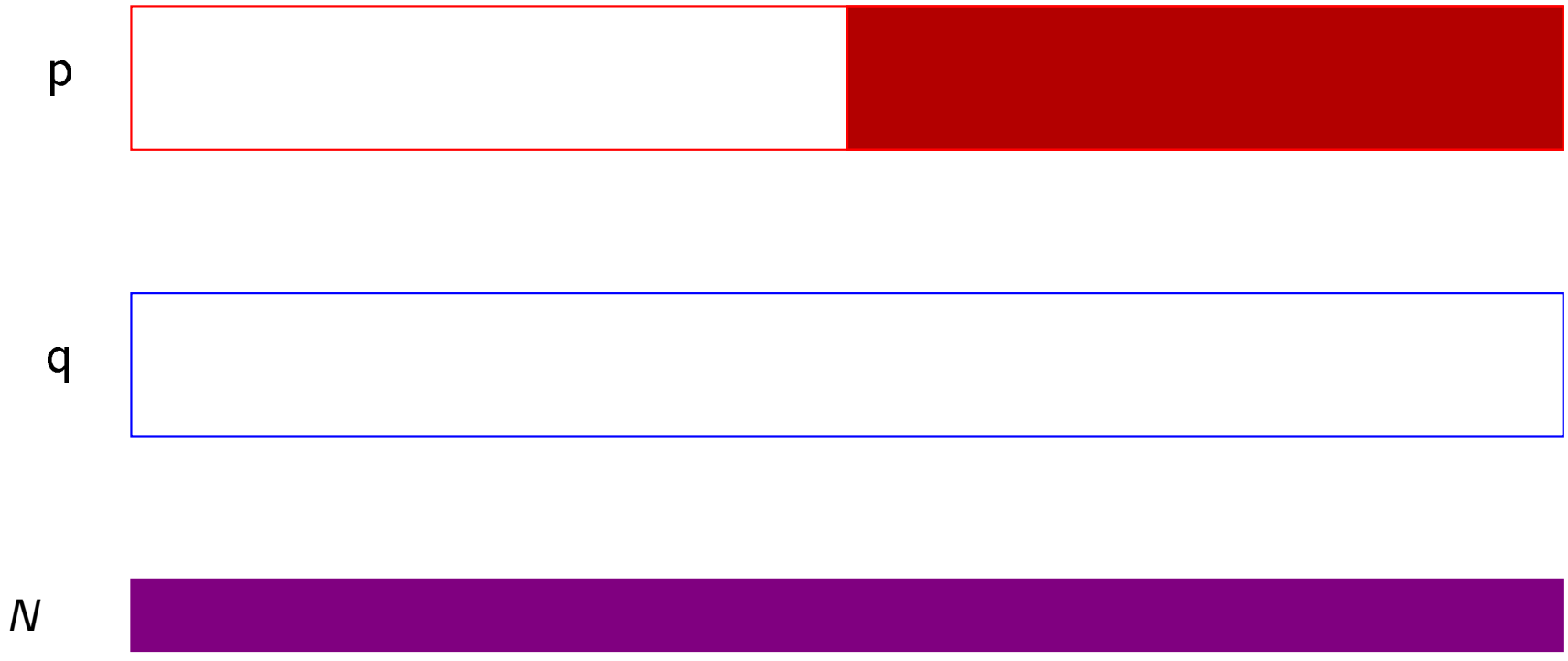
- More lattice-based cryptanalysis

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
4330240bc00000000000000000000'
```

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

Assumption: Attacker does not start knowing factorization of N.

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 $|r| < N^{\beta^2/d}$ in polynomial time in $\lg N$ and d.

For RSA partial key recovery, we have

$$f(x) = a + x \quad d = 1 \quad \beta = 1/2$$

and we want to find a solution vanishing modulo $p \approx N^{1/2}$ for some $p \mid N$. Theorem: can find $|r| < N^{1/4}$.

Partial key recovery example

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

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

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 $x(x + a), (x + a), N$.
2. This corresponds to a lattice basis

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

$\dim L = 3$

$\det L = R^3 N$

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 $x(x+a), (x+a), N$.
2. This corresponds to a lattice basis

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

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

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 $x(x + a), (x + a), N$.
2. This corresponds to a lattice basis

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

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 $\lg p = 512$.

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

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

  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
0000000000000000000000000000000000
0000000000000000000000000000000000
00000000000000000000000000002f9
```

Investigating the factors...

Most common factor appears 46 times

```
c000000000000000000000000000000
000000000000000000000000000000
000000000000000000000000000000
0000000000000000000000000002f9
```

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

Investigating the factors...

Most common factor 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
249292499249249244924249224929249
924949244924249224929249924949
24924249224929249924924924e5
```

Several other factors exhibit such a pattern.

How is this pattern generated?

110010010010010000100100100100100010010010010010100100100100100
100100100100100101001001001001000100100100100100000100100100100
001001001001001010010010010010011001001001001001010010010010010
010010010010010000100100100100100010010010010010100100100100100
100100100100100101001001001001000100100100100100000100100100100
001001001001001010010010010010011001001001001001010010010010010
010010010010010000100100100100100010010010010010100100100100100
100100100100100101001001001001000100100100100100000100100111001 01

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

00100100100100101100100100100100100100100100100100100100100100
00100100100100100100100100100100100100100100100100100100100100
00100100100100100100100100100100100100100100100100100100100100
00100100100100100100100100100100100100100100100100100100100100
00100100100100100100100100100100100100100100100100100100100100
00100100100100100100100100100100100100100100100100100100100100
00100100100100100100100100100100100100100100100100100100100100
00100100100100100100100100100100100100100100100100100100100100
00100100100100100100100100110010101001001001001001001001001001

How is this pattern generated?

Realign

00100100100100101100100100100100100100100100100100100100100100
00100100100100100100100100100100100100100100100100100100100100
00100100100100100100100100100100100100100100100100100100100100
00100100100100100100100100100100100100100100100100100100100100
00100100100100100100100100100100100100100100100100100100100100
00100100100100100100100100100100100100100100100100100100100100
00100100100100100100100100100100100100100100100100100100100100
00100100100100100100100100100100100100100100100100100100100100
00100100100100100100100100110010101001001001001001001001001001

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
0000000000000000000000000000000000
0000000000000000000000000000000000
00000000000000000000000000101ff

c0000000000000000000000000000000
0000000000000000000000000000000000
0000000000000000000000000000000000
00000000000000000000000100000177
```

Some more prime factors

```
c0000000000000000000000000000000
0000000000000000000000000000000000
0000000000000000000000000000000000
0000000000000000000000000101ff

c0000000000000000000000000000000
0000000000000000000000000000000000
0000000000000000000000000000000000
0000000000000000000000100000177
```

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.

ROCA: Return of Coppersmith's Attack

[Nemec Sys Svenda Klinec Matyas 2017]

Infineon smartcards always generated RSA primes that had the form

$$p = kM + (65537^a \bmod M)$$

for $M = 2 \cdot 3 \cdot 5 \ldots 167$ (first n primes)

ROCA attack: Guess a, and use Coppersmith/Howgrave-Graham lattice attack to recover k and thus p.

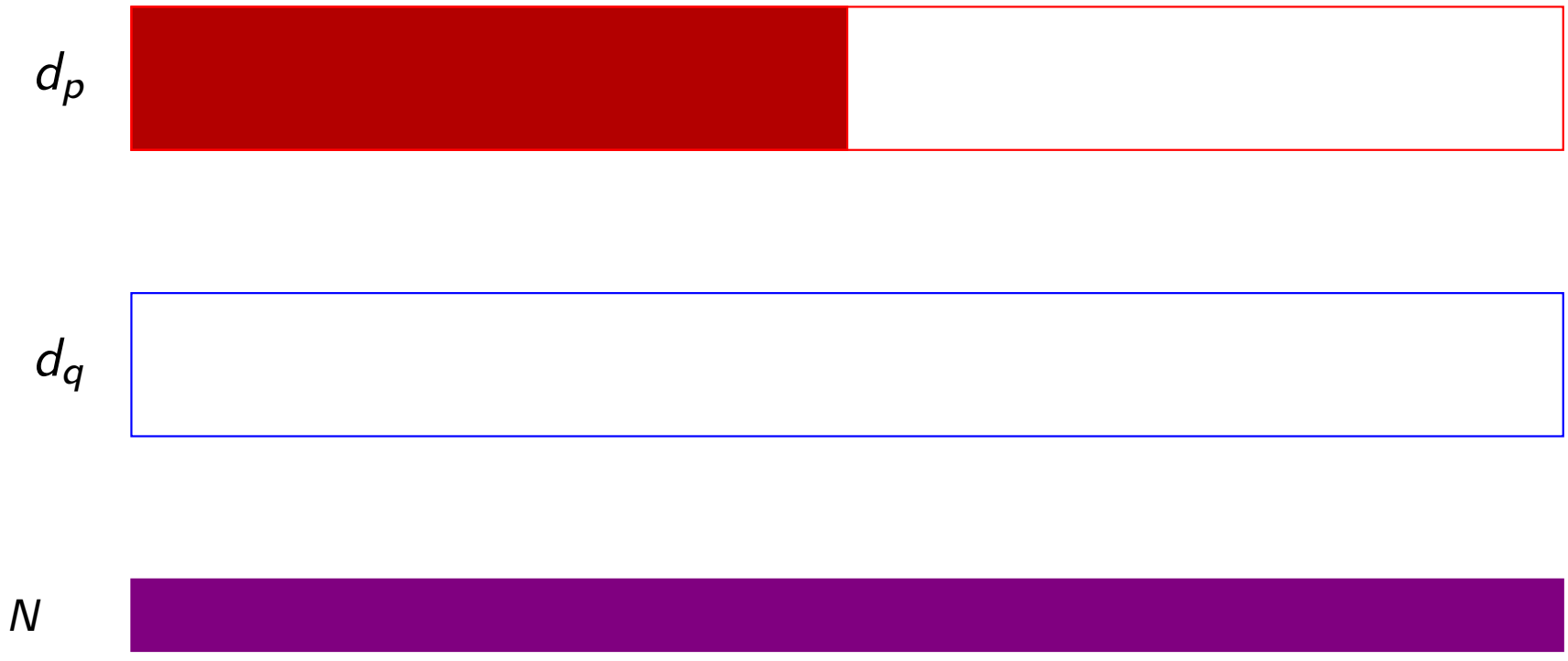
Affected all Infineon cards

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]

RSA key recovery from CRT coefficients



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.

```
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

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

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

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

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.

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: You just did this.

k 

Sign

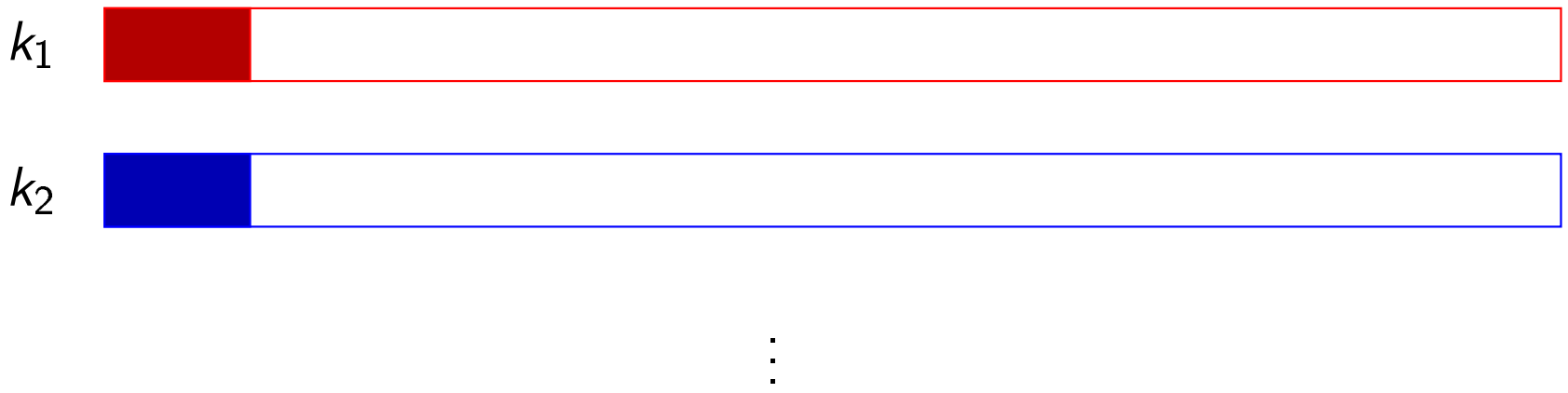
1. Input message hash h.
2. Choose integer k mod n.
3. Compute point $(r, y_r) = kG$.
4. Output $(r, s = k^{-1}(h + dr) \bmod n)$.

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 $k_1, \ldots, k_m, d$:

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

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 $b_1, \ldots, b_m, d$:

$$\begin{aligned} b_1 - t_1 d - u_1 &\equiv 0 \bmod n \\ b_2 - t_2 d - u_2 &\equiv 0 \bmod n \\ &\vdots \\ b_m - t_m d - u_m &\equiv 0 \bmod n \end{aligned}$$

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:

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

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:

- $\dim L = m + 2$ $\qquad\qquad$ $\det L = B^2 n^{m-1}$
- 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: Learn key from 2 bits of information per signature by reducing 100-dimensional lattice [LN 13]
- 256 bit n: Learn key from 3 bits of information per signature by reducing 100-dimensional lattice

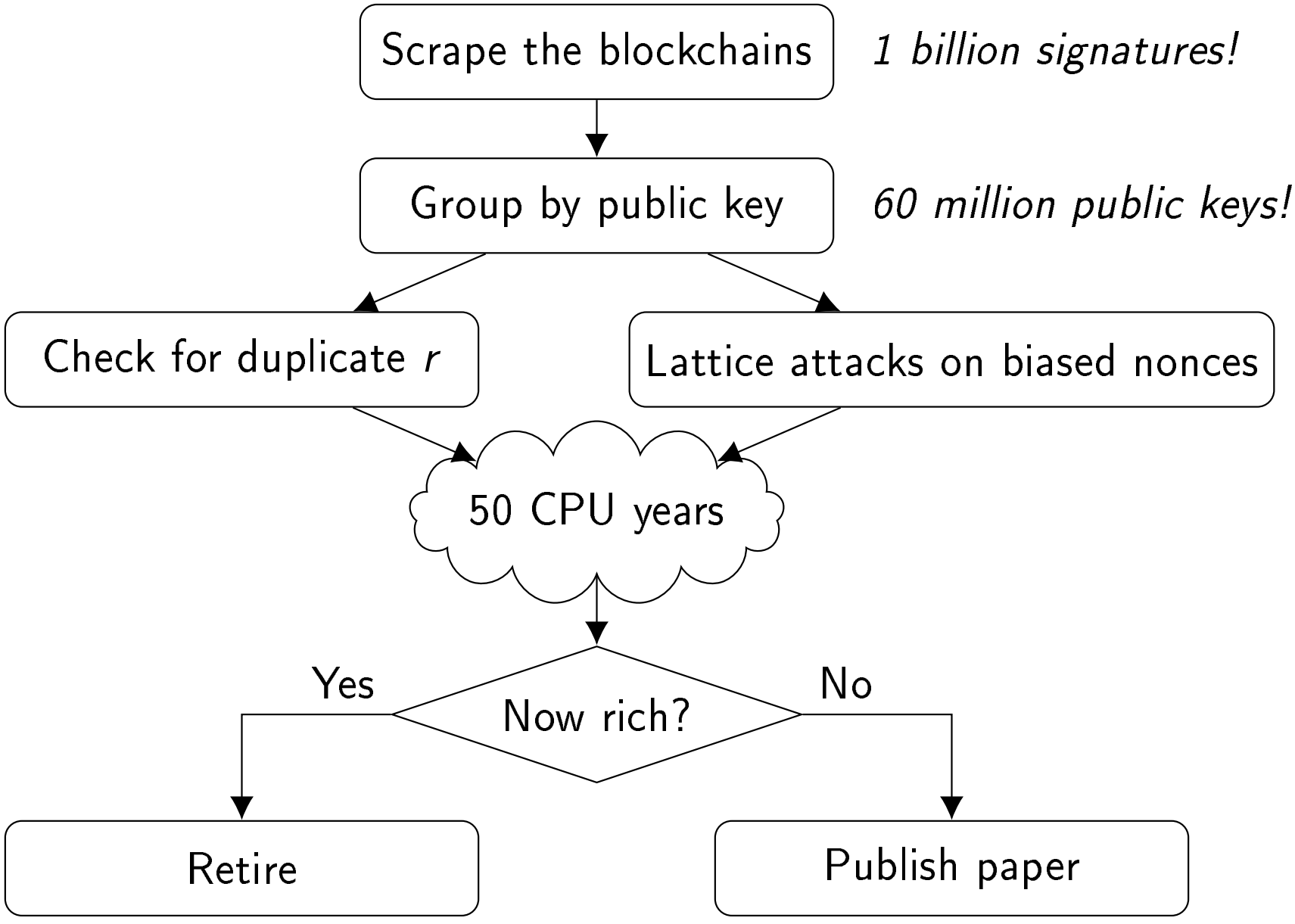
Where to find billions of ECDSA keys and signatures...

HTTPS SSH

Extracting signatures and keys from cryptocurrencies

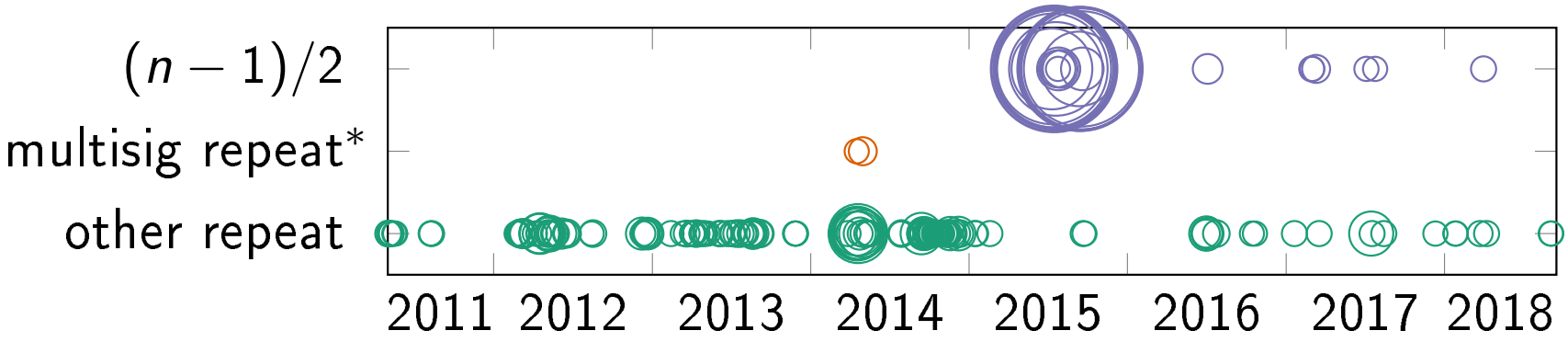
- Bitcoin, Ethereum, and Ripple all use `secp256k1`
- Sender signs hash h of transaction.
- An "address" is a hash of a public key.
- Public key revealed by outgoing transactions *from* an address.

Cryptanalysis program for ECDSA signatures



Repeated nonce results

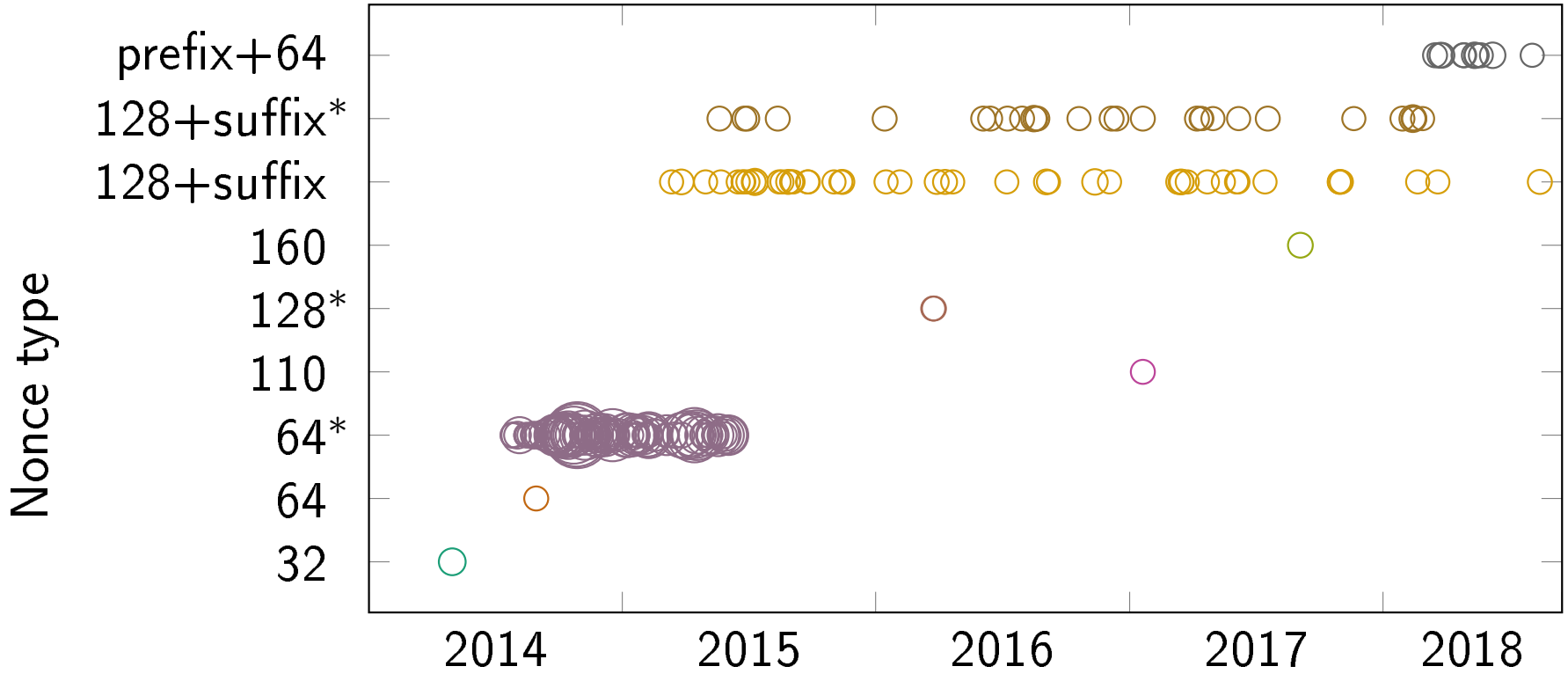
Bitcoin nonces have been analyzed many times since 2013



- Bitcoin: 2.5M signatures with non-unique k from 1300 keys.
- Ethereum: 185 signatures; 3 keys.
- Ripple: 21 signatures; 1 key; 30 XRP.

Biased nonce results

(New; our results.)



- Bitcoin: 6000 signatures from 300 keys; 0.008 Bitcoin.
- Ethereum: 5 signatures from 1 key; 0.00002 Ether.
- SSH: 80 signatures from 4 keys.

Weird trick 1: Small signatures for dust transactions

- 99.9% of the repeated Bitcoin nonce values are
  `0x7fffffffffffffffffffffffffffffff5d576e7357a4501ddfe92f46681b20a0`
- This is $(n-1)/2$ where n is the order of `secp256k1`.
- The x-coordinate of $G \cdot (n-1)/2$ has 166 bits.
- Signatures shorter by 11 bytes.
- Greg Maxwell suggested this to clear "dust" transactions.

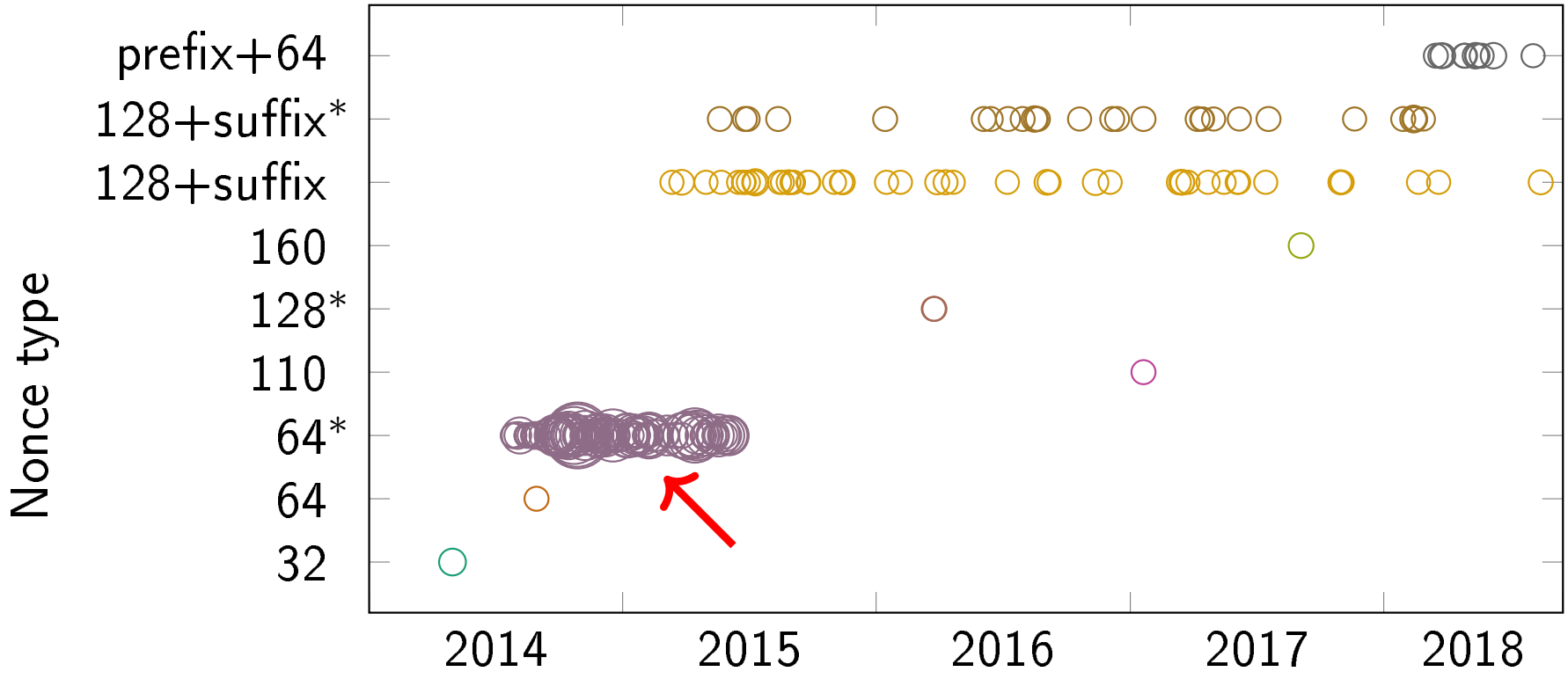
A mystery: Why does G have this property?

Screwup 2: Human factors

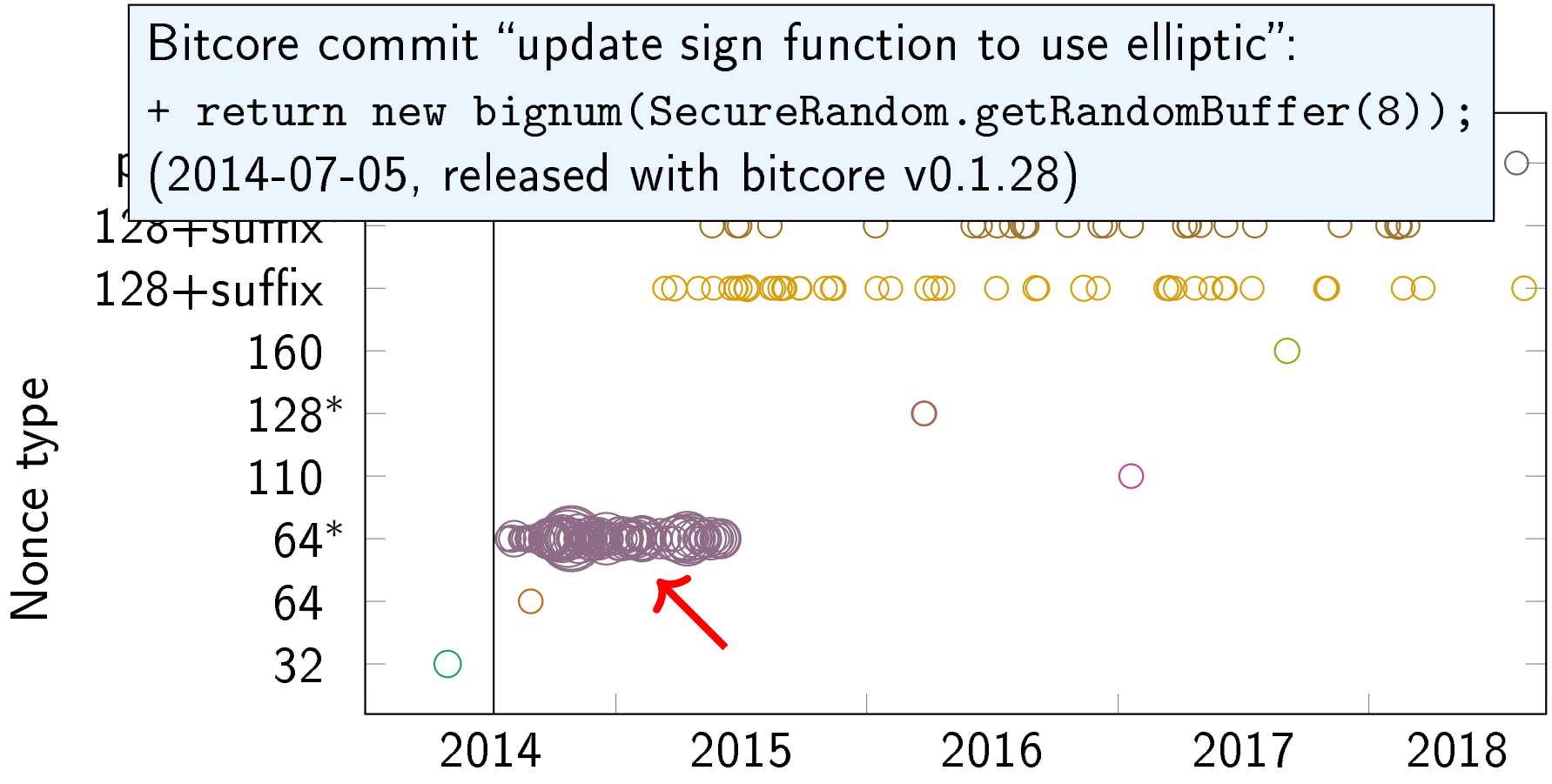
- We traced one compromised key to `darkwallet.is`.
- Part of a 3-out-of-5 multisig address, used for donations `31oSGBBNrpCiENH3XMZpiP6GTC4tad4bMy`.
- Holds 17 BTC $\approx$ ~~$110k~~ $60k.
- Amir Taaki, one the authors of `darkwallet.is`, explained:

  It's either me (I was calculating the signatures manually) or my friend who was working on darkwallet (it might have been an earlier version)

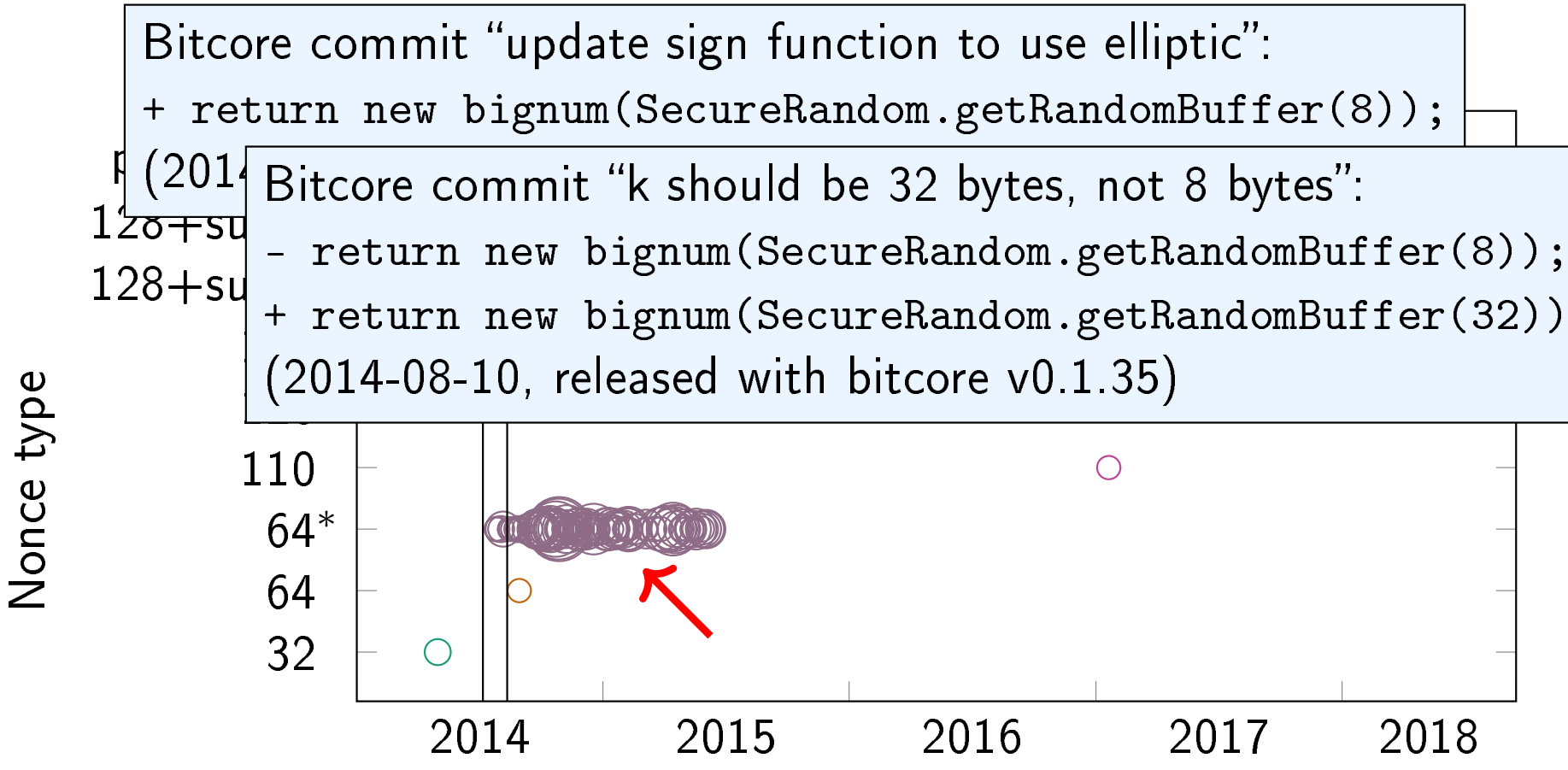
Screwup 3: The Bitpay multisig disaster



Screwup 3: The Bitpay multisig disaster



Screwup 3: The Bitpay multisig disaster



HT to Gregory Maxwell for finding this.

Screwup 4: The SHA-256 round constant

60 signatures by SSH servers with a shared 32bit suffix:

nonce k	secret key d
c010..85eaf27871c6	362e..33f9
f021..cb18f27871c6	362e..33f9
⋮	362e..33f9
1d39..69cef27871c6	362e..33f9
0009..e58ef27871c6	362e..33f9
9e00..4620f27871c6	ca42..3ad7
a8d8..f92ff27871c6	ca42..3ad7
⋮	ca42..3ad7
7aad..0f5bf27871c6	ca42..3ad7
20c1..5dd1f27871c6	ca42..3ad7
b620..447cf27871c6	713a..f2fa
3478..0fabf27871c6	713a..f2fa
⋮	713a..f2fa
4738..0017f27871c6	713a..f2fa
25b1..6638f27871c6	713a..f2fa

Screwup 4: The SHA-256 round constant

60 signatures by SSH servers with a shared 32bit suffix:

nonce k	secret key d
c010..85eaf27871c6	362e..33f9
f021..cb18f27871c6	362e..33f9
⋮	362e..33f9
1d39..69cef27871c6	362e..33f9
0009..e58ef27871c6	362e..33f9
9e00..4620f27871c6	ca42..3ad7
a8d8..f92ff27871c6	ca42..3ad7
⋮	ca42..3ad7
7aad..0f5bf27871c6	ca42..3ad7
20c1..5dd1f27871c6	ca42..3ad7
b620..447cf27871c6	713a..f2fa
3478..0fabf27871c6	713a..f2fa
⋮	713a..f2fa
4738..0017f27871c6	713a..f2fa
25b1..6638f27871c6	713a..f2fa

SHA-224 and SHA-256 use the same sequence of sixty-four constant 32-bit words, K0, K1, ..., K63. These words represent the first 32 bits of the fractional parts of the cube roots of the first sixty-four prime numbers. In hex, these constant words are as follows (from left to right):

```
428a2f98 71374491 b5c0fbcf e9b5dba5
3956c25b 59f111f1 923f82a4 ab1c5ed5
d807aa98 12835b01 243185be 550c7dc3
72be5d74 80deb1fe 9bdc06a7 c19bf174
e49b69c1 efbe4786 0fc19dc6 240ca1cc
2de92c6f 4a7484aa 5cb0a9dc 76f988da
983e5152 a831c66d b00327c8 bf597fc7
c6e00bf3 d5a79147 06ca6351 14292967
27b70a85 2e1b2138 4d2c6dfc 53380d13
650a7354 766a0abb 81c2c92e 92722c85
a2bfe8a1 a81a664b c24b8b70 c76c51a3
d192e819 d6990624 f40e3585 106aa070
19a4c116 1e376c08 2748774c 34b0bcb5
391c0cb3 4ed8aa4a 5b9cca4f 682e6ff3
748f82ee 78a5636f 84c87814 8cc70208
90befffa a4506ceb bef9a3f7 c67178f2
```

Screwup 5: Memory-unsafe code

54 signatures with a shared 128bit suffix had a peculiar cause:

nonce k	secret key d
30e7..6b9f0000..0000	0000..00000000..000b
7d52..688f0000..0000	0000..00000000..000b
02a9..4fcc0000..0000	0000..00000000..0018
232c..daba0000..0000	0000..00000000..0018
1315..80860f80..710b	0f80..710b75f7..ae4b
3da9..42420f80..710b	0f80..710b75f7..ae4b
60fe..970c0f80..710b	0f80..710b75f7..ae4b
32c4..b2ad448e..e255	448e..e25525a3..9d39
5e22..ef90448e..e255	448e..e25525a3..9d39
750c..3600448e..e255	448e..e25525a3..9d39
7917..0cde448e..e255	448e..e25525a3..9d39
1c9a..ec714c7a..0d8a	4c7a..0d8a35f8..c9ab
1d5f..7e434c7a..0d8a	4c7a..0d8a35f8..c9ab

Screwup 5: Memory-unsafe code

54 signatures with a shared 128bit suffix had a peculiar cause:

nonce k	secret key d
30e7..6b9f0000..0000	0000..00000000..000b
7d52..688f0000..0000	0000..00000000..000b
02a9..4fcc0000..0000	0000..00000000..0018
232c..daba0000..0000	0000..00000000..0018
1315..80860f80..710b	0f80..710b75f7..ae4b
3da9..42420f80..710b	0f80..710b75f7..ae4b
60fe..970c0f80..710b	0f80..710b75f7..ae4b
32c4..b2ad448e..e255	448e..e25525a3..9d39
5e22..ef90448e..e255	448e..e25525a3..9d39
750c..3600448e..e255	448e..e25525a3..9d39
7917..0cde448e..e255	448e..e25525a3..9d39
1c9a..ec714c7a..0d8a	4c7a..0d8a35f8..c9ab
1d5f..7e434c7a..0d8a	4c7a..0d8a35f8..c9ab

Screwup 5: Memory-unsafe code

Possible explanation:

```
char *create_signature(char *secret_key, char *hash) {
  char k[32];
  char d[16];
  fill_random(k, 32);
  memcopy(d, secret_key, 32);
  ...
  return signature;
}
```

Screwup 5: Memory-unsafe code

Possible explanation:

```
char *create_signature(char *secret_key, char *hash) {
  char k[32];
  char d[16];
  fill_random(k, 32);
  memcopy(d, secret_key, 32); // facepalm
  ...
  return signature;
}
```

A simple countermeasure

Use deterministic (EC)DSA!

$k = H(d \,||\, h)$

Bitcoin, Ethereum, Ripple official libraries already do.

ed25519 builds in deterministic nonce generation from the start.

Summary

Lattices are a powerful tool especially in side-channel attack scenarios where an attacker can observe partial information about secrets like keys or nonces.