Biased nonce sense

Joachim Breitner
formerly University of Pennsylvania

Nadia Heninger
formerly University of Pennsylvania
now UC San Diego

GPN 20
Karlsruhe, May 20, 2022

Theorem (Law of large numbers)

Average behavior converges almost surely to the expected value as the number of samples increases.

Theorem (Law of large numbers)

Average behavior converges almost surely to the expected value as the number of samples increases.

Adage (Law of truly large numbers [Diaconis and Mosteller])

When a sample size is large enough, any outrageous thing is likely to happen.

Theorem (Law of large numbers)

Average behavior converges almost surely to the expected value as the number of samples increases.

Adage (Law of truly large numbers [Diaconis and Mosteller])

When a sample size is large enough, any outrageous thing is likely to happen.

Conjecture (Cryptographic law of truly large numbers)

Given samples from enough independent cryptographic implementations, any outrageous vulnerability is likely to be present.

ECDSA (Elliptic Curve Digital Signature Algorithm)

Global Parameters Elliptic curve E, point G on E of order n.

Private Key	Public Key
Integer d	Curve point $Q = dG$

Signature Generation

Message Hash: h

Per-signature "nonce": Integer k

Signature on h: (r, s) $\quad r = (kG)_x \quad s = k^{-1}(h + dr) \bmod n$

Potential ECDSA disasters: Public nonce

Global Parameters Elliptic curve E, point G on E of order n.

Private Key	Public Key
Integer d	Curve point $Q = dG$

Signature Generation

Message Hash: h

Per-signature "nonce": Integer k

Signature on h: (r, s) $\quad r = (kG)_x \quad s = k^{-1}(h + dr) \bmod n$

Potential pitfall #1

Nonce k must remain secret, or else the secret key d is revealed.

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

Potential ECDSA disasters: Repeated nonce

Global Parameters Elliptic curve E, point G on E of order n.

Private Key
Integer d

Public Key
Curve point $Q = dG$

Signature Generation

Message Hash: h

Per-signature "nonce": Integer k

Signature on h: (r, s) $\quad r = (kG)_x \quad s = k^{-1}(h + dr) \bmod n$

Potential pitfall #2

If k is ever reused to sign distinct messages h_1, h_2, it is revealed

$$k = (h_1 - h_2)(s_1 - s_2)^{-1} \bmod n$$

and thus the long-term private key d is revealed.

Potential ECDSA disasters: Biased nonce

[Boneh Venkatesan 96], [Howgrave-Graham Smart 2001], [Nguyen Shparlinski 2003]

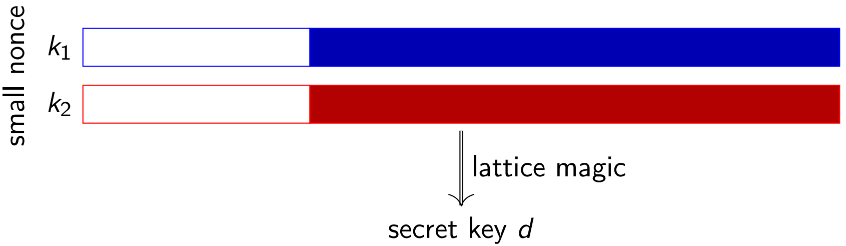
Potential pitfall #3

k must be generated uniformly at random,
or we can use many signatures to compute the private key d.

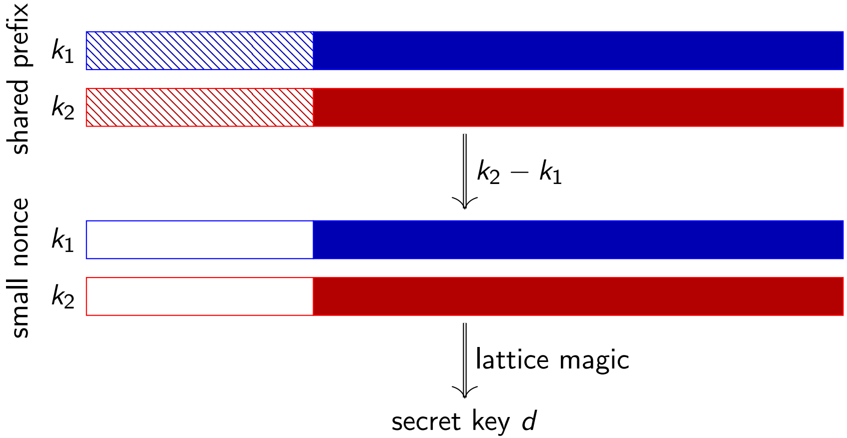
$$\left.\begin{aligned} k_1 - s_1^{-1} r_1 d - s_1^{-1} h_1 &\equiv 0 \bmod n \\ k_2 - s_2^{-1} r_2 d - s_2^{-1} h_2 &\equiv 0 \bmod n \\ &\vdots \\ k_m - s_m^{-1} r_m d - s_m^{-1} h_m &\equiv 0 \bmod n \end{aligned}\right\} \rightarrow \text{lattice magic} \rightarrow d$$

If the k_i are *small*, system of equations likely has unique solution and lattice techniques can find d.

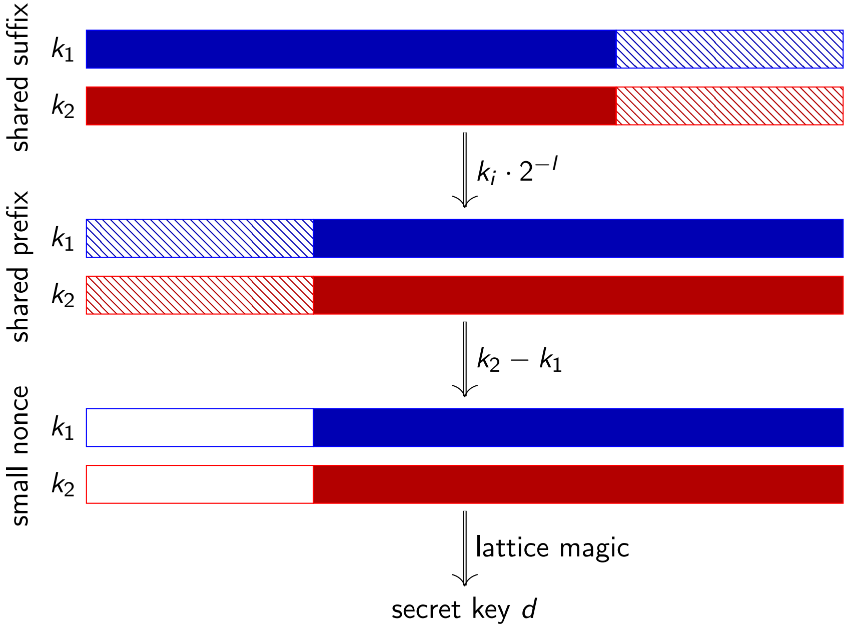
Extending to other biases



Extending to other biases



Extending to other biases



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

bitcoin

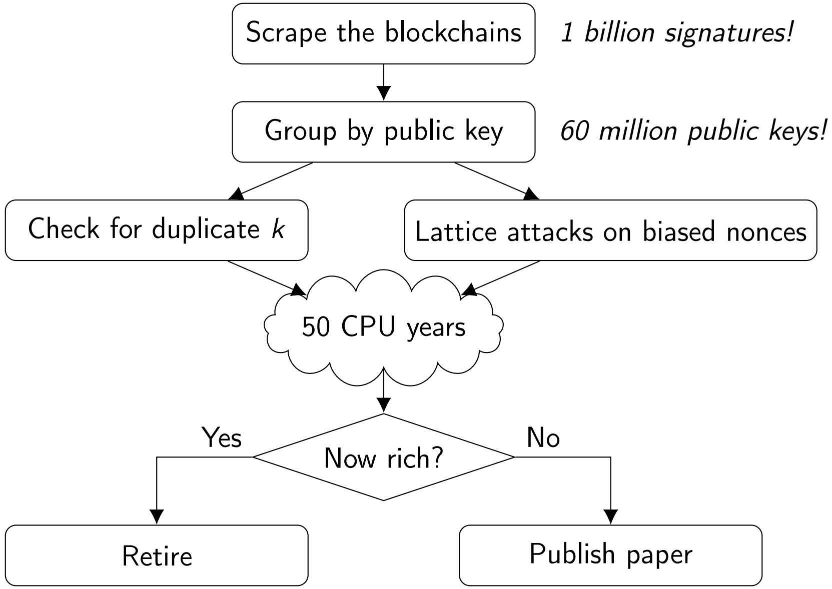
ethereum

ripple

HTTPS

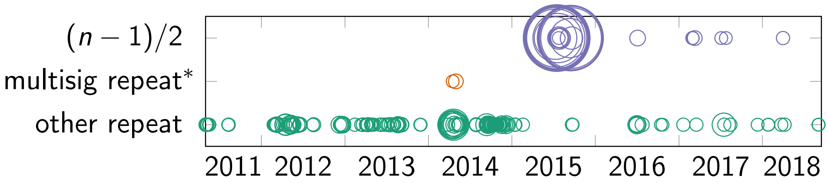
SSH

Cryptanalysis program for ECDSA signatures



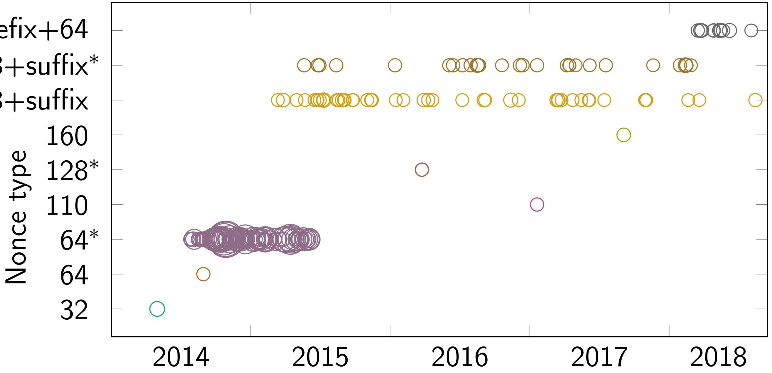
Repeated nonce results

Bitcoin nonces have been analyzed many times since 2013



- Bitcoin: 2.5M signatures k from 1300 keys.
  People check for this and and steal!
- Ethereum: 185 signatures; 3 keys.
- Ripple: 21 signatures; 1 key; 30 XRP.

Biased nonce 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.

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 ≈ ~~$110k~~ ~~$60k~~ ~~$172k~~ €484k.
- 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 2: More Human factors

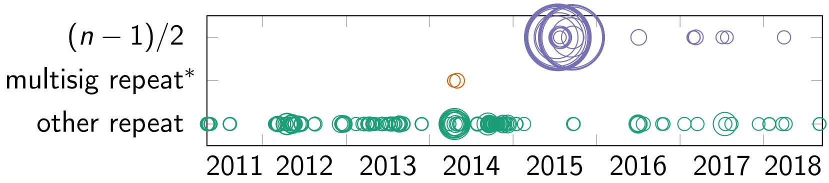
After finding some very small nonces, we brute forced all 32-bit nonces.

- 275 signatures from 52 keys.

Observed nonce values: 1, 2, 9, 100, 1337, 13337, 133337, 1333337, 12345678, and 2147491839

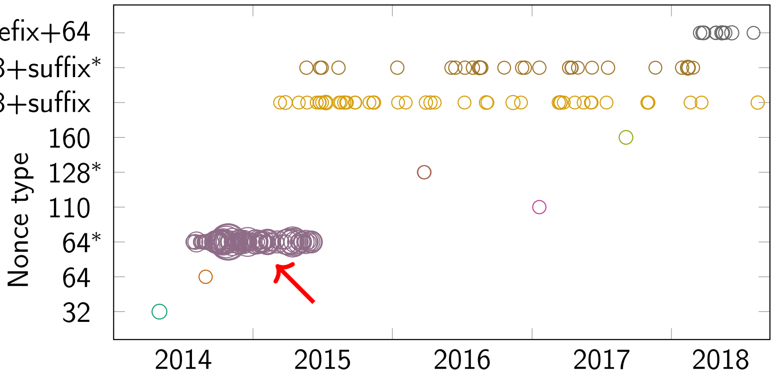
Clearly hand-generated signatures.

Screwup 3: Client RNG Vulnerabilities

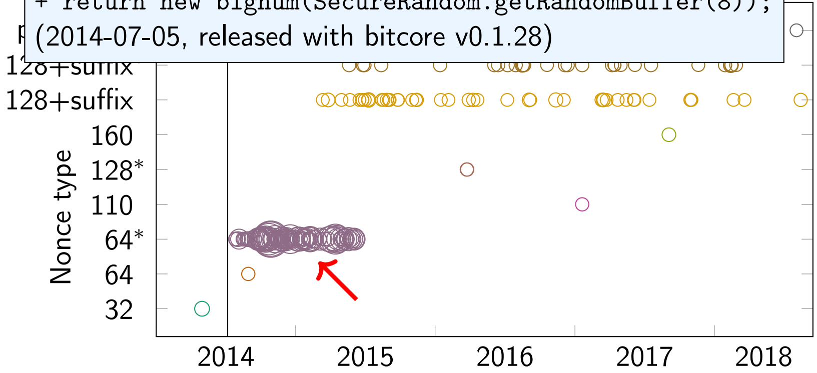


- August 11, 2013: Android SecureRandom() vulnerability disclosed.
- January 4, 2015: random.org/blockchain.info RNG vulnerability

Screwup 4: The Bitpay multisig disaster



Screwup 4: The Bitpay multisig disaster

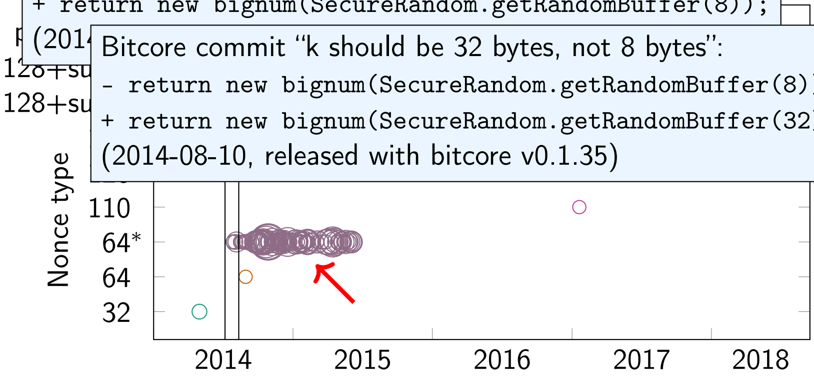


Bitcore commit "update sign function to use elliptic":

```
+ return new bignum(SecureRandom.getRandomBuffer(8));
```

(2014-07-05, released with bitcore v0.1.28)

Screwup 4: The Bitpay multisig disaster



Bitcore commit "update sign function to use elliptic":

```
+ return new bignum(SecureRandom.getRandomBuffer(8));
```

(2014

Bitcore commit "k should be 32 bytes, not 8 bytes":

```
- return new bignum(SecureRandom.getRandomBuffer(8));
+ return new bignum(SecureRandom.getRandomBuffer(32));
```

(2014-08-10, released with bitcore v0.1.35)

HT to Gregory Maxwell for finding this.

Screwup 5: 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`
$\vdots$	`362e..33f9`
`1d39..69cef27871c6`	`362e..33f9`
`0009..e58ef27871c6`	`362e..33f9`
`9e00..4620f27871c6`	`ca42..3ad7`
`a8d8..f92ff27871c6`	`ca42..3ad7`
$\vdots$	`ca42..3ad7`
`7aad..0f5bf27871c6`	`ca42..3ad7`
`20c1..5dd1f27871c6`	`ca42..3ad7`
`b620..447cf27871c6`	`713a..f2fa`
`3478..0fabf27871c6`	`713a..f2fa`
$\vdots$	`713a..f2fa`
`4738..0017f27871c6`	`713a..f2fa`
`25b1..6638f27871c6`	`713a..f2fa`

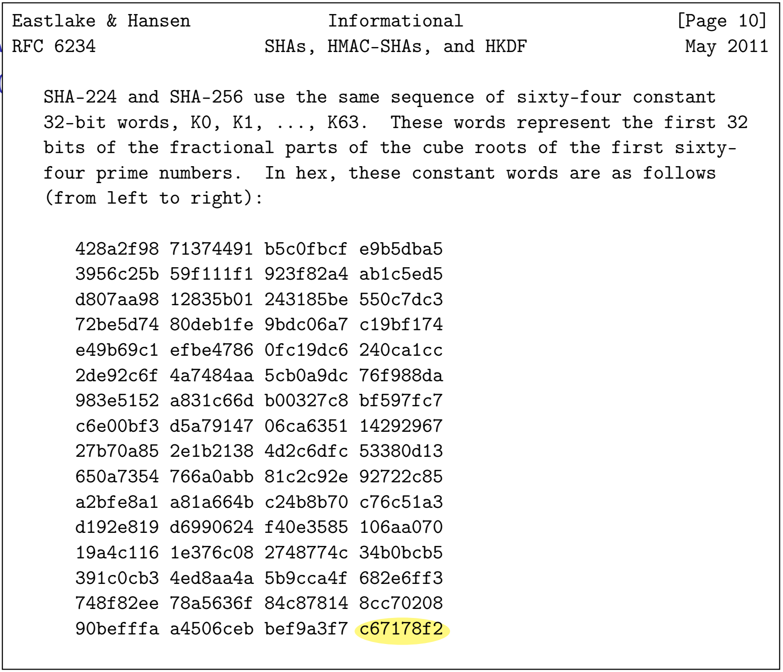
Screwup 5: The SHA-256 round constant

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

Scre

6



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
```

25b1..6638f27871c6   713a..f2fa

Screwup 6: 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 6: 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 6: 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 6: 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 6: 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;
}
```

Screwup 6: ~~Memory-unsafe code~~

Breaking news (this morning)

Actually simply $k = h[:16] \mid\mid d[:16]$

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.

More in the paper!

- More math!
- Full algorithmic details!
- Tables with numbers!
- More examples of bad implementations!
- Why is $G/2$ small? (conspiracy theory material)
- How to brute-force 64 bit nonces!
- The snag with the normalized signatures!

Biased Nonce Sense:
Lattice Attacks against Weak ECDSA Signatures in Cryptocurrencies.
FC 2019 Joachim Breitner and Nadia Heninger.
`https://eprint.iacr.org/2019/023`

Backup slides

\begin{scary lattice section}

Formulating ECDSA as a hidden number problem

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

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

$$\begin{gathered} k_1 - t_1 d - a_1 \equiv 0 \bmod n \\ k_2 - t_2 d - a_2 \equiv 0 \bmod n \\ \vdots \\ k_m - t_m d - a_m \equiv 0 \bmod n \end{gathered}$$

We assume the k_i are small.

(Instance of the *hidden number problem* [Boneh Venkatesan 96].)

Solving the hidden number problem with CVP

Input:
$$\begin{aligned} k_1 - t_1 d - a_1 &\equiv 0 \bmod n \\ &\vdots \\ k_m - t_m d - a_m &\equiv 0 \bmod n \end{aligned}$$

in unknowns $k_1, \ldots, k_m, d$, where $|k_i| < B$.

Construct the lattice basis

$$M = \begin{bmatrix} n & & & & \\ & n & & & \\ & & \ddots & & \\ & & & n & \\ t_1 & t_2 & \ldots & t_m & \end{bmatrix}$$

Solve CVP with target vector $v_t = (a_1, a_2, \ldots, a_m)$.

$v_k = (k_1, k_2, \ldots, k_m)$ will be the distance.

Solving the hidden number problem with CVP embedding

Input:

$$\begin{gathered} k_1 - t_1 d - a_1 \equiv 0 \bmod n \\ \vdots \\ k_m - t_m d - a_m \equiv 0 \bmod n \end{gathered}$$

in unknowns $k_1, \ldots, k_m, d$, where $|k_i| < B$.

LLL, BKZ implementations better than CVP implementations.

Construct the lattice basis

$$M = \begin{bmatrix} n & & & & & \\ & n & & & & \\ & & \ddots & & & \\ & & & n & & \\ t_1 & t_2 & \ldots & t_m & B/n & \\ a_1 & a_2 & \ldots & a_m & & B \end{bmatrix}$$

$v_k = (k_1, k_2, \ldots, k_m, Bd/n, B)$ is a short vector in this lattice.

Solving the hidden number problem with CVP embedding

Construct the lattice

$$M = \begin{bmatrix} n & & & & & \\ & n & & & & \\ & & \ddots & & & \\ & & & n & & \\ t_1 & t_2 & \dots & t_m & B/n & \\ a_1 & a_2 & \dots & a_m & & B \end{bmatrix}$$

Want vector
$v_k = (k_1, k_2, \dots, k_m, Bd/n, B)$

We have:

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

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

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

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

Solving the hidden number problem with lattices

We expect to find v_k when

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

Minimizing the number of samples, for 256-bit n:

#samples	$\log \lvert k \rvert$
2	128
3	170
4	190
20	242
40	248

\end{scary lattice section}

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.
- Transactions recorded on each currency’s blockchain.
- Download client, sync blockchain, extract signatures.

Bitcoin	Calculating h from transaction is complicated. Patched client to log h, revalidated chain.
Ethereum	API conveniently provides h. We ran a local client, extracted data via RPC.
Ripple	API conveniently provides h. We extracted data via RPC from the public node.

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 $(1/2) \cdot G$ has 166 bits instead of 256.
- Signatures shorter by 11 bytes.
- Greg Maxwell suggested this to clear "dust" transactions.

A mystery: Why does G have this property?

Searching for more 64-bit nonces

Precomputation:

1. Precompute 2.2TB hash table of 2^{39} elements.
2. Precompute 2^{32} lookup table of logs of these elements.
   (Took $\approx$ 4 days on a few hundred nodes to precompute.)

Online computation:

3. For each of our 2^{30} signatures, do 2^{25} work to look up.
4. On one machine with 48 cores and 3TB RAM,
   17 days to look up 140,000 signatures.
   (10s per lookup).

Tentative conclusion: 64-bit nonces are not much more common than the ones we found.