Attacks on Schnorr signatures with biased nonces

Mehdi Tibouchi

NTT Secure Platform Laboratories

ECC Workshop, 2017–11–13

Outline

Schnorr signatures with biased nonces
- Schnorr signatures
- Nonce biases

The lattice approach
- Description of the attack
- Limitations and extensions

The statistical approach
- Attack overview
- Using Schroeppel–Shamir

Outline

Schnorr signatures with biased nonces

- Schnorr signatures
- Nonce biases

The lattice approach

- Description of the attack
- Limitations and extensions

The statistical approach

- Attack overview
- Using Schroeppel–Shamir

Schnorr signatures

- Public parameters: cyclic group $\mathbb{G}$ of prime order q, generator g, hash function $H: \{0,1\}^* \times \mathbb{G} \to \mathbb{Z}/q\mathbb{Z}$
- Key pair: secret $x \xleftarrow{\$} \mathbb{Z}/q\mathbb{Z}$, public $h = g^x$

$\mathrm{Sign}(x, m)$

1. $k \xleftarrow{\$} \mathbb{Z}/q\mathbb{Z}$
2. $r \leftarrow g^k$
3. $h \leftarrow H(m, r)$
4. $s \leftarrow k - hx \bmod q$
5. **return** (h, s)

EC-Schnorr signatures

- Public parameters: elliptic curve $E/\mathbb{F}_p$, point $P \in E(\mathbb{F}_p)$ of prime order q, hash function $H\colon \{0,1\}^* \times \mathbb{F}_p \to \mathbb{Z}/q\mathbb{Z}$
- Key pair: secret $x \xleftarrow{\$} \mathbb{Z}/q\mathbb{Z}$, public $Q = [x]P$

$\mathrm{Sign}(x, m)$

1. $k \xleftarrow{\$} \mathbb{Z}/q\mathbb{Z}$
2. $(u, v) \leftarrow [k]P$
3. $h \leftarrow H(m, u)$
4. $s \leftarrow k - hx \bmod q$
5. **return** (h, s)

Security and variants

- Secure (EUF-CMA) if the discrete logarithm is hard in $\mathbb{G}$ in the ROM for H
- Common variants:
  - Hash the public key as well
  - Deterministic k, e.g. $k = \mathrm{MAC}_S(m)$ for an auxiliary key S
  - Give out g^k (resp. $k[P]$) instead of h in the signature
- EdDSA ≈ EC-Schnorr on X25519; qDSA ≈ Schnorr on Kummer lines/Kummer surfaces
- DSA, ECDSA: badly designed variants of Schnorr (for patent reasons)
- Results in this talk apply to all of the above
  - Caveat: specific ways of inducing nonce biases may only apply to a subset of them

Outline

Schnorr signatures with biased nonces
- Schnorr signatures
- Nonce biases

The lattice approach
- Description of the attack
- Limitations and extensions

The statistical approach
- Attack overview
- Using Schroeppel–Shamir

Sensitivity of the nonce

- The random value k in signature generation usually called the nonce
- Should never be repeated! If (h, s), (h', s') signatures on m, m' with the same value k, we have:

  $$s \equiv k - hx \pmod{q} \qquad s' \equiv k - h'x \pmod{q}$$

  Subtract the two:

  $$(h' - h) \cdot x \equiv s - s' \pmod{q}$$

  Immediate recovery of the secret key x
- That attack (for ECDSA) was applied to the Sony PlayStation 3. Also used to steal some Bitcoins

Sensitivity of the nonce (cont'd)

- Nonce = value that can only be used once
- However Schnorr nonces are even more sensitive than that!
- k should (statistically close to) uniform in $\mathbb{Z}/q\mathbb{Z}$. Significant biases can be used reveal the key
- Intuition: linear relation

$$x = h^{-1} \cdot (-s + k) \bmod q$$

$\rightsquigarrow$ partial info. on k (e.g. ℓ known bits) should translate to partial info. on x (ℓ bits of info)

How do biases occur?

- Incorrect implementation
  - PlayStation 3
  - GLV/GLS setting: k implicitly defined as $k_1 + \lambda k_2$ with k_1, k_2 of roughly half size; if "half size" is interpreted as $\lceil (\log_2 q)/2 \rceil$ bits, bias can occur
- Poor random number generators
- Side-channel leakage
  - e.g. emanations during scalar multiplication revealing the first few LSBs of k
- Fault attacks
  - errors injected before/during the computation of $[k]P$ forcing k to a biased value

Classical fault attack on ECDSA

- Successfully demonstrated by Naccache et al. against 8-bit smartcards (PKC 2005)
- Upon signature generation, new k generated uniformly at random in $\mathbb{Z}/q\mathbb{Z}$
- Typically done machine word by machine word (sample each word with a random number generator), with rejection sampling at the end
- Glitch attack: inject a fault during the random sampling loop to cause an early exit, so LSBs or MSBs of k are left equal to zero
- Usual countermeasure: use double loop counters

New: a fault attack on qDSA

- Fault injection on a much less protected part of signature generation: the group generator P
  - usual ECDSA/EC-Schnorr: a random fault yields a point $\widetilde{P}$ outside the curve $\rightsquigarrow$ scalar multiplication makes no sense
  - qDSA: x-only arithmetic on X25519 $\rightsquigarrow$ result on the curve or its twist
- With prob. $\approx 1/4$, the faulty generator $\widetilde{P}$ is on the curve itself and has order $4q$
- qDSA signatures include $\pm R = \pm[k]P$ instead of the hash
  - faulty case $[q]\widetilde{R} = [k]\big([q]\widetilde{P}\big)$ point of order 4, revealing the 2 LSBs of k
  - only known up to sign $\rightsquigarrow$ deduce if the 2 LSBs are 00, 10 or ?1
- Suppose we can generate many signatures with the same $\widetilde{P}$ (semi-permanent fault setting)
  - Can check that $\widetilde{P}$ has order $4q$
  - Mount attack with 2-bit nonce bias (throw away the ?1 case)

Exploiting the nonce biases

- Given biases nonces k, two main approaches to recover x
- Lattice-based attack (Howgrave-Graham–Smart; Nguyen–Shparlinski)
  - based on solving BDD in a lattice
  - requires relatively few signatures
  - for large biases ($\gtrsim 5$ bits depending on the size of q), very efficient in practice
  - for small biases, impractical (lattice dim. too large) or even inapplicable (hidden vector not close enough)
  - cannot use more data
  - bias must be "predictable"
- Statistical attack (Bleichenbacher)
  - based on purely statistical techniques (FFT)
  - requires many signatures, large space complexity
  - can in principle deal with arbitrarily small biases
  - more data improves the attack
  - irregular biases OK

Current records

- Lattice-based attack
  - 160 bits: 2-bit bias done ([LN13], ≈ 100 sigs., BKZ–90), 1-bit infeasible
  - 256 bits: 4-bit easy, 3-bit not easy, 2-bit infeasible?
  - 384 bits: 6-bit easy, 5 or 4-bit not so easy, 3-bit infeasible?
- Statistical attack
  - 160 bits: 1-bit bias done ([AFGKTZ14], $\approx 2^{30}$ sigs., 1 TB RAM)
  - 256 bits: 2-bit looks hard, 1-bit possible with nation-state resources and many sigs.?
  - 384 bits: 5-bit done ([DHMP13], ≈ 4000 sigs.), 4-bit feasible?, 3-bit hard?

Outline

Schnorr signatures with biased nonces
Schnorr signatures
Nonce biases

The lattice approach
Description of the attack
Limitations and extensions

The statistical approach
Attack overview
Using Schroeppel–Shamir

Formal attack setting

- We obtain n faulty signatures (h_i, s_i) on messages m_i
- Each signature generated with nonce k_i with ℓ LSBs equal to zero:

$$k_i = 2^\ell b_i, \quad (0 \le b < q/2^\ell)$$

- We thus get relations:

$$h_i \cdot x \equiv 2^\ell b_i - s_i \pmod{q}$$

which we can rewrite as:

$$x \equiv u_i + v_i b_i \pmod{q}$$

for known constants $u_i = -s_i/h_i \bmod q$, $v_i = 2^\ell/h_i \bmod q$.

Orthogonal lattices

- Rewrite the previous relation in vector form:

$$x \equiv \langle \mathbf{u}_i, \mathbf{b} \rangle \pmod{q}$$

with:

$$\mathbf{b} = (a, b_1, \dots, b_n) \in \mathbb{Z}^{n+1}$$
$$\mathbf{u}_i = (u_i/a \bmod q, 0, \dots, 0, v_i, 0, \dots, 0) \in \mathbb{Z}^{n+1}$$

- In particular, $\mathbf{b}$ orthogonal mod q to $\mathbf{u}_1 - \mathbf{u}_2$, $\mathbf{u}_2 - \mathbf{u}_3$, ..., $\mathbf{u}_{n-1} - \mathbf{u}_n$
- Introduce the lattice L of vectors in $\mathbb{Z}^{n+1}$ orthogonal to those $n-1$ vectors mod q, and such that the first component is a multiple of a
- $\mathbf{b} \in L$, relatively short

Recovering b

- L is the kernel of the map $\mathbb{Z}^{n+1} \to (\mathbb{Z}/a\mathbb{Z}) \times (\mathbb{Z}/q\mathbb{Z})^{n-1}$:

$$\mathbf{b} \mapsto \big(b_1 \bmod a, \langle \mathbf{b}, \mathbf{u}_1 - \mathbf{u}_2 \rangle \bmod q, \dots, \langle \mathbf{b}, \mathbf{u}_{n-1} - \mathbf{u}_n \rangle \bmod q\big)$$

  surjective with high probability
- Therefore $\mathrm{vol}(L) = [\mathbb{Z}^{n+1} : L] = a \cdot q^{n-1}$
- Gaussian heuristic: the shortest vector in L should be of length approximately

$$\lambda = \frac{n+1}{2\pi e} \cdot \mathrm{vol}(L)^{1/(n+1)}$$

- We can hope to recover $\mathbf{b}$ if $\|\mathbf{b}\| \ll \lambda$. Choosing $a = q/2^\ell$, we have $\mathbf{b} \leq \sqrt{n+1} \cdot q/2^\ell$
- Recovering $\mathbf{b}$ of course reveals the secret key x

Condition on n

- The size condition is thus:

$$\sqrt{n+1}\cdot\frac{q}{2^\ell} \ll \sqrt{\frac{n+1}{2\pi e}}\left(\frac{q}{2^\ell}\cdot q^{n-1}\right)^{1/(n+1)}$$

  which simplifies to:

$$\frac{n}{n+1}\ell \gtrsim \frac{1}{n+1}\log_2 q + \log_2\sqrt{2\pi e}$$

- In particular, the attack only works when $\ell > \log_2\sqrt{2\pi e} \approx 2.05)$
  - constant slightly too large: can be improved by using a centered **b**, and taking expected size into account
- For fixed ℓ, we need a number of faulty signatures satisfying:

$$n \gtrsim \frac{\log_2(q\sqrt{2\pi e})}{\ell - \log_2\sqrt{2\pi e}}$$

- Large ℓ: close to "information-theoretic" bound $(\log_2 q)/\ell$

Outline

Schnorr signatures with biased nonces
- Schnorr signatures
- Nonce biases

The lattice approach
- Description of the attack
- Limitations and extensions

The statistical approach
- Attack overview
- Using Schroeppel–Shamir

Remarks on the attack

- Different from traditional presentation of the attack (uSVP vs. BDD), but mostly equivalent
- Must know the size of the bias(es) to construct the lattice
- As already mentioned: hard limit on how small the bias can get
- Having more signatures doesn't seem to help

Structure of the bias

- Attack uses in a crucial way the zero LSBs form of the bias
  - known LSBs/MSBs of course also OK
- Does not generalize easily to more general bias structure
  - doable: string of zero bits in the middle at known position
  - hard?: string of zero bits in the middle at unknown position
- Recent generalizations with some practical relevance
  - zero LSBs/MSBs in the τ-adic expansion of k for Kobliz curves [BFMT16]
  - zero LSBs/MSBs in k_i for $k = \sum k_i \lambda_i$ GLV/GLS decomposition
  - fun fact: can use lattice reduction over Euclidean rings
- Hard to formulate general result?

Outline

Schnorr signatures with biased nonces
Schnorr signatures
Nonce biases

The lattice approach
Description of the attack
Limitations and extensions

The statistical approach
Attack overview
Using Schroeppel–Shamir

Bleichenbacher's attack

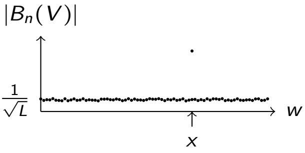
- Before the lattice attack was proposed, Bleichenbacher suggested a different approach based on a Fourier notion of bias
- Requires many more signatures for similar parameters, but applies in principle to arbitrarily small biases
- Presented at an IEEE P1363 meeting in 2000, never formally published. Revisited by De Mulder et al. (CHES 2013), Aranha et al. (ASIACRYPT 2014).

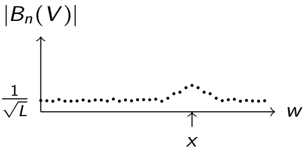
Overview of Bleichenbacher's approach (I)

- Consider again: we are given signatures (h_j, s_j) such that, for the secret key x, the MSBs of the values $k_j = s_j + h_j x \bmod q$ vanish.
- The sampled bias of a set of points $V = (v_0, \cdots, v_{L-1})$ in $\mathbb{Z}/q\mathbb{Z}$ defined as $B_q(V) = \frac{1}{L}\sum_{j=0}^{L-1} e^{2\pi i \cdot v_j/q}$
- Now consider some secret key candidate $w \in \mathbb{Z}/q\mathbb{Z}$ and the corresponding nonce candidates $v_j := s_j + h_j w \mod q$. Claim:
  - if $w \neq x$, $B_q(V) \approx 1/\sqrt{L}$ is small
  - if $w = x$, $B_q(V)$ is close to 1

Range reduction

- Peak of the bias function: only for candidate w exactly equal to x; would need to check all possible $w \in \mathbb{Z}/q\mathbb{Z}$
- Clearly infeasible for large q
- Bleichenbacher's solution: reduce the size of h_j's to $[0, L)$ to broaden the peak
  $\rightarrow$ only need to check L evenly-spaced values in $[0, q)$





Small, sparse linear combinations problem

- How do we carry out this range reduction? Linear combinations!
- Input: a list $(h_0, \ldots, h_{L-1})$ of large, random integers (of 160 bits, say)
  - we can choose L, preferably small
- Looking for: many linear combinations $\sum \omega_i h_i$ which are
  - much smaller, e.g. $|\sum \omega_i h_i| < 2^{32}$
  - very sparse, e.g. $\sum |\omega_i| \leq 16$
- We would like to find many of those linear combinations (say 2^{32})
  - as fast as possible
  - using as little memory as possible,
  - starting with as few h_i's as possible

Idea 1: lattice reduction

- "Short linear combinations" sounds like a lattice problem
- So use lattice reduction? (LLL, BKZ)
  - De Mulder et al.'s approach
- Upside: should be able to start from relatively small L
- Downside: only get a few linear combinations for each lattice we reduce + have to use very large lattice dimensions to find the very sparse combinations we need
- Even a single lattice reduction takes seconds with our parameters, and we need $\approx 2^{32}$ of them: not really practical
- Other issue: for ℓ-bit bias, we really need combinations with coefficients $< 2^{\ell}$
  - Doable for $\ell = 5$, infeasible for $\ell = 2$

Idea 2: sort and difference

- Approach in AC'14 paper: sort-and-difference
  1. Sort the list (h_i) to get $h'_0 < \cdots < h'_{L-1}$
  2. Take the successive differences $h''_0 = h'_1 - h'_0, \ldots, h''_{L-2} = h'_{L-1} - h'_{L-2}$
- We obtain a list of $\approx L$ elements h''_i, linear combinations of two elements h_i each
- On average, two successive elements h'_i, h'_{i+1} should have their $\log_2 L$ MSBs in common
- Hence the h''_i are roughly $\log_2 L$ bit shorter
- Doing this 4 times in total yields $\approx L$ linear combinations of 16 elements h_i, each of size $\approx 160 - 4 \times \log_2 L$
- Works with L a bit larger than 2^{32}, in time $O(L \log L)$ and space $O(L)$ (about 1 TB RAM!)

Idea 3: Schroeppel–Shamir

- Shamir's comment after my presentation at ASIACRYPT: you can get away with a smaller L and less memory by using the Schroeppel–Shamir algorithm
- Basic principle: instead of looking for short combinations of 2 h_i at a time with sort-and-difference, we have techniques to generate short(er) combinations of 4 (or more) h_i in one go
- Recently worked this out with an internship student, A. Takahashi
- Surprise realization after we did this: Schroeppel–Shamir was the method suggested by Bleichenbacher all along!

Outline

Schnorr signatures with biased nonces
Schnorr signatures
Nonce biases

The lattice approach
Description of the attack
Limitations and extensions

The statistical approach
Attack overview
Using Schroeppel–Shamir

Schroeppel–Shamir

- At its core, the Schroeppel–Shamir algorithm lets you do the following
- Given two lists (u_i), (v_i) of N integers, find the M smallest sums $u_i + v_j$ ($M \leq N^2$) in time $O((N + M)\log N)$ and space $O(N)$
- Algorithm is not too complicated if we know the heap data structure:
  1. Assume the list (v_i) is sorted (costs $O(N \log N)$ time)
  2. Store the values $u_i + v_0$ associated with the pairs $(i, 0)$ in a heap (costs $O(N \log N)$ time and $O(N)$ space)
  3. Repeat M times:
  4. Get the smallest element in the heap, which is of the form $u_i + v_j$ (associated with (i, j)), and replace it with $u_i + v_{j+1}$ (costs $O(\log N)$ time)

How Schroeppel–Shamir helps

- So remember our original problem: we have a list of integers, and we want to find short linear combinations of four elements at a time (say)
- To do so, divide the large list into 4 lists (x_i), (y_i), (z_i), (t_j) of the same size N
- Schroeppel–Shamir lets us enumerate the elements $x_i + y_j$ and $z_i + t_j$ in increasing order ((L_n) and (R_m) respectively)
- Easy to find short differences between those elements:
  1. Let $m = n = 0$ and $D = L_0 - R_0$
  2. Repeat:
  3. output $D = L_m - R_n$
  4. if $D > 0$ then increment n else increment m
- We thus heuristically get M elements $x_i + y_j - z_k - t_\ell$ which are $\approx \log_2 M$ bit smaller than the original values (or $M/2^s$ elements which are $s + \log_2 M$ bit smaller), in time $O(M \log N)$ and space $O(N)$

How Schroeppel–Shamir helps (II)

- Numerical application: lets say we start with $N = 2^\alpha$ elements of 160 bits, and use Schroeppel–Shamir to get the entire sorted lists of sums, i.e. $M = N^2$
- First iteration: $2^{2\alpha}$ linear combinations of 4 which are of $\lesssim 160 - 2\alpha$ bits, among which we keep the expected 2^α elements of $160 - 3\alpha$ bits or less
- Second iteration: $2^{2\alpha}$ linear combinations of 16 which are of $160 - 5\alpha$ bits or less, among which we keep the expected 2^{32} of $160 - 7\alpha + 32$ bits or less
- We want $160 - 7\alpha + 32 \leq 32$, so $\alpha \gtrsim 23$ should suffice
- We can do all of this in time $\widetilde{O}(2^{46})$ and space $O(2^{23})$: much better than the $O(2^{32})$ memory we started with, at the cost moderate increase in computation
- Possible to minimize the length of the first list even further

Complexity estimate: 160 bits

Bias(bit)	Algorithm	#Round	Time	Space
1	S-S	2	$2^{46.3}$	$2^{25.1}$
	S&D	4	$2^{32.8}$	$2^{32.8}$
2	S-S	3	$2^{38.8}$	$2^{21.4}$
	S&D	6	$2^{23.7}$	$2^{23.7}$
3	S-S	3	$2^{32.8}$	$2^{18.4}$
	S&D	7	$2^{20.9}$	$2^{20.9}$
4	S-S	4	$2^{25.5}$	$2^{14.8}$
	S&D	9	$2^{16.9}$	$2^{16.9}$
5	S-S	5	$2^{21.0}$	$2^{12.5}$
	S&D	11	$2^{14.2}$	$2^{14.2}$

Complexity estimate: 256 bits

Bias(bit)	Algorithm	#Round	Time	Space
1	S-S	2	$2^{73.7}$	$2^{38.9}$
	S&D	5	$2^{43.7}$	$2^{43.7}$
2	S-S	3	$2^{52.0}$	$2^{28.0}$
	S&D	6	$2^{37.4}$	$2^{37.4}$
3	S-S	4	$2^{40.3}$	$2^{22.2}$
	S&D	8	$2^{29.3}$	$2^{29.3}$
4	S-S	5	$2^{38.0}$	$2^{21.0}$
	S&D	10	$2^{24.2}$	$2^{24.2}$
5	S-S	6	$2^{38.0}$	$2^{21.0}$
	S&D	12	$2^{21.0}$	$2^{21.0}$

Work in progress

- Use this approach to mount the 2-bit bias fault attack on qDSA
- Need for large-scale parallelization:
  - not so easy with direct Schroeppel–Shamir (due to heaps)
  - use a simple trick of Howgrave-Graham and Joux to parallelize
  - + some systems design
- More refinements
  - can improve the attack with more data (keep signatures with small h_i's!)
  - can improve the attack with adaptive signature queries (Nikolic–Sasaki, AC'15)
  - asymptotically, can use Generalized Birthday algorithms with more than 4-way collisions

Thank you!

Dank je!