MoSP presentation

Generating backdoored EC curves

March 15, 2020

Project description

- EC cryptography is all over the place
- There are reasonable doubts about "verified-ness" of several named NIST curves
- Let's try to build a parametrized curve generator!

EC quick intro

$$y^2 = x^3 + ax + b$$

$$x, y \in \mathbb{F}_p \text{ or } x, y \in \mathbb{F}_{2^p}$$

k - private key, kG - public key

(a, b, p, G, n, h) - EC params

Project goal

- Build a EC parameter generation tool, which would produce parameters that are:
- Statistically indistinguishable from “normal” parameters
- Allow for relative ease of “cracking” (which can mean decrypting, signing, etc.)

Planned tasks

- Build a simple degenerate curve
- Try to brute-crack it
- Build a more complex weak curve
- Also crack it
- Combine multiple approaches to allow for a final complex curve
- Review the practical implications

Methodology

- There are a plethora of well-known methods to achieve the goal
- It's just that there are no implementations
- So, the general plan for each subtask is:
  - Find a paper
  - Try to implement it manually

Already implemented

- Build a simple degenerate curve
- Try to brute-crack it
- Build a more complex weak curve
- Also crack it
- Combine multiple approaches to allow for a final complex curve
- Review the practical implications

Already implemented

- Linear crack
- Pollard's rho crack
- 24-bit curve generator
- Degenerate 192-bit curve generator

Degenerate 128-bit curve generator

- Take a known curve
- Replace generator with something less interesting
- E.g. take `secp192k1`, and replace generator with $(0, 2)$

Degenerate 192-bit curve generator

```
$ openssl ecparam -in sec_params.pem -text -noout
Field Type: prime-field
Prime:
    00:ff:ff:ff:ff:ff:ff:ff:ff:ff:ff:ff:ff:ff:ff:ff:
    ff:ff:ff:ff:ff:fe:ff:ff:ee:37
A:    0
B:    3 (0x3)
Generator (uncompressed):
    04:db:4f:f1:0e:c0:57:e9:ae:26:b0:7d:02:80:b7:
    f4:34:1d:a5:d1:b1:ea:e0:6c:7d:9b:2f:2f:6d:9c:
    56:28:a7:84:41:63:d0:15:be:86:34:40:82:aa:88:
    d9:5e:2f:9d
Order:
    00:ff:ff:ff:ff:ff:ff:ff:ff:ff:ff:fe:26:f2:
    fc:17:0f:69:46:6a:74:de:fd:8d
Cofactor:  1 (0x1)
```

Degenerate 128-bit curve

```
$ openssl ecparam -in deg_params.pem -text -noout
Field Type: prime-field
Prime:
    00:ff:ff:ff:ff:ff:ff:ff:ff:ff:ff:ff:ff:ff:ff:ff:
    ff:ff:ff:ff:ff:fe:ff:ff:ee:37
A:    0
B:    4 (0x4)
Generator (uncompressed):
    04:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:
    00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:
    00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:
    00:00:00:02
Order:
    00:ff:ff:ff:ff:ff:ff:ff:ff:ff:ff:fe:26:f2:
    fc:17:0f:69:46:6a:74:de:fd:8d
Cofactor:  1 (0x1)
```

24-bit curve

- Just a usual small Koblitz curve
- (a, b) are taken as in `secp192k1`
- Generator is random (random as in "I have rolled the dice beforehand")

24-bit curve

```
$ openssl ecparam -in 24bit_params.pem -text
Field Type: prime-field
Prime: 16777213 (0xfffffd)
A:    0
B:    3 (0x3)
Generator (uncompressed):
    1125899923619842 (0x4000001000002)
Order:  2796306 (0x2aab12)
Cofactor:  1 (0x1)
```

Linear crack

- Take a number
- Do a linear search
- Try to use each as a private key
- Complexity: $\mathcal{O}(N)$ in the key space

Linear crack

```
$ bin/crack_ec_linear ../files/deg_key.pub.pem
[+] Trying to load ../files/deg_key.pub.pem
    as a public key
[+] Everything loaded successfully, trying
    to linearly brute the params
[+] Hey, found the private key!
[+] private_key: 02
```

Linear crack

```
$ openssl ec -in ../files/deg_key.pem -text -noout
read EC key
Private-Key: (192 bit)
priv:
    0b:2e:88:2d:12:78:ac:07:5c:af:38:54:53:74:25:
    e8:3d:bf:b6:a5:f8:3c:95:f4
pub:
    04:00:00:00:00:00:00:00:00:00:00:00:00:00:00:
    00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:
    00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:
    00:00:00:02
Field Type: prime-field
Prime:
    00:ff:ff:ff:ff:ff:ff:ff:ff:ff:ff:ff:ff:ff:ff:
    ff:ff:ff:ff:ff:fe:ff:ff:ee:37
...
```

Linear crack

```
$ bin/crack_ec_linear ../files/24bit_key.pub.pem
[+] Trying to load ../files/24bit_key.pem
    as a public key
[+] Everything loaded successfully, trying
    to linearly brute the params
[*] Currently at 2710
<snip>
[*] Currently at 26E8F0
[+] Hey, found the private key!
[+] private_key: 2709D9
```

Linear crack

```
$ time bin/crack_ec_linear 24bit_key.pub.pem
[+] Trying to load ../files/24bit_key.pem
    as a public key
[+] Everything loaded successfully, trying
    to linearly brute the params
[*] Currently at 2710
<snip>
[*] Currently at 26E8F0
[+] Hey, found the private key!
[+] private_key: 2709D9
bin/crack_ec_linear ../files/24bit_key.pub.pem
 133.32s user 0.81s system 99% cpu 2:15.06 total
```

Pollard's rho crack

- Basically a baby-step-giant-step with optimizations
- Take an element of the group, then do some stuff until we arrive
- Probabilistic
- Complexity (with a high probability): $\mathcal{O}(\sqrt{N})$

Pollard's rho crack

```
$ time bin/crack_ec_pollard 24bit_key.pub.pem
[+] Trying to load ../files/24bit_key.pub.pem
    as a public key
[+] Everything loaded ok, trying to brute the
    key using Pollard's rho
[+] Found the private key!
[+] private_key: 13500618
[+] Generating and writing into the output file
bin/crack_ec_pollard 24bit_key.pub.pem
 0.07s user 0.00s system 89% cpu 0.077 total
```

Pollard's rho crack

```
$ openssl ec -in 24bit_key.pem -text -noout
read EC key
Private-Key: (22 bit)
priv: 2558425 (0x2709d9)
pub:  1320030616208987 (0x4b08f93f8c65b)
Field Type: prime-field
Prime: 16777213 (0xfffffd)
A:    0
B:    3 (0x3)
Generator (uncompressed):
    1125899923619842 (0x4000001000002)
Order:  2796306 (0x2aab12)
Cofactor:  1 (0x1)
```

Pollard's rho crack

```
$ openssl ec -in 24bit_cracked_key -text -noout
read EC key
Private-Key: (22 bit)
priv: 13500618 (0xce00ca)
pub:  1320030616208987 (0x4b08f93f8c65b)
Field Type: prime-field
Prime: 16777213 (0xfffffd)
A:    0
B:    3 (0x3)
Generator (uncompressed):
    1125899923619842 (0x4000001000002)
Order:  2796306 (0x2aab12)
Cofactor:  1 (0x1)
```

Problems

- Why so slow?
- The problem is lack of actual tooling to do that
- The only actually good API - OpenSSL
- OpenSSL uses C
- C is hard sometimes