Iteration III report

Leonid Lygin

Contents

1 Intro

- **Student name**: Leonid Lygin.
- **Topic**: "nice" EC param design.
- **Supervisor**: Phillip Braun.
- **Iteration number**: 3.

2 Initial plan

Initial plan was to (quoting):

> Refine the parameter generator, try some well-known attacks from wikipedia.

3 Actual work done

Refining the parameter generator was a bit lengthy, the only implemented well-known attacks were *linear bruteforce* and *Pollard's rho for logarithms*.

This step took a long time, because there are no freely available cracking tools for elliptic curves, and implementing even a general-purpose Pollard's rho is relatively time-consuming in C (for performance reasons).

Parameter generator was also rewritten into C, because that's where the OpenSSL API could provide proper support and hours would not be wasted on debugging why doesn't OpenSSL understand anything about the PEM.

So, to conclude, these are three things which were implemented in this iteration: linear crack tool, Pollard cracking tool, parameter generation tool, and here are their explanations a bit more in-depth:

3.1 Linear crack

As we know, a private key is just an element of the field, on which the curve resides (well, basically a number in this case). That means that a "viable" (in a loose sense) method would be linear search starting from 0 and building up until the p from $\mathbb{F}_p$.

The tool in the github repository at `c_tools/src/crack_ec_linear.c` does just that, you can feed it the private key (for the PoC purposes that works, I guess, my python generator did not generate parameters well for some reason), and then it will print the private key, when it finds it.

3.2 Pollard's rho crack

This is basically a baby-step-giant-step algorithm to find discrete logarithms in groups. Here's a quick rundown on how it works:

- We want to find a scalar k, such that $kP = Q$.
- Let's initialize a point A_0 inside the group (meaning on our elliptic curve in this particular case), equal to the P from the first point.
- Define:

$$A_{i+1} = \begin{cases} A_i + P, & \text{if } A_i \in S_0 \\ A_i + Q, & \text{if } A_i \in S_1 \\ A_i + A_i, & \text{if } A_i \in S_2 \end{cases}$$

  Where S_i is an arbitrary almost-equal division of the group.
- When some A_i and A_j are equal, the private key can be trivially deduced using egcd.

The tool in the github repository at `c_tools/src/crack_ec_pollard.c` does just that, you can feed it the public key, and then it will write out the private key, when it finds it.

3.3 Parameter generator

This time, the parameter generator is hard-coded to produce a random (well, "random" as in I have rolled the dice previously) curve with a 24-bit private key.

That tool has some hardcoded parameters, but they are all pretty arbitrary, they can be tweaked, and the target order will be calculated.

The tool in the github repository is at `c_tools/src/generate_24bit_params.c`, when it is ran, it produces an output file with the specified parameters in PEM format.

4 References to results

All results can be found in the github repo: https://github.com/ionagamed/iu-bs3-project.

5 Current issues

All presented attacks worked on the previous degenerate curve, and worked on new 24-bit curves, so there's that, but also there still were no really well-known wikipedia-grade weak curves in this iteration

6 Plan for the next iteration

Finally do the wikipedia-grade curves, and maybe look into practical applications, where can we really use these params, what can we do.