Google CTF - Crypto Backdoor

Carl Löndahl

June 20, 2017

0.1 Basic theory

Problem 1 (Elliptic curve discrete logarithm (ECDLP))**.** *Let E be an elliptic curve over $\mathbb{Z}/\mathbb{Z}_p$. Given points on a curve $G, H \in E$ where G is a generator, find and integer k such that $k \times G = H$.*

Defintion 1 (Chinese Remainder Theorem)**.**

0.2 Problem

This is an ECDH problem. We are given an elliptic curve E, a generator G and two points $A, B \in E$ which correspond to the public variables in a Diffie-Hellman key-exchange protocol. The task is to find the solution to the ECLDP of either A or B. If we can find k such that $k \times G = A$, then we simply have to multiply B with k to get the shared secret $C = k \times B$ and compute `encrypted_message` $\oplus\, C$ to obtain the flag.

0.3 The curve E

0.3.1 Operations $(+, \times)$

Given two points on the curve $A, B \in E$, we perform addition as follows:

```
def add(a, b, p):
  if a == -1:
    return b
  if b == -1:
    return a
  x1, y1 = a
  x2, y2 = b
  x3 = ((x1*x2 - x1*y2 - x2*y1 + 2*y1*y2) \
       * modinv(x1 + x2 - y1 - y2 - 1, p)) % p
```

```
  y3 = ((y1*y2)*modinv(x1 + x2 - y1 - y2 - 1, p)) % p
  return (x3, y3)
```

Multiplication of a point $A \in E$ with a scalar b is defined using addition, i.e.,

$$b \times A = \underbrace{A + A + \cdots + A}_{b \text{ times}}.$$

To achieve a computational speed-up, the multiplication is implemented using square-and-multiply type method (double-and-add):

```
def mul(m, g, p):
  r = -1
  while m != 0:
    if m & 1:
      r = add(r, g, p)
    m >>= 1
    g = add(g, g, p)
  return r
```

0.4 Our curve

The underlying modulus of the curve E is

$$p = 6063413719011923544702597030763287169922463176938122380452864 63.$$

We quickly notice that p is not prime. In fact, it is a rather smooth number consisting of smaller primes. The factors are as follows:

$$\begin{aligned} p \;=\; & 901236131 \cdot 901236131 \cdot 921236161 \cdot 931235651 \cdot \\ & 941236273 \cdot 951236179 \cdot 961236149 \end{aligned}$$

This means that we can solve the ECDLP for E problem in smaller subgroups and then use approriate methods to reconstruct the full solution. This is interesting because the subgroup problem is much easier. For a subgroup E' over $\mathbb{Z}/\mathbb{Z}_{p'}$, the complexity of finding a solution is $\mathcal{O}(\sqrt{p'})$. Being approximately 2^{15} addition operations, this is well within the limits of a modern consumer-grade laptop.

The above computation can be performed using a very simple algorithm called *baby-step giant-step* (BSGS). BSGS is applicable to a lot of problems, thereamong discrete logarithm. It can be summarized as follows. Take a number $m \in \mathbb{Z}$ and compute $i \times (m \times G)$ for all positive integers $i < \text{order}(E)/m$ and put in a table $T : E \to \mathbb{Z}_p$.

Example 1 (BSGS)**.** *Let the curve E be an elliptic curver over $\mathbb{Z}/\mathbb{Z}_{13}$. We are given a generator $G = (3, 2)$ and a point $A = (11, 10)$ and we wish to find an integer k such that $k \times G = A$. In our small example, let $m = 3$. We have bolded the numbers that are relevant.*

i	1	2	**3**	4	5	**6**
	$(3, 2)$	$(5, 4)$	$(9, 8)$	$(4, 3)$	$(7, 6)$	$(0, 12)$
i	7	8	**9**	10	11	**1**2
	$(12, 11)$	$(10, 9)$	$(6, 5)$	$(11, 10)$	$(8, 7)$	$(2, 1)$

So the algorithm computes the table

i	**3**	**6**	**9**	**1**2
	$(9, 8)$	$(0, 12)$	$(6, 5)$	$(2, 1)$

indexed by the points. If we lookup $T[(2, 1)]$ it will give us 12*. This is called the giant step. The next step is the baby step, which computes $A, A+G, A+2\times G, \ldots$, which is $k \times G, (k+1) \times G, (k+2) \times G$. Each time it computes a new value, it looks in the table. Let us simulate the execution. $A = (10, 9)$ is not present in the table, so we compute $A + G = (8, 7)$. This is not present in the table either, so we move on. $A + 2 \times G = (2, 1)$ is however, returning the value* 12*. We know that we moved two steps, so $k = 12 - 2 = 10$. And we have found the solution, i.e., $10 \times G = (11, 10)$.*

An example implementation of BSGS in Python, which accumulates found solutions, is as follows:

```
def baby_step_giant_step(A, g, p):
    m = int((math.sqrt(p))+1)
    T = {}

    mapping = lambda (x,y) : x * p + y

    # Giant steps
    mG = mul(m, g, p)
    Z = mG
    for j in range(1, m):
        if mapping(Z) in T:
            T[mapping(Z)] += [j * m]
        else:
            T[mapping(Z)] = [j * m]
        Z = add(mG, Z, p)
```

```
# Baby steps
solutions = []
Z = A
for j in range(0, m):
    if mapping(Z) in T:
        solutions += [(q-j) % p for q in T[mapping(Z)]]
    Z = add(Z, g, p)

return solutions
```

Since we perform $m + \text{order}(E)/m$ operations, the optimum value is $m = \sqrt{\text{order}(E)}$. Then, the number of operations is $2\sqrt{\text{order}(E)}$ point additions.

To choose our parameters *properly*, we need to determine the order of the curve. In particular, the order of the generator G is the relevant here. This is a bit involved, so we can take a shortcut here. If we assume that for a subgroup with elements on E' has order $p' - 1$, we might get several candidates. For instance, if G has order $(p' - 1)/2$, then running BSGS up to $p' - 1$ will give two candidates.

Example 2. *If $G = (5, 2)$, then the elements of E' will be the following table*

i	1	2	3	4	5	6
	$(5, 4)$	$(4, 3)$	$(0, 12)$	$(10, 9)$	$(11, 10)$	$(2, 1)$
i	7	8	9	10	11	12
	$(5, 4)$	$(4, 3)$	$(0, 12)$	$(10, 9)$	$(11, 10)$	$(2, 1)$

For instance, $(0, 12)$ would have solutions 3 and 9. Note that the order is $(p' - 1)/2 = 6$, and that the solutions differ by that number. In particular, $(p' - 1)/j$ is the order of G, where j the multiplicity of solutions.

For $p_1 = 901236131$, we get the generator $G_1 = (616163248, 505799793)$ and the point $A_1 = (362075264, 381230257)$. Running with BSGS takes only a fraction of a second using `pypy`. It finds that $418335728 \times G_1 = A_1$. For $p_2 = 911236121$, we get the generator $G = (666529991, 829667411)$ and the point $A_2 = (241079607, 464805773)$. We find that that $552714778 \times G_2 = A_2$ and that $97096718 \times G_2 = A_2$. Clearly, the generator has not order $p_2 - 1$ under this modulus.

We will now move on the reconstruction part. The problem we want to solve is

$$\begin{cases} k_1 \times G_1 = A_1 \pmod{p_1} \\ k_2 \times G_2 = A_2 \pmod{p_2} \\ \qquad\qquad \vdots \\ k_7 \times G_7 = A_7 \pmod{p_7} \end{cases}$$

First we notice that $k_1 \times G_1 = (k_1 + \phi(p_1)) \times G_1$, where ϕ is the totient function. In particular, it holds that if $k_1 = k_1' \pmod{p_1 - 1}$, then $k_1 \times G_1 = k_1' \times G$. This is exactly what we saw earlier in Example 2, i.e., that $3 = 9 \pmod 6$ implies $3 \times (5, 2) = 9 \times (5, 2) = (0, 12)$.

We want to find a k such that

$$\begin{cases} k = k_1 \pmod{p_1 - 1} \\ k = k_2 \pmod{p_2 - 1} \\ \qquad \vdots \\ k = k_7 \pmod{p_7 - 1}. \end{cases}$$

This is a direct application of CRT.

Example 3. *Let us define a helper function* `pointmod`*:*

```
pointmod = lambda x, p : (x[0] % p, x[1] % p)
```

In `Sage`*, we may run*

```
crt([418335728,552714778],[901236131-1,911236121-1])
```

which outputs `4175417495028298`*. If we are right, this will be k such that $k \times G$ over $\mathbb{Z}/\mathbb{Z}_{p_1 \cdot p_2}$. We can try it out:*

```
mul(4175417495028298, g, 901236131 * 911236121) \
== pointmod(A, 901236131 * 911236121)
```

which ouputs `True`*. The other solution also evaluates to* `True`*:*

```
mul(45237363210279078, g, 901236131 * 911236121) \
== pointmod(A, 901236131 * 911236121)
```

In the above example, we have show that

$$\begin{cases} k = k_1 \pmod{p_1 - 1} \\ k = k_2 \pmod{p_2 - 1} \end{cases} \implies \begin{cases} k \times G_1 = A_1 \pmod{p_1} \\ k \times G_2 = A_2 \pmod{p_2} \end{cases}$$

so

$$k \times G' = A' \pmod{p_1 \cdot p_2}.$$

```
factors = [901236131, 911236121, 921236161, 931235651, \
             941236273, 951236179, 961236149]
remainders = [(baby_step_giant_step(A, g, fact)) for fact in factors]
[[418335728],
[552714778, 97096718],
[331747054, 700241518, 147499822, 515994286, 884488750],
[749957078],
[916005214, 445387078],
[468722272], [793852246]]
```