Lecture 6

Sunday, 25 September 2022 13:59

BSGS

$P_A = aP$ target, find a

output: i, j with

$P_A + j\,mP = i\,P$ and

$0 \le i < m$, $0 \le j \le m+1$

DL is $a \equiv i - jm \bmod N$

Such i and j exist because $0 \le a < N$ and we can write

$a = a_0 + m a_1$ in base m

$m = \lfloor \sqrt{n} \rfloor$

hence a_1 is no larger than $m+1$

BSGS takes $m-1$ additions for the BS
& 1 addition per GS, of which we will have at most $m-1$.

this is an asymptotic statement, saying that the ratio grows faster than $c\sqrt{N}$ for some C. Thus, $\text{cost}(N) \le c\sqrt{N}$ for $N \ge N_0$ for some N_0.

BSGS is a deterministic algorithm that breaks the DLP in $O(\sqrt{N})$ steps

We want our attacker to take time $\ge 2^{128}$, so choose $N \approx 2^{256}$ (maybe a bit smaller to account for cost of addition)

For BSGS we can be more precise: worst-case $2\sqrt{N}$ and average case $\frac{3}{2}\sqrt{N}$ additions, and $\sqrt{N}$ storage

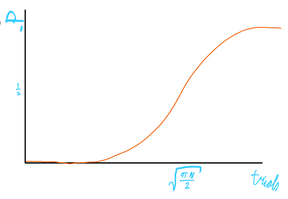
want: less storage, same computational complexity ; let's say we're OK with a probabilistic algorithm

Birthday paradox

Given N numbered balls in a box

after $\sqrt{\frac{\pi N}{2}}$ tries of taking out a ball and putting it back,
we have a 50% chance of seeing one ball twice.

Probability grows like this



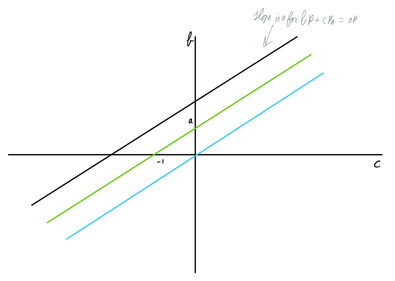
Assume we have a pseudo-random (behaves as random) way of moving in $\langle P \rangle$ (cycle group generated by P) so that I know

that $f(W_i) = W_{i+1} = b_{i+1}P + c_{i+1}P_A$ then I expect to step on the same point twice after $\sqrt{\frac{\pi N}{2}}$ steps. We know $W_i = W_j$ and hopefully $c_i \ne c_j$ & $b_i \ne b_j$

random walk function

note: there is a lot of redundancy: $b_i P + c_i P_A = (b_i + a)P + (c_i - 1)P_A$
because $P_A = aP$

slope is a for $b_i P + c_i P_A = aP$



line determines the colliding point; there are N points per line

$W_i = W_j$ means $b_i P + c_i \underbrace{P_A}_{=aP} = b_j P + c_j \underbrace{P_A}_{=aP}$

$(c_i - c_j)\,aP = (b_j - b_i)P$

$a \equiv \frac{b_j - b_i}{c_i - c_j} \bmod N$

Pollard's rho method $\longrightarrow$ find the cycle; issue: how to get through the tail?

resembles the Greek letter rho

How to detect collision without storing all the points?

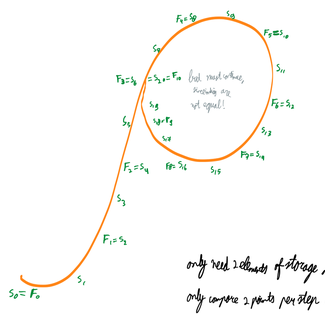
Floyd's cycle-finding method

detect being in a cycle; accept some delay in finding out

walk walks, slow walk S_i and fast walk F_i with $S_i = W_i$ (only change in notation) and $F_i = W_{2i}$ (twice the speed)

$F_0 = S_0 = W_0$, $S_1 = f(S_0)$
$F_1 = f(f(F_0))$ $\Big\} \to S_2 = f(S_1)$ this equals F_1, but walk 1 has a useful collision.

We only compare F_i and S_i, i.e. at the same index



only need 2 elements of storage,
only compare 2 points per step. F_i goes around the cycle at least once, so on average $2.5\sqrt{\frac{\pi N}{8}}$
S_i goes tail and half the cycle on average, so $1.5\sqrt{\frac{\pi N}{8}}$

In total $4\sqrt{\frac{\pi N}{8}} = 2\sqrt{\frac{\pi N}{2}}$, hence we need twice as many steps to find the collision with Floyd, but don't need storage.

Starting point is chosen randomly

How to use many computers?

So far: pick different starting points

Given T computers, this gives $\sqrt{T}$ speedup at cost T

Van Oorschot & Wiener

get a speedup of T with T computers, by making walks shorter, find collisions using distinguished points

points are distinguished points if they meet some conditions; e.g. if the ... least significant bits are 0.

Store starting point and distinguished point when walk comes across distinguished points

lucky case: two walks end in same distinguished point

Each computer picks a random starting point W_{i_0} (with b_i, c_i known)
and computes $W_{i+1} = f(W_i)$ until hitting a distinguished point (Typically defined as having the last ℓ bits of a certain $= 0$; tune ℓ to balance length of walks & storage)

Then report W_{i_0} and final distinguished point to main server. Server finds collision of distinguished points, then recall those walks to find c_i, c_j, b_i, b_j of the collision

What if?

Easy case: take coords of W and compute b_i & c_i as integers in $[0, N-1]$

This costs 2 scalar multiplications per step; actually 1 double-scalar multiplication (store doublings and add one of P, P_A and $P+P_A$ per step)