Cryptanalysis Lec 6 (Apr 18 '2024)

Last Lecture:

Coppersmith's Attack : Infineon.

"optimized KeyGen" $\Rightarrow$ Broken Security.

Today:

* How hard is Discrete Log?

(1) Generic attacks:

(2) DLog in Integer groups.

The Dlog problem:

Given a cyclic group G of order p (prime), generated by g, and

Given a uniformly random group element $h \in G$, find $x \in \mathbb{Z}_p$ s.t. $h = g^x$

i.e. find Dlog of h, in base g (generator of G).

$$\left(G = \langle g \rangle = \{ g^0, g^1, \ldots, g^{p-1} \} \right)$$

(notation!)

Fundamental assumption in Crypto:
$\exists$ groups where DLog is "Hard", i.e.
$\forall$ efficient adversaries A,

$$\Pr\left(A(g^x) = x \; : \; \begin{array}{l} G, g, p \leftarrow \text{Setup}(1^\lambda) \\ x \xleftarrow{\$} \mathbb{Z}_p \end{array}\right) \leq \text{negl}(\lambda)$$

p: $\exp(\lambda)$

"no efficient adv. can solve DLog with non-negl. prob."

e.g. DDH hardness relies on DLog Hardness

But, is DLog actually hard for the groups used in practice?

We'd like concrete security bounds!

Lets see some generic algorithms to solve DLog:

Can be used to solve DLog in ANY cyclic group.

1. (Warm up) Naive algorithm:-

Brute force: Try all possible values of x:

$$\left[\begin{array}{l} \text{For } i = 0, 1, \ldots, p-1, \\ \quad \text{if } h = g^i, \text{ O/P } i. \end{array}\right]$$

Time: p ← assuming exponentiation is $O(1)$

Space: $O(1)$

Success Probability: 1

(Finds DLog for all possible values of h)

We can make this faster by trading-off success probability:

[for $i = 0, 1, \ldots, t-1$,
if $h = g^i$, o/p i]

Time: t

Space: $O(1)$

Success Probability: $\frac{t}{p}$

will only work for $h \in \{g^0, \ldots, g^{t-1}\}$.

2. Collision - Based:- (Skipped in class)

Recall Pedersen commitments:

$\text{Com}(m)$: $r \leftarrow\$ \mathbb{Z}_p$, o/p $g^m h^r$.

we proved that an adversary that breaks binding, solves dlog of h.

Lets use this to build an algorithm:

$M = \perp$: key-value map
For $i = 1, 2, \ldots, t$:
sample $m_i, r_i \leftarrow\$ \mathbb{Z}_p$,

If $g^{m_i} h^{r_i} \in M$, i.e. $g^{m_i} h^{r_i} = g^{m_j} h^{r_j}$
and $m_i \neq m_j$ for some $j < i$,

O/P $\dfrac{(r_i - r_j)}{(m_j - m_i)}$.

O.w. $M \leftarrow M \cup \{(g^{m_i} h^{r_i}, (m_i, r_i))\}$

O/P $\perp$.

* If $g^{m_i} h^{r_i} = g^{m_j} h^{r_j}$, then,

$$g^{m_j - m_i} = h^{r_i - r_j}$$

i.e. $h = g^{\frac{m_j - m_i}{r_i - r_j}}$

So, if a collision is found, we O/P the correct DLog.

Time: $\Theta(t)$

Space: $\Theta(t)$: need to store all the $g^{m_i} h^{r_i}$ values.

Success probability:

For each i, Prob. that we find a collision $= \dfrac{(i-1)}{p}$.

Total: $\displaystyle\sum_{i=1}^{t} \frac{(i-1)}{p} \sim \frac{t^2}{p}$

(this is basically the Birthday Bound).

* If we set $t = \Omega(\sqrt{p})$, we can get constant success probability, but then, both time, space will be $\underbrace{\Omega(\sqrt{p})}_{\text{LARGE}}$!!!

LARGE :(

③ Baby-Step Giant-Step Algo:

Let $h = g^x$, where $x \in \{0, 1, 2, \dots, p-1\}$.

Main Idea: $x = \hat{i} * \sqrt{p} + \hat{j}$ for some $\hat{i}, \hat{j}$ in $\{0, 1, \dots, \sqrt{p} - 1\}$

So, we can search for $\hat{i}, \hat{j}$ separately...

<u>Step 1</u>:

"GIANT STEPS"

Compute $\{g^0, g^{\sqrt{p}}, g^{2\sqrt{p}}, \dots g^{i\sqrt{p}}, \dots\}$ for $i \in \{0, \dots \sqrt{p} - 1\}$

& store a map M: key: $\underbrace{g^{i\sqrt{p}}, \text{value}: i\sqrt{p}}$.

<u>Step 2</u>:

Covers all possible values of $\hat{i}$.

"BABY STEPS"

For j in $\{0, 1, \ldots, \sqrt{p} - 1\}$.

If $\frac{h}{g^j}$ is in M,

then, output $j + M[h/g^j]$

O/P $\perp$.

[$\frac{h}{g^j} \in M$ means that $\frac{h}{g^j} = g^{i\sqrt{p}}$ for some i, i.e. $h = g^{j + i\sqrt{p}}$. i.e. we've found $\hat{i}, \hat{j}$ and $x = j + \underbrace{i\sqrt{p}}_{M(h/g^j)}$]

Time: $\underbrace{O(\sqrt{p})}_{\text{step 1}} + \underbrace{O(\sqrt{p})}_{\text{step 2}}$ [Ignoring log factors]

LARGE :(

Space: $O(\sqrt{p})$: need to store M!

Success probability: 1.

bc we know $\hat{i}, \hat{j}$ exist s.t. $x = \hat{j} + \hat{i}\sqrt{p}$.

We can again reduce time by trading-off success probability:

Restrict i, j to $\{0, \ldots, t-1\}$ in steps 1 and 2.

Then, Time: $O(t)$ (Ignoring log factors)

Space: $O(t)$

Success probability: It will work only if $x \in \{0, 1, \ldots, t^2\}$, so, $\frac{t^2}{p}$

(because x is chosen uniformly at random from $\mathbb{Z}_p$)

So far, we've seen 3 different algos:

	Time	Space	Success Prob.
Naive :	t	1	t/p.
Collision-based :	t	t	t^2/p.
BSGS :	t	t	t^2/p.

Can we do better in terms of space?

(4.) Pollard's Rho Algorithm :

★ Idea: Do a random walk in G.
Then find a cycle.

Lets say we sample $a_0, b_0 \leftarrow \mathbb{Z}_p$.

$$u_0 = g^{a_0} h^{b_0} \xrightarrow[\text{transition}]{\text{Random}} u_1 = f(u_0) = g^{a_1} h^{b_1}$$

"Random walk in G"
Using a function f ...

$$u_2 = f(u_1) = g^{a_2} h^{a_2}$$

$$u_t = f(u_{t-1}) = g^{a_t} h^{b_t}.$$

If we find a cycle, i.e. i, j s.t.

$$u_i = u_j \text{ , i.e. } g^{a_i} h^{b_i} = g^{a_j} h^{b_j}$$

then,
$$x = \frac{a_j - a_i}{b_i - b_j}$$

(This is a different take on the collision finding algo.)

For this algo, we need :

1) A "random" looking function f.

2.) An efficient cycle-finding algo.

Lets start with 2.) :

Naive : Just store $u_0, u_1, \ldots$ in a map and check if $u_i = u_j$ for some $j < i$

(similar to collision-finding algo).

Issue : This uses too much space !

<u>SOLN</u> : Floyd's tortoise & hare algo. :

$O(1)$ space !!

Given : a sequence $x_1, x_2 = f(x_1), x_3 = f(x_2) \ldots$

Find a collision, i.e. $i > j$ s.t. $x_i = x_j$.

$\Rightarrow$ Find a cycle : $x_{i+1} = x_{j+1}$, and so on.

Lets say the sequence has a cycle of length c :-

$$x_1 \xrightarrow{f} x_2 \xrightarrow{f} \cdots \xrightarrow{f} x_\ell \xrightarrow{f} x_{\ell+1} \xrightarrow{f} x_{\ell+2} \cdots x_{\ell+c-1} \xrightarrow{f} x_\ell$$

$[f(x_{\ell+c-1}) = x_\ell]$

i.e. $x_{\ell+c} = x_\ell = x_{\ell+2c} \ldots$

In general, $x_{\ell+i} = x_{\ell+i+c \cdot k}$ $\forall i \geq 0, k \geq 0$.

Specifically, Let $\hat{i}$ be the smallest index $\geq l$ that is a multiple of c.

i.e. $\hat{i} = k \cdot c \geq l$ for some k.

Then, $x_{\hat{i}} = x_{\hat{i}+c} = \cdots\cdots = x_{\hat{i}+c\cdot k}$

$\Rightarrow \underbrace{x_{\hat{i}} = x_{2\hat{i}}}$.

This is the collision that Floyd's algo finds, in LOW SPACE.

Formally:

Let $t_0 = x_0 = h_0$.

For $i = 1, 2, \ldots$:

$t_i = f(t_{i-1})$ ← Tortoise

$h_i = f(f(h_{i-1}))$ ← Hare

← collision found!

If $t_i = h_i$: o/p (t_i, h_i, i)

For $i=1$: comparing x_1 vs x_2.

For $i=2$: " x_2 vs x_4 and so on...

Space: $O(1)$: Just 2 pointers !!!

This finds a cycle if it exists.

Time: $O($length of sequence$)$

Success Probability:

But, what is the probability that the sequence $u_0, u_1, u_2, \ldots$

has a cycle?

If f is pseudo-random, then,

By Birthday Bound, (Same as in analysis of collision finding algo)

$\{u_0, u_1, \ldots, u_t\}$

has collision probability $\sim \frac{t^2}{p}$.

so, assuming f is pseudo-random, we get

Time: $O(t)$

Space: $\boxed{O(1)}$ YAY!

Success Probability: $\sim \frac{t^2}{p}$

Back to 1.) A pseudo-random f:

Option (i): use a Hash function? i.e. $f = H$

i.e. $u_i = H(u_{i-1})$

$u_i = g^{a_i} h^{b_i}$

$H(u_{i-1}) \rightarrow g^{a_{i-1}} * h^{a_{i-1}}$

Issue :

we won't know a_i, b_i . . .
(But we need them for computing DLog!)

Solution : split G into random ✶ disjoint subsets : → Important for f to look random.

$$G = S_0 \cup S_g \cup S_h$$

$$f(u = g^a h^b) = \begin{cases} g^{2a} h^{2b} & \text{if } u \in S_0 \\ g^{a+1} h^{b} & \text{if } u \in S_g \\ g^{a} h^{b+1} & \text{if } u \in S_h \end{cases}$$

Putting 1.) and 2.) together :

Sample $a_0, b_0 \xleftarrow{\$} \mathbb{Z}_p$. Let $u_0 = g^{a_0} h^{b_0}$ → Tortoise

and $v_0 = u_0$. → Hare

For $i = 1, 2, \ldots t$:

t_i : $(a_i, b_i, u_i) = f(u_{i-1})$

h_i : $(a_{2i}, b_{2i}, v_i) = f(f(v_{i-1}))$

Lets assume, f also gives the corresponding a_i, b_i values...

← collision found.

If $u_i = v_i$ and $b_i \neq b_{2i}$:

O/P $\dfrac{(a_i - a_{2i})}{(b_{2i} - b_i)}$

O/P $\perp$.

Time : $O(t)$

Space : $O(1)$

Success Prob. : $\sim \dfrac{t^2}{p}$.

We've seen 4 generic algorithms, 3 have success prob. $\dfrac{t^2}{p}$ for runtime t:

Turns out, this is an upper bound!

Shoup '97 :

The success probability of ANY generic DLog algorithm running in time t is bounded by $\frac{t^2}{p}$.

<u>Corollary</u>: Any generic DLog algo. requires $\Omega(\sqrt{p})$ group operations to get non-negligible success prob.

* So, Random Collision-based BSGS, and Pollard's Rho are <u>all</u> time optimal. (ignoring log factors)

* Pollard's Rho is space optimal!

The bound is based on the Generic Group Model (GGM):

[A model where the adv is not allowed to learn anything about the specific group structure.]

The 4 algos we've seen work for ANY group, e.g. integer groups.

But, we'll now see a non-generic Dlog algorithm, that works only for Integer groups:

5. Index Calculus:

Consider an integer group $\mathbb{Z}_q^*$ for prime q.

Lets say, $q \approx 2^\lambda$ (exponential sized group).

Then, for non-negligible success prob.,
Trivial Algo (1): Time $\sim 2^\lambda$

Algos (2) - (4): Time $\sim \sqrt{2^\lambda} = 2^{\lambda/2}$

$\Rightarrow$ Still exponential time.

Index Calculus: sub-exponential time,

i.e. $\Theta(2^{\lambda^\epsilon})$ for $\epsilon < 1$.

(skipped in class) $(\mathbb{Z}_q, +)$

Warmup: consider group $\mathbb{Z}_q$ with + operation.

Elements: $\{0, 1, \ldots\ldots, q-1\}$.

Group operation: $a, b \rightarrow (a+b) \% q$.

Dlog problem: given $g, h \in \mathbb{Z}_q$, find x s.t. $\underbrace{g+g+\ldots+g}_{x \text{ times}} = h \% q$

Easy to solve! $x = \frac{h}{g} \in \mathbb{Z}_q$.

then, $g+g\ldots+g = h \% q$

Index Calculus: for $(\mathbb{Z}_q^*, *)$.

i.e. group operation: product.

Elements: $\{1, 2, \ldots, q-1\}$.

Assume that $q-1 = 2p$ for prime p.

i.e q is a "safe prime" and

ord(g) is p.

(Note, Dlog will be trivial if ord(g) were 2)

[If q were not safe, Dlog would be less Hard.]

[Updated on Apr 22]

Let $\hat{g}$ be the generator of $\mathbb{Z}_q^*$.

(It exists bc $\mathbb{Z}_q^*$ is a cyclic group for prime q.)

i.e. $\mathbb{Z}_q^* = \langle \hat{g} \rangle$. Let $G = \langle g \rangle \subseteq \mathbb{Z}_q^*$

$|\mathbb{Z}_q^*| = q-1$, $|G| = p = (q-1)/2$.

If $q \sim \exp(\lambda)$, $p \sim \exp(\lambda)$. one would expect DLog to be hard in G.... But, we can use Index calculus :-

I. We'll first solve DLog w.r.t. $\hat{g}$. i.e. given h, find $\mathrm{DLog}_{\hat{g}}\, h$.

II. We can then compute $\mathrm{Log}_g\, h$ as follows:-

$g = (\hat{g})^2$.

so if $h = (\hat{g})^x$ $\quad (x = \log_{\hat{g}} h)$,

then, $h = (\hat{g})^x = (\hat{g}^2)^{x/2}$.

so, $\log_g h = \frac{1}{2} \cdot \log_{\hat{g}} h$

(If $\log_{\hat{g}} h$ is not even, then, h is NOT in the p-sized subgroup.)

we'll now see the algo. to compute $\log_{\hat{g}} h$:-

Dlog algorithm :- (for $(\mathbb{Z}_q^*, *)$)

step 1.) Let $L = \{2, 3, 5, \ldots, p_t\}$ be all primes $\leq B$. ← parameter, to be set later.

L will be our 'factorization basis'.

e.g. for $B = 6$, $L = \{2, 3, 5\}$.

$120 = 2^3 \cdot 3 \cdot 5$ can be factored in L,

$49 = 7^2$ cannot.

A number is B-smooth if all its prime factors are $\leq B$.

i.e. 120 is 6-smooth, 49 is not.

step 2.) Compute $\log_{\hat{g}} p_i$ for all $i \in [t]$.

(we'll see how later) → now, we can find Dlog of ANY B-smooth number.

step 3.)

step 3.1):

$r = 0$

WHILE $h * \hat{g}^r$ is NOT B-SMOOTH:

$r \leftarrow\$ \mathbb{Z}_q$

i.e. FIND a value r s.t.

$h\hat{g}^r$ factors in B, i.e.

$$h\hat{g}^r = \prod_{i \in [t]} p_i^{e_i}$$

$$= 2^{e_1} * 3^{e_2} * 5^{e_3} \cdots * (p_t)^{e_t}$$

Lets take log base g on both sides:-

$$\log_{\hat{g}} h + r = \sum_{i \in [t]} e_i * (\log_{\hat{g}} p_i)$$

$$\Rightarrow \log_{\hat{g}} h = -r + \sum_{i \in [t]} e_i * (\log_{\hat{g}} p_i)$$

(we computed these in Step 2)

Step 3.2) * Factorize $h \cdot \hat{g}^r$ in L, i.e. find e_i.

* Output: $-r + \sum_{i \in [t]} e_i * (\log_{\hat{g}} p_i)$.

How efficient is step (3)?

(i) How many r values do we need to sample until $h \cdot \hat{g}^r$ is smooth?

MATH FACT 1: There are $\frac{N}{u^u}$ B-smooth numbers $\leq N$, where $u = \frac{\log N}{\log B}$.

(N is q for us)

So, $\Pr\left[\begin{array}{c}\text{random number} \\ \text{in } \mathbb{Z}_q^* \text{ is } B\text{-smooth}\end{array}\right] = \frac{1}{u^u}$.

so, in expectation, we'll need to try u^u different values of r.

(ii) How much work do we do for each value of r sampled?

We try to factorize $h \cdot \hat{g}^r$ with basis L, (to check if its B-smooth.)

<u>MATH FACT 2</u>: There are $\frac{B}{\log B}$ primes $\leq B$. $\Rightarrow |L| = \frac{B}{\log B}$. (asymptotic)

so, to factorize $h \cdot \hat{g}^r$ with basis L:

~ Try dividing by each prime in L

$$\tilde{O}\left(\frac{B}{\log B} * \text{polylog}(B)\right)$$

)

primes in L = $\tilde{O}(B)$. Time to divide.

So, total runtime of step (3): $\tilde{O}(n^4 \cdot B)$

Back to step (2):

How to compute $\log_{\hat{g}} p_i$ $\forall p_i \in L$?

Sample r_1 s.t. $\hat{g}^{r_1}$ factors in B.

(e.g. by running a while loop, like in Step (3))

i.e. $\hat{g}^{r_1} = \prod_{i \in [t]} p_i^{e_i^{(1)}}$

$$\Rightarrow \underbrace{r_1}_{\text{Known}} = \sum \underbrace{e_i^{(1)}}_{\text{Known (by factoring } \hat{g}^{r_1} \text{ in L.)}} * \log_{\hat{g}} p_i \quad \ldots (1)$$

Think of each $\log_{\hat{g}} p_i$ as a variable x_i

Then, (1) is a linear equation in $\{x_i\}$.

There are t variables, so if we can get t equations, we could solve for $\{x_i\}$.

Formally:

step (2.1):

For $i \in \{1, 2, \ldots, t\}$:

$r_i = 0$
while $\hat{g}^{r_i}$ does not factor in L:
$r_i \xleftarrow{\$} \mathbb{Z}_q$

$\Rightarrow$ so, we'll get t equations :-

$r_1 = \sum e_i^{(1)} \cdot x_i$ $\left(\text{i.e. } \hat{g}^{r_1} = \prod_{i \in [t]} p_i^{e_i^{(1)}}\right)$

$r_2 = \sum e_i^{(2)} \cdot x_i$ $\left(\text{i.e. } \hat{g}^{r_2} = \prod_{i \in [t]} p_i^{e_i^{(2)}}\right)$

$\vdots$

$r_t = \sum e_i^{(t)} \cdot x_i$ $\left(\text{i.e. } \hat{g}^{r_t} = \prod_{i \in [t]} p_i^{e_i^{(t)}}\right)$

★ Since we sample r_i randomly, w.h.p., we'll get t independent eqns.

Step 2.2: Solve the linear system of eqns!

so we'll get $x_i = \log_{\hat{g}} p_i$ for all $i \in [t]$.

Efficiency of Step 2:-

By Math fact (2), $t = \dfrac{B}{\log B} := |L|$.

For each $i \in [t]$: $\underbrace{u^u}_{\text{\# times we'll sample } r_i \text{ until } \hat{g}^{r_i} \text{ factors in } L} * \Big(\underbrace{\dfrac{B}{\cancel{\log B}} * \cancel{\text{polylog } B}}_{\text{Time to factorize } \hat{g}^{r_i} \text{ in } L}\Big)$ Expected time.

So, step 2.1: $\overbrace{\dfrac{B}{\log B}}^{t} * \overbrace{u^4 * B}^{\text{Time for each } t}$ time.

Step 2.2: solving linear system: $\left(\dfrac{B}{\log B}\right)^3$

So, step 2 total: $< \left(B^3 + \dfrac{u^4 \cdot B^2}{\log B}\right)$

Step 1 + step 2 + step 3:

$< \tilde{O}(B^3 + u^4 \cdot B^2)$ [Ignoring log factors]

$\left(\text{Recall, } u = \dfrac{\log q}{\log B}\right)$

If we set $B = e^{\sqrt{\log q \cdot \log\log q}}$, then, $\log B = \sqrt{\log q \cdot \log\log q}$.

$$u = \frac{\log q}{\log B} = \sqrt{\frac{\log q}{\log\log q}}$$

(skipped in class)

$$u^u = \left(e^{\log u}\right)^u = \exp(u \log u)$$

$$= \exp\left(\sqrt{\frac{\log q}{\log\log q}} \cdot \frac{1}{2}\left(\log\log q \cancel{- \log\log\log q}\right)\right)$$

$$\leq \exp\left(\frac{1}{2}\sqrt{\log q \cdot \log\log q}\right)$$

$$\leq \exp\left(\sqrt{\log q \cdot \log\log q}\right) = B.$$

i.e $u^u \leq B$.

Overall time : $\tilde{O}(B^3)$.

where $B = \exp\left(\sqrt{\log q \cdot \log\log q}\right)$

Notation :

$$L_n(\alpha, c) = \exp\left(c \cdot (\log n)^{\alpha} \cdot (\log\log n)^{1-\alpha}\right).$$

So, Runtime : $\tilde{O}(B^3)$, $B = L_q(1/2, 1)$.

$\Rightarrow L_q\left(\frac{1}{2}, 3\right)$.

If $q = 2^{\lambda}$, then,

the above algo has runtime :

$$\exp\left(3 \cdot \sqrt{\lambda} \cdot \sqrt{\log \lambda}\right) \approx O\left(2^{\text{poly}(\sqrt{\lambda})}\right)$$

$\Rightarrow$ Sub-exponential Algo for DLog in Integer groups !!!

Best known DLog attack : $L_q\left(\frac{1}{3}, 2\right)$

i.e. $\exp\left(2 \cdot (\log q)^{1/3} \cdot (\log\log q)^{2/3}\right).$

Conclusions :

* Sub-exp. DLog algo. for integer groups!

Implication: lets say we want 128-bit security.

i.e. Solving DLog should take time $\geq 2^{128}$.

For integer groups: we'll have to set $q \sim 2^{2048}$ to get ~ 122 bit security!

$$\left(\text{Want } \underbrace{\exp\left(2 (\log q)^{1/3} \cdot (\log \log q)^{2/3}\right)}_{\text{Runtime of best DLog algo}} \approx 2^{122} \right)$$

security level

$\Downarrow$

(This gives $q \sim 2^{2048}$)

i.e. If we want to rely on DLog hardness, and have 128-bit security, we'll have to use VERY LARGE Integer groups!!

☆ WE need to use BETTER DLog groups!!

(i.e where ideally, there is no DLog algorithm better than the generic algorithms.)

(which are exponential time).