Cryptanalysis Lec 6 (Apr 15' 2026)

Last Lecture:

Coppersmith's Attack : Infineon.

"optimized KeyGen" ⇒ Broken Security.

Today:

* How hard is Discrete Log?

(1) 'Generic' attacks: ——— Pollard's Rho

Naive / Baby step, Giant step

(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).

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

Fundamental assumption in Crypto:
∃ groups where DLog is "Hard", i.e.
∀ efficient adversaries A,

$$\Pr\left[A(g^x) = x \;:\; G, g, p \xleftarrow{exp(\lambda)} \text{Setup}(1^\lambda),\ x \xleftarrow{\$} \mathbb{Z}_p\right] \leq \text{negl}(\lambda)$$

"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:

Dont rely on the structure of the group: Can be used to solve DLog in ANY cyclic group.

(1) (Warm up) Naive algorithm:-

Brute force: Try all possible values of x:

[For $i = 0, 1, \ldots, p-1$, if $h = g^i$, O/P i.]

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:

$\text{Com}(m)$: $r \xleftarrow{\$} \mathbb{Z}_p$, 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 $m_i, r_i \xleftarrow{\$} \mathbb{Z}_p$,

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 $\frac{(r_i - r_j)}{(m_j - m_j)}$.

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

O/P 1.

* 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 $= \frac{(i-1)}{p}$.

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

(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 $\Omega(\sqrt{p})$!!!

LARGE :(.

(3.) Baby-Step Giant-Step Algo:

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

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

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

Step 1: "GIANT STEPS"

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

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

Step 2:

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_{-1}.

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

O(p) 1.

[$\frac{h}{g^j} \in M$ 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 $\sqrt{p}$.

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_1. 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_1" 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.

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

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

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

For this algo, we need:

1) A "random" looking function f.
that uses less space

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!

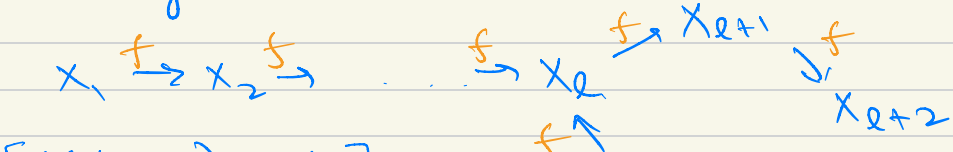
SOLN: Floyd's tortoise & hare algo.:
O(1) space!!

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:-



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

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

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

↳ 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

If $t_i = h_i$: collision found! : dP (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, $\{u_0, u_1, \ldots, u_t\}$ has collision probability $\sim \frac{t^2}{p}$.

(Same as in analysis of collision finding algo)

So, assuming f is pseudo-random, we get

Time: $O(t)$

Space: $O(1)$ 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. $u_i = H(u_{i-1})$

$g^{a_i} h^{b_i}$ $\quad$ $g^{a_{i-1}} * h^{b_{i-1}}$

Issue:

we won't know $a_i, b_i \ldots$

But we need them for computing DLog!

Q: what goes wrong if $(a_i, b_i) = H(a_{i-1}, b_{i-1})$?

Solution: split G_1 into random* disjoint subsets: → important for f to look random.

$$G_1 = S_0 \cup S_g \cup S_h$$

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

Putting 1) and 2) together:

Sample $a_0, b_0 \leftarrow \mathbb{Z}_p$. 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 $\dfrac{(a_i - a_{2i})}{(b_{2i} - b_i)}$.

↑ need denominator to be non-zero

O/P $\perp$.

Time: $O(t)$

Space: $O(1)$

Success Prob. : $\sim O\left(\dfrac{t^2}{p}\right)$

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 $\frac{t^2}{p}$.

Corollary: 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 all 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.
>
> i.e. can only do 'generic' group operations

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 $\epsilon < 1$.

e.g. $2^{\sqrt{\lambda}}$, $2^{\lambda^{1/3}}$ ...

(Skipped in class)

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

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

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

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 a subset of $(\mathbb{Z}_q^*, *)$

i.e. group operation: product.

Elements: $\{1, 2, \ldots, q-1\}$. (This is NOT a prime order group)

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

i.e. q is a "safe prime" and $\text{ord}(g)$ is p. $g^p = 1 \% q$

(Note, DLog will be trivial if $\text{ord}(g)$ were 2)

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

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 $G_1 = \langle g \rangle \subseteq \mathbb{Z}_q^*$

$|\mathbb{Z}_q^*| = q-1$, $|G_1| = p = (q-1)/2$.

If $q \sim \exp(\lambda)$, $p \sim \exp(\lambda)$. one would expect DLog to be hard in G_1.... But, we can use Index Calculus :-

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

II. We can then compute $\text{Log}_g h$ 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.

e.g. consider $\hat{h} = p_1 \cdot p_2$

Let $\log_{\hat{g}} p_i = l_i$. Then,

$\hat{h} = \hat{g}^{l_1} \cdot \hat{g}^{l_2} = \hat{g}^{l_1 + l_2}$, i.e.

$DLog_{\hat{g}}(\hat{h}) = \log_{\hat{g}} p_1 + \log_{\hat{g}} p_2$

But, Recall, we want DLog of h, which may not be B- smooth....

Solution :-

* Multiply h with random group elements $\hat{g}^r$ until $h \cdot \hat{g}^r$ is smooth...

step 3.)

step 3.1): $r = 0$

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

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

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:

(unknown)

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

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

(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 $h\hat{g}^r$ 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, $\Pr\left[\text{random number in } \mathbb{Z}_q^* \text{ is } B\text{-smooth}\right] = \frac{1}{u^u}$.

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

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

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

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

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

~ Try dividing by each prime in L

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

)

# of primes in L = $\tilde{O}(B)$.

Time to divide.

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

Back to step (2):

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

Sample r_1 s.t. $\hat{g}^{r_1}$ is B-smooth

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

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

$\Rightarrow$ $r_1 = \sum e_i^{(1)} * \log_{\hat{g}} p_i$ ... (1)

r_1: known

$e_i^{(1)}$: known (by factoring $\hat{g}^{r_1}$ in L.)

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:

$r_i \overset{+}{\leftarrow} \mathbb{Z}_q$

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

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

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

$$\vdots$$

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

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

<u>step 2.2:</u> 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), $t = \frac{B}{\log B} := |L|$.

For each $i \in [t]$: $u^u * \left(\frac{B}{\log B} * \text{polylog} B \right)$ Expected time.

- u^u: # times we'll sample r_i until $\tilde{g}^{r_i}$ factors in L
- $\frac{B}{\log B} * \text{polylog} B$: Time to factorize $\tilde{g}^{r_i}$ in L.

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

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

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

Step 1 + Step 2 + Step 3:

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

(Recall, $u = \frac{\log q}{\log B}$.)

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)

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

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

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

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

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

$\Rightarrow L_q(1/2, 3)$.

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

the above algo has runtime :

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

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

Best known DLog attack : $L_q(1/3, 2)$

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'd have to set $q \sim 2^{2048}$ to get ~122 bit security!

(Want $\exp\left(2\,(\log q)^{1/3}\cdot(\log\log q)^{2/3}\right) \simeq 2^{122}$ ← security level.)

Runtime of best DLog algo.

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

i.e. If we want to rely on DLog hardness, and have 128-bit security, we'd 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).