Preprocessing Attacks on Symmetric-Key Primitives

Today

* Preprocessing attacks
* Hellman tables (Rainbow tables)
  OWPs, OWFs
* Yao's lower bound
* Open questions

Logistics

- HW5 due on Friday 6/8 at 5pm
  ↳ Do not exceed your late days
- Keep in touch re: crypto
  ↳ Seminar, Security lunch, ...
- Course evals on Axess
  ↳ Your best way to thank us!
  ↳ Your best way to get revenge! :)
  ↳ If you really like the course ...

First, some history...

Up until 60s cryptography had essentially been a military-only field.

(Read Kahn's "Codebreakers" this summer... life advice)
(Also Rhodes "Making of the Atomic Bomb")

In 1969, this changed... what happened?
↳ ATMs! → Non-military need for crypto!

Lucifer cipher from IBM in 1971
↳ Horst Feistel led effort, Hellman overlapped with him there in 1968
Key size $k \in \{48, 64, 128\}$ bits

National Bureau of Standards and DES
- wanted to standardize unclassified encryption scheme
- DES - Variant of Lucifer... NSA wanted 48-bit keys, IBM wanted 64,... so?... 56! ← Only knew this years later
  ↳ standardized March 1975
  ↳ Diffie & Hellman complained about key size 1975-77 at Stanford (!)
  Said that 128-bit keys would be necessary to get "future-proof" security

Today, can crack DES for \$30 (https//crack.sh)
Use 128-bit keys today... moral?

Some people suggested that NSA had a way to break DES faster, but how?
↳ No obvious attack

2^{56} work to do brute force search is a lot!

Hellman: Use <u>precomputation</u>

Idea: Build a data structure of size $\ll 2^{56}$ that lets you break DES in time $\ll 2^{56}$.

↳ NSA builds data structure <u>once</u> at cost $\approx 2^{56}$
Then breaks DES for much less cost.

Life lessons:
1) Great research often "talks to" the world
2) Gödel's incompleteness thms
   ↳ See Aaronson's survey

Preprocessing Attacks

A general notion in crypto...

Makes sense when everyone uses the same crypto primitives (fns, hashes, groups, etc)

Let's focus on problem of inverting functions

$f: [N] \to [N]$ (think of $N \approx 2^{128}$)

Examples?

$f_{AES}(x) := AES(x, \text{"}000\cdots0\text{"}) \| AES(x, \text{"}000\cdots01\text{"})$

↳ Given encryption of two msgs, recover key
key recovery

$f_{SHA}(x) := SHA256(x)$

↳ Given hash of value, find preimage (password cracking)

$f_{PRG}(x) := PRG(x)$

↳ Given output of PRG, find seed

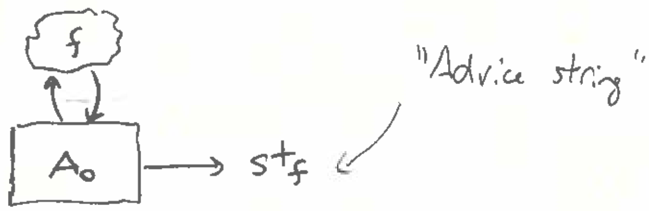
⋮

Almost everything in crypto is about inverting fns!

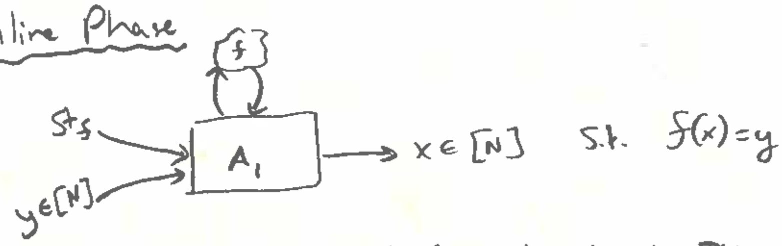
Preprocessing Attack

Function $f: [N] \to [N]$

Preproc Phase



Online Phase



[Think: Do preproc relative to f_{AES}, then break TLS connection in real time]

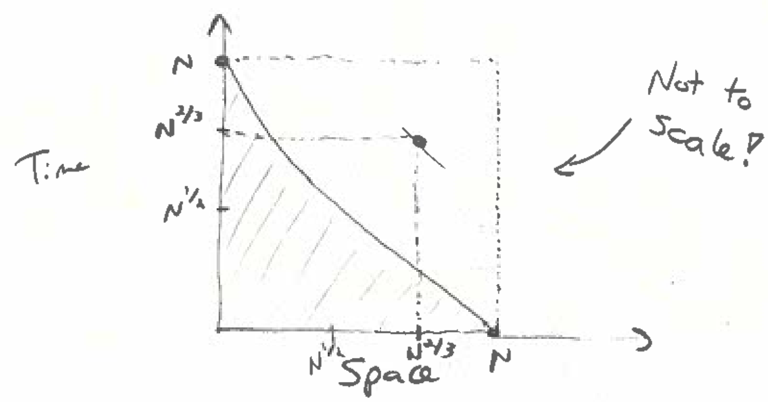
We will be interested in $S = |st_f|$ ("space")

T = # of online f queries ("time")

Two simple ideas?

1) Store $(x, f(x))$ pairs in a look-up table
   ↳ To succeed w/ good prob, need $S = \Omega(N)$.

2) Store nothing, do full attack online
   ↳ Need $T = \Omega(N)$



* Can we do better?
* Magic of Hellman tables is that achieves

$$S = T = \tilde{O}(N^{2/3}) \longleftarrow (2^{56})^{2/3} \approx 2^{38} \text{ time!}$$

With 2^{19} cores, can run in 2^{19} time

$\Rightarrow$ If you want to "crack" passwords, this is the technique that you use!
↳ Not just a theoretical result! [Life lesson: Dangerous to bet against theory in long run]

Warm-Up: Inverting permutation w/ preprocessing

Let $f: [N] \to [N]$ be 1-to-1 / permutation

$G_f = (V, E)$ $\quad V = [N]$

$E = \{ (x, f(x)) \mid x \in [N] \}$

When f is permutation, G_f is union of cycles

In this case, get preproc alg for inverting f w/ space S, time T s.t. $ST = \tilde{O}(N)$.

e.g. $S = T = \tilde{O}(\sqrt{N})$.

Preproc Phase

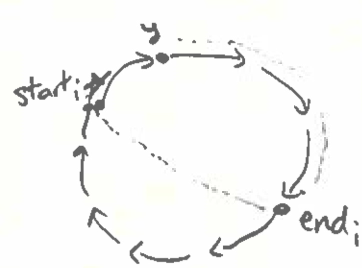
* Divide cycles into length-$\sqrt{N}$ segments
* Store $(\text{start}_i, \text{end}_i)$ pairs of segments.

Space: $\tilde{O}(\sqrt{N})$

Online Phase

* On input $y \in [N]$, iterate f on y until hitting segment endpoint
* Continue iterating from start of seg until hitting y.

Time: $O(\sqrt{N})$



<u>Hellman Tables</u>

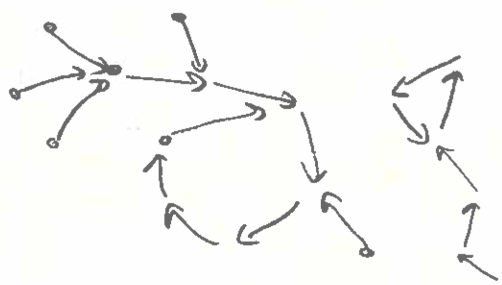
Most crypto interesting fns are <u>not</u> 1-to-1! (See egs)

<u>Thm</u> (Hellman) There is a preproc attack that inverts a <u>random</u> fn $f: [N] \to [N]$ with space $N^{2/3}$ time $N^{2/3}$ (under mild heuristic assump.).

↳ Fiat-Naor make it rigorous.

Problem: Trick for perms doesn't work when f is fn.

Look at G_f:



$\Big\}$ Can't cover G_f with only $\sqrt{N}$ segments of length $\sqrt{N}$!

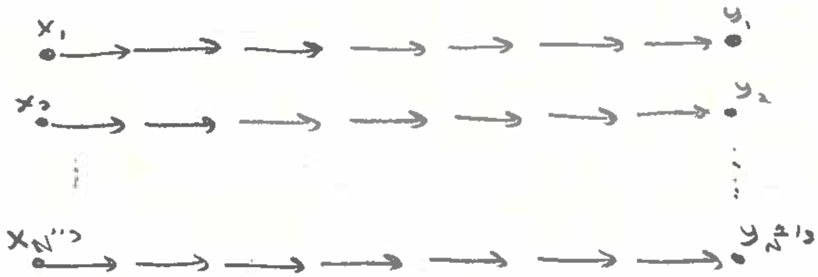
Idea: We <u>can</u> find $N^{1/3}$ segments of length $N^{1/3}$ that are non-overlapping in G_f.

↳ If we apply perm trick now, we will be able to invert

$$\varepsilon = \frac{(N^{1/3})(N^{1/3})}{N} = \frac{1}{N^{1/3}}$$ fraction of points in image of f.

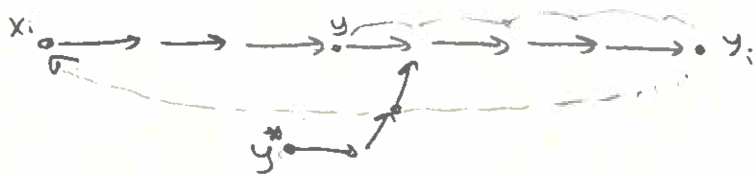
Hellman Table

Preproc: Build $N^{1/3}$ chains of length $N^{1/3}$



Store (x_i, y_i) pairs

Online: On input y



Lucky case

Unlucky case

$\Pr[\text{lucky}] = \dfrac{\#\text{ of points in table}}{\#\text{ points}} \geq \dfrac{\Omega(N^{2/3})}{N}$ ← Claim

Space: $\tilde{O}(N^{1/3})$ Time: $\tilde{O}(N^{1/3})$ $\varepsilon = \dfrac{1}{N^{1/3}}$.

[Ignore time to sort, etc.
Just count queries to f]

Hellman Tables: Analysis

Why does this work?

Claim: Preproc chains cover $\Omega(N^{2/3})$ points in expectation.

Let G_i be event that ith chain is "good" – doesn't collide w/ prior chains

$$\Pr[G_i] = \left(1 - \frac{N^{2/3}}{N}\right)^{N^{1/3}}$$

$$\approx \left(e^{-\frac{N^{2/3}}{N}}\right)^{N^{1/3}}$$

$$\approx 1/e \leftarrow \text{constant}$$

Important life fact: $1+x \le e^x$

For x small $1+x \approx e^x$

$$\mathbb{E}[\#\text{good chains}] = \overbrace{N^{1/3}}^{\#\text{chains}} \cdot \mathbb{E}[G_i] = \Omega(N^{1/3}).$$

$$\mathbb{E}[\#\text{pts covered}] = \underbrace{N^{1/3}}_{\text{Chain length}} \cdot \Omega(N^{1/3}) = \Omega(N^{2/3}).$$

So, if challenge pt $y \in [N]$ is on a good chain, can invert in time $\tilde{O}(N^{1/3})$.

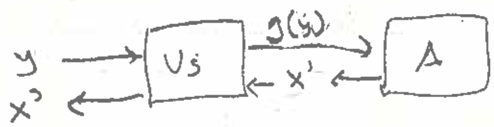
↳ We have shown succ prob $\varepsilon \approx \frac{1}{N^{1/3}}$.

Hellman's Good Idea

Now can invert f w/ $T = S = \frac{1}{\epsilon} \approx N^{1/3}$.
How can we invert f everywhere?

Idea: "Rerandomize f into $f_1, f_2, f_3, \dots, f_{N^{1/3}}$"

Choose random perms $g_1, \dots, g_{N^{1/3}} : [N] \to [N]$
If can invert $g(f(x))$, can invert $f(x)$!



$g(f(x')) = g(y)$
$f(x') = y$

Preproc

Construct $N^{1/3}$ Hellman tables, one for each g_i

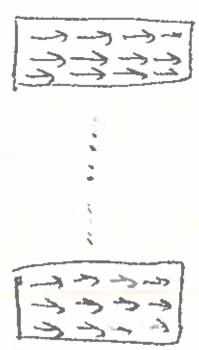
Space: $\tilde{O}(N^{2/3}) \leftarrow N^{1/3}$ tables, size-$N^{1/3}$ each

Online

Try to invert $f(x)$ using each table in sequence
↳ Time $\tilde{O}(N^{2/3})$. $\leftarrow N^{1/3}$ tables, $N^{1/3}$ time to search each

Success Prob

Heuristically, treat each $f \circ g_i$ as an indep random fn.



T_1 inverts $f \circ g_1$

$T_{N^{1/3}}$ inverts $f \circ g_{N^{1/3}}$

$\Pr_y [\text{Table}_i \text{ covers } y \in [N]] \approx \frac{1}{N^{1/3}}$

On avg. all tables cover $\Omega(N)$ points
↳ Invert any point w/ constant prob.

Yao's Lower Bound

We have a preproc alg for inverting

OWPs: $S = T = \tilde{O}(N^{1/2})$

OWFs: $S = T = \tilde{O}(N^{2/3})$

Can we do better?

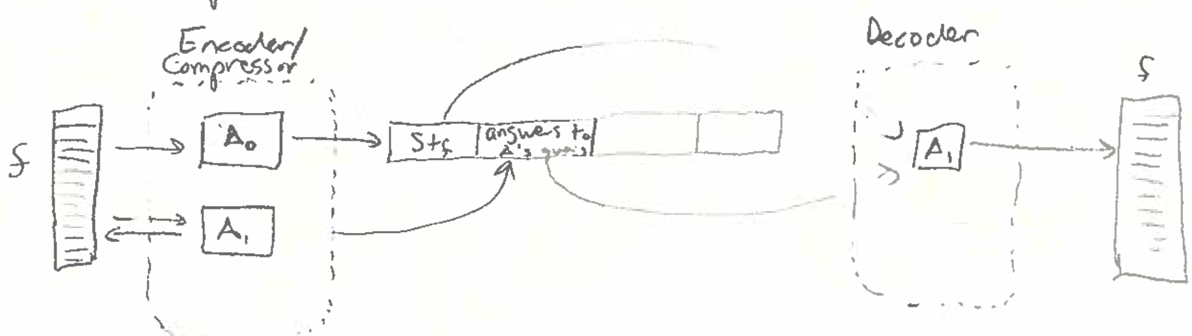
Thm [Yao]: Any preproc alg that inverts all OWP must satisfy $ST \geq \tilde{\Omega}(N)$.

↳ Shows that "cycle-walking" is optimal for OWP

Pf Idea

If we had a better preproc alg, could compress a random string.

↳ Not possible



* Run A_0 to get advice
* Run A_1 on $\sqrt{N}$ points in $[N]$
  - answer A_1's queries
  - write answers into encoding

Idea: - Each time we run A_1 we "pay for" T queries, we "get back" $T-1$ points $\Big\} \times \log N$ bits of "profit"

- Can run $A_1 \approx \frac{N}{T}$ times → $\sim \frac{N}{T} \log N$ bits of profit

- Encoding overhead: $S - \frac{N}{T} \log N \geq 0$ $\Big\{$ Random string is incompressible, so overhead is non-neg

$\Rightarrow ST \geq \tilde{\Omega}(N)$.