Lecture 1 : Intro & Basic Primitives

3/30/21

Welcome to CS 355!

Instructors: Alex Ozdemir
Riad Wahby
Saba Eskandarian
} PhD students with Dan Boneh

Today's Plan

Introduction

Course overview
Course logistics

Foundations of Cryptography

Defining security
One-way functions (OWF) ← basic tool for all of symmetric crypto!
OWFs → PRGs

What is CS355?

1. Your <u>first</u> advanced course in Cryptography:
   - learn to use important formalisms and tools
   - understand open problems in crypto
   - prepare for doing crypto research

2. Your <u>last</u> advanced course in cryptography:
   - understand cutting-edge crypto in use today
   - background to read and understand crypto papers
   - prepare you to use crypto to change the world!

Topics

Unit 1: Foundations of Crypto

Unit 2: Cryptanalysis & Elliptic Curve Cryptography

Unit 3: Zero Knowledge

Unit 4: Multiparty Computation

Unit 5: Lattice-based Crypto

↳ one of the most popular forms of <u>post-quantum</u> crypto!

Logistics

Website: https://cs355.stanford.edu (make sure you're on the 2021 site!)

Look here for course policies, HWs, OHs, and links to everything

Contact: cs355@cs.stanford.edu for individual questions

Piazza for questions about material, problem sets, policies
also all announcements

Anonymous feedback link on website (course staff page)
please send feedback throughout quarter!

Lectures:
- lecture recordings on Canvas, but please attend lecture!
- lecture notes posted to website after class
- No textbook, optional supplemental readings on website
- If interested in learning more, try cryptobook.us

Office hours:
- See website (please give feedback if times don't work)
- Not recorded
- feel free to email us if you want to talk 1-on-1

Problem sets:
- 5 total, one every two weeks
- Use Latex, submit via Gradescope
- 1^{st} Pset is out today, due Monday, April 12

Note: CS355 = CS255 + 100
The HW problems are meant to be challenging!
- Start early
- Come to OH
- ask questions

Foundations of Modern Cryptography

Modern cryptography is closely connected to the study of <u>hardness</u>

↳ lots of overlap b/w foundations of crypto and computational complexity

General approach: hard problem → crypto scheme

Security: if crypto scheme broken → hard problem solved

Examples of hard problems from CS255:
- factoring
- discrete logarithm
- DDH

Q: Using fancier and fancier assumptions, we can get more and more interesting crypto. But where do these assumptions come from? Can we get closer to building crypto from hard problems we may recognize from a theory or complexity class, like NP-hard problems? Is there a minimal assumption we can have a lot of confidence in that's enough to build some crypto?

Introducing the basic assumption of modern symmetric crypto:

One-way functions! (OWFs)

Note: In the first couple lectures, we will focus on <u>symmetric crypto</u>.
The tools & notions we introduce here will be important for more advanced concepts.

One-way functions (OWFs)

Intuitively, a OWF is a function that's easy to compute but hard to invert.

i.e. given x, easy to compute $f(x)=y$ poly-time

given y, hard to compute x s.t. $f(x)=y$ not poly-time

It turns out that given a function f with this property, we can build <u>all</u> the symmetric crypto from CS255: PRGs, PRFs, PRPs, etc...
We'll see how in this lecture and the next.

More formally

input & output spaces parameterized by λ, often called the "security parameter." Think of X_λ, Y_λ as $\{0,1\}^\lambda$

Def: A function $f: X_\lambda \to Y_\lambda$ is one-way if for all PPT ("efficient") adversaries ("algorithms") A:

$$\Pr[\underbrace{x \xleftarrow{R} X_\lambda}_{\text{for } x \text{ chosen at random from } X_\lambda} : A(\underbrace{1^\lambda}_{\text{a string of } \lambda \text{ "1"s.}}, \underbrace{f(x)}_{A \text{ is given } f(x)=y\text{, wants to invert it.}}) \in f^{-1}(f(x))] \leq \underbrace{\text{negl}(\lambda)}$$

This would be just "$=x$", but $f(x)=y$ may have more than one preimage

a function f is "negligible" in λ if $f(\lambda) = o\left(\frac{1}{\lambda^c}\right)$ for all constants $c \in \mathbb{N}$

a string of λ "1"s.
This technicality ensures that a poly-time algorithm A can run in time $\text{poly}(\lambda)$

How to build a OWF? (do they even really exist??)

Candidate OWFs:

1) $f(x,y) = x \cdot y$ for equal length primes x, y.
2) $f(x) = g^x$ where $g \in G$, G is prime order group
3) Levin's universal OWF:
   Function f_L s.t. $\exists$ OWF $\rightarrow f_L$ is OWF.

Can we prove OWFs exist without making additional assumptions?

Probably not. OWFs exist $\rightarrow$ $P \neq NP$

Come to OH and we can talk about why!

Can we prove OWFs exist assuming only that $P \neq NP$?

Major open problem!

From OWFs to Symmetric Crypto

$$\begin{tikzcd}[column sep=huge]
\text{OWFs} \arrow[r, bend left, "\substack{\text{Today!}\\ \text{Goldreich-Levin}}"] & \text{PRGs} \arrow[l, bend left, "\text{by definition}"] \arrow[r, bend left, "\substack{\text{Thursday!}\\ \text{Blum-Micali}\\ \text{GGM}}"] & \text{PRFs} \arrow[l, bend left, "\text{counter mode}"] \arrow[r, bend left, "\substack{\text{Feistel networks/}\\ \text{Luby-Rackoff}}"] & \text{PRPs} \arrow[l, bend left, "\substack{\text{Switching}\\ \text{Lemma}}"] \arrow[r, "\substack{\text{MAC from PRF}\\ +\\ \text{Enc then MAC}}"] & \text{Auth Enc}
\end{tikzcd}$$

Some notes:

Collision resistant hash functions are missing from this picture. OWFs are <u>not</u> known to imply CRHFs and vice versa.

Public-key crypto is missing from this picture. It can be proven, with some caveats, that OWFs are <u>insufficient</u> for public key crypto!

↳ There is an area of crypto that focuses on <u>separations</u>, proving that some assumption <u>cannot</u> get you a desired functionality!

Theory vs Practice: in practice, we usually directly assume that AES is a secure PRP and go from there.

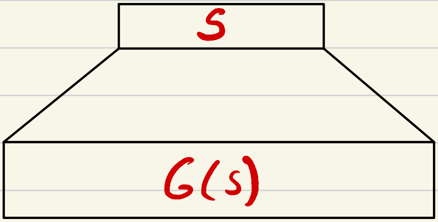
why? The relationships above are very important for our theoretical understanding of crypto and its foundations in complexity theory, but the transformations, while "efficient" (i.e. poly-time), are often not practical.

Also, the tools/techniques we develop here will come up again & again

Today we will see how OWFs imply PRGs

Pseudorandom Generators (PRGs)

Recall: A PRG takes a short random seed s and expands it into a longer "random-looking" string $G(s)$.



$s \in \{0,1\}^\lambda$ ← security parameter is length of seed

$G(s) \in \{0,1\}^{\ell(\lambda)}$ ← the "stretch" of the PRG

Q: Why must it be the case that $\ell(\lambda) > \lambda$?

Q: What does it mean to be "random-looking"?

1) "information-theoretic security": $G(s)$ is uniformly random in $\{0,1\}^{\ell(\lambda)}$

This is impossible!

2) "Computational security": No efficient (PPT) algorithm (adversary) can distinguish $G(s)$ from a truly random string

→ Define a "distinguishing algorithm" as one that takes a string as input and "guesses" whether the string is the output of a PRG or a truly random string.

e.g. algorithm outputs 1 when it guesses the string is pseudorandom

Formalizing PRGs

Def: A PRG $G: \{0,1\}^\lambda \to \{0,1\}^{\ell(\lambda)}$ is a deterministic poly-time algorithm. It is secure if for all PPT adversaries A:

$$\Big| \underbrace{\Pr[s \xleftarrow{R} \{0,1\}^\lambda : A(G(s)) = 1]}_{\text{Probability } A \text{ outputs } 1 \text{ given pseudorandom value}} - \underbrace{\Pr[t \xleftarrow{R} \{0,1\}^{\ell(\lambda)} : A(t) = 1]}_{\text{Probability } A \text{ outputs } 1 \text{ given truly random value}} \Big| \leq \text{negl}(\lambda)$$

Intuitively, behavior of A should not vary much between PRG outputs and truly random outputs.

We call the difference between these two probabilities the adversary's distinguishing advantage PRGAdv[A, G].

A more general view: Computational indistinguishability

Often we'll define probability distributions corresponding to two "worlds":

distribution $D_0 = \{s \xleftarrow{R} \{0,1\}^\lambda : G(s)\}$ "pseudorandom world"

distribution $D_1 = \{t \xleftarrow{R} \{0,1\}^{\ell(\lambda)} : t\}$ "truly random world"

We say D_0 and D_1 are computationally indistinguishable if no PPT adversary can distinguish draws from D_0 from draws from D_1.

This is denoted $D_0 \approx_c D_1$

OWF → PRG

Will actually show OWP → PRG (OWF case is harder)

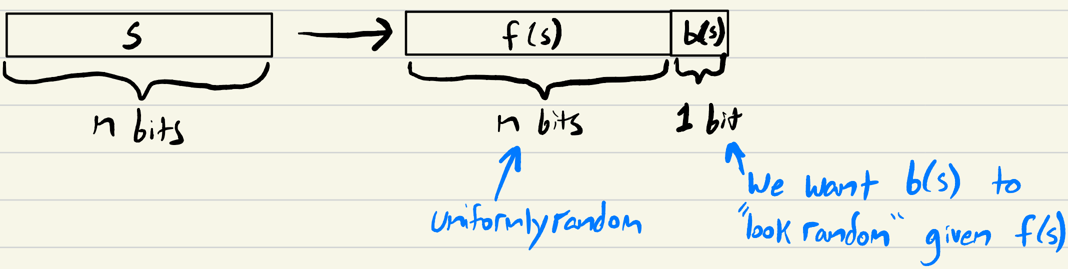
One Way Permutation (OWP): f is a OWP if

1) f is a OWF

2) $\forall n \in \mathbb{N}$: $f\{0,1\}^n \to \{0,1\}^n$ is a permutation

Today: OWP → PRG with 1-bit stretch
next time we'll extend to arbitrary stretch

Observation: on a random input, output of OWP is also random
↳ let's use that as first n bits of PRG output!



New idea: Hard core bits (AKA hard core predicates)

Given $f(x)$ for a OWF, finding x is hard.

Q: what about finding the first bit of x?
A: Not necessarily! E.g. $f(x_1, \dots, x_n) = x_1, f'(x_2, \dots, x_n)$

But it cannot be the case that all $x_1, \dots, x_n$ are easy to compute, or else f isn't a OWF!

Def: A hard core bit b for a OWF f is a bit s.t.

1) $b(x)$ can be computed in poly time

2) Any PPT adv A given $f(x)$ can only guess $b(x)$ with probability at most $\frac{1}{2} + \text{negl}(n)$

b(x) looks random!

If we have a hard-core bit $b(x)$ for a OWP f, we can build our PRG G as $G(s) = f(s) \| b(s)$.

But how do we get a hard core bit?

Thm (Goldreich-Levin): Every OWF has a hard core bit!

idea: a random linear combination of the bits should be hard to compute.

First extend the function f to $g(x,r) = (f(x), r)$ where $|r| = |x|$

$\uparrow$ n bits

Observe that g is still one-way

Now $b(x,r) = \langle x, r \rangle = \sum_{i=1}^{n} x_i \cdot r_i \bmod 2$ $\leftarrow$ Inner product mod 2

See Supplemental reading on web site for details / proof!