600.641 Special Topics in Theoretical Cryptography January 22, 2007

Lecture 1: Introduction to Cryptography

Instructor: Susan Hohenberger *Scribe: Karyn Benson*

1 Background

This is an advanced graduate course. We will begin by studying some of the major results in "modern cryptography" starting in the mid 1980's. At this point in time, Diffie-Hellman and RSA have already been introduced to the public. What is next? We should start with something new; instead of starting with signatures and encryption we start with complexity theory. We ask the question: Can I convince you of an answer without giving you an answer?

2 Introduction to Complexity Theory

Let's begin with an introduction to complexity theory, since these concepts will be critical to understanding the some of the major advances in modern cryptography. We'll start with some definitions.

2.1 Definitions

Definition 1 *A* <u>*language*</u> *is a set of strings.*

Example 1 $PRIMES = \{2, 3, 5, 7, 11, 13, \cdots\}$

Definition 2 *We are interested in* <u>*decision problems*</u>*, where we are given some string and must determine if it is in the set.*

Example 2 *Given an integer n is, $n \in PRIMES$?*

Question 1 *How much harder does the decision problem become as n gets larger?*

Note that complexity theory considers the worse case, that is, for all n of a given length, what is the *worst* performance of the algorithm. Also, the algorithm must always be correct.

$PRIMES$ is an example of a decision problem that was studied for centuries. (Recently, it was found to be in P [AKS04]. We'll say more on this later.)

The resources we consider when answering the above question are **time** and **space**.
We consider polynomial time to be "efficient time". And we consider polynomial space to be "efficient space". In this course, we will primarily be concerned with time rather than space.

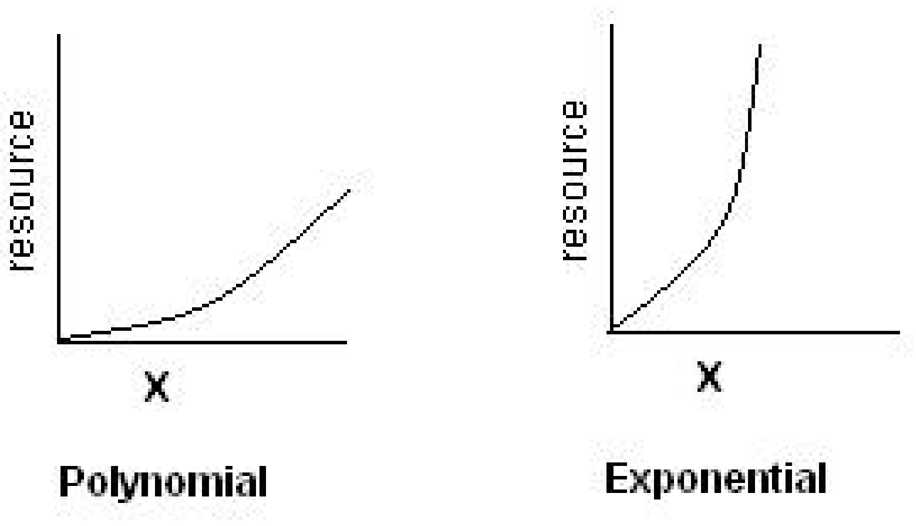

Figure 1: Polynomial vs. Exponential: Plotted above on the x-axis x represents the length of the string and on the y-axis the maximum resource consumption for any string of length x. In the exponential case, the problem gets *much* harder as x increases.

2.2 Classification of Languages

The languages before the curved line in Figure 2 are hard but computable; those after are not even computable! It is impossible to write an algorithm that can always decide if a program will get into an infinite loop or not. It is also impossible to write a program that always predicts the future.

As computer scientists we only concern ourselves with what is computable. On the left side of the curved line, we are asking questions like: are all numbers in this list prime?

In Figure 3, we illustrate some known relationships between classes of languages. Let us now define and discuss these classes.

P is the set of languages decidable in **p**olynomial time.
PSPACE is the set of languages decidable in **p**olynomial **space**.
EXPTIME is the set of languages decidable in **exp**onential **time**.
It is known that P $\subseteq$ PSPACE $\subseteq$ EXPTIME.

NP, which stands for **n**ondeterministic **p**olynomial time, is the class of languages L where, for all $x \in L$, this fact can be verified in polynomial time. That is, for all $x \in L$ there exists a proof of this fact that a verifier can check in polynomial time. It is known that P $\subseteq$ NP $\subseteq$ PSPACE $\subseteq$ EXPTIME.

The only additional thing we know about P, NP, PSPACE, and EXPTIME is that

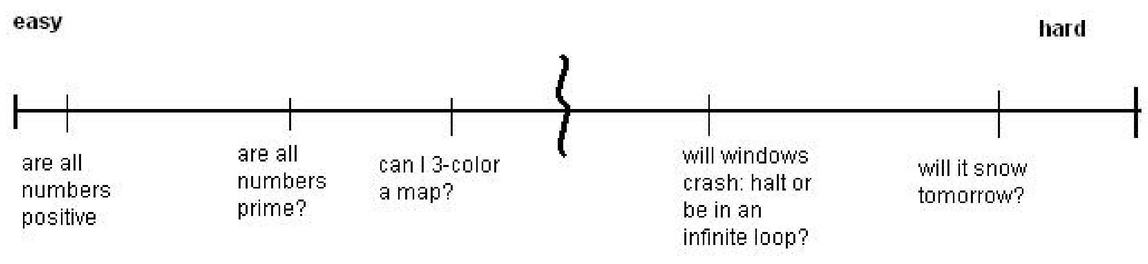

Figure 2: Range of Languages

EXPTIME $\neq$ P. (Or in other words, P is a strict subset of EXPTIME, denoted as P $\subset$ EXPTIME.) In complexity and cryptography PSPACE is considered efficient space, P is considered efficient time.

2.3 More on NP

Question 2 *Suppose I claim $x \in L$, and I give you a convincing argument. How hard is it for you to check my answer? What if I can always give you a proof that you can check quickly?*

A language L is in NP if a given argument A can be checked in polynomial (efficient) time.

NP = $\underline{\text{N}}$on Deterministic $\underline{\text{P}}$olynomial Time

We know that NP is enclosed in PSPACE. However, the greatest unsolved problem is: is $P = NP$ or $P \neq NP$.

In fact, it is truly amazing that PRIMES is in P (which means it is also in NP). Take a moment to think about what kind of short proof you could give to someone so that they can quickly verify that a number is prime. Proving that a number is *not* prime is easy (provide a divisor), but proving that it is prime seems much more difficult.

2.4 Complexity of Games

Just for fun, suppose each of the games below is treated as a decision problem. For example, suppose the language TETRIS is the set of all (gameboard, sequences of pieces) such that there exists a strategy for playing this sequence of pieces on this gameboard that result in a total clearing of the board. Then the table below summarizes the complexity of some popular games.

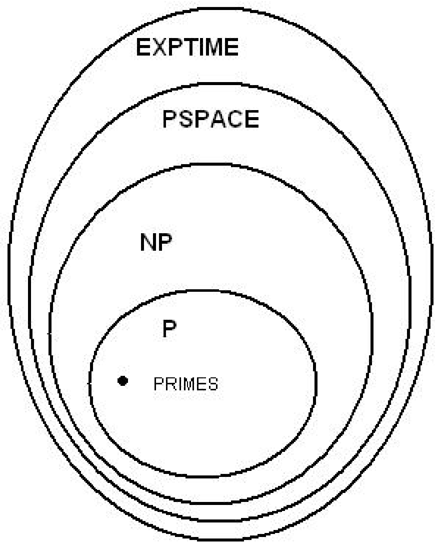

Figure 3: Relationships Between Classes of Languages

Class	Games
P	"I'm thinking of a number," Jenga
NP	Minesweeper, Tetris
PSPACE	Sokoban, Go (Japanese Version)
EXPTIME	Checkers,Chess

2.5 Complexity and Cryptography

Let's return for a moment to the greatest open problem in computer science: P vs NP. Suppose $P = NP$, this would be a positive for many operations research areas, such as airline scheduling, factory layouts, UPS scheduling,.... Today, we only have algorithms that approximate answers to these problems and/or run with some probability of error. However, if $P = NP$, this would be a negative for modern cryptography since it exploits the gap between P and NP. In fact, while $P \neq NP$ is not *sufficient* for many complexity assumptions made in modern cryptography, it is *implied* by the majority of them (i.e., the existence of one-way functions.)

Informally, a one-way function is a function that is easy to compute, but hard to invert. Consider the following example: factoring.

Example 3 *Given two random large primes p, q.*
$\phi(p, q) = n = pq$ is easy to compute but $\phi^{-1}(n) = (p, q)$ is believed to be hard.

Factoring, when reformulated as a language, is known to be in NP but not necessarily in P.

Human knowledge advances either way! The majority of modern cryptographic systems are based on complexity *assumptions*. That is, assumptions that a certain function is hard to invert (which implies $P \neq NP$). These assumptions may in fact be false! However, as researchers, we win either way! Suppose a cryptosystem is based on factoring and that factoring is truly hard – great, we have a secure cryptosytem. Now, suppose a cryptosystem is based on factoring and that factoring turns out to be easy – great, humankind has solved another hard math problem!

2.6 NP-Completeness

Some progress in the P-NP gap has been made. Through a class of problems known as NP Complete, through independent work by Cook and Levin in the 1970's. Before this concept to prove $P = NP$ it was necessary to prove that every language in the P-NP gap is actually in P. However, there are many of problems in this gap, and in fact there does not exist a central directory listing all of these problems. So how would it even be possible to show that $P = NP$ if this is indeed the case? Cook and Levin showed that there exists a "hardest" problem in NP; if the hardest problem could be solved in P, then all problems in NP can be solved in P. There are actually many problems which are the hardest problem. The equivalence is shown through reductions.

Definition 3 *A language is* NP Complete *if it is both*

1. *in NP*
2. *hard as the hardest language in NP.*

The NP-Complete Language Cook and Levin came up with was SAT: the set of all satisfiable circuits.
Let $\wedge$=and, $\vee$=or.
An example of a circuit in SAT is: $(x_1 \vee x_2) \wedge x_3 \wedge \bar{x_1}$.
An example of a circuit not in SAT is: $x_1 \wedge \bar{x_1}$. The decision question here is: given a circuit, is there any way to set its variables such the circuit evaluates to true?

References

[AKS04] Manindra Agrawal, Neeraj Kayal, and Nitin Saxena. PRIMES is in P. *Ann. of Math.*, 160(2):781–793, 2004.