Basic Cryptography Overview

Jared Saia

January 15, 2006

1 Introduction

In these notes, we will briefly review the main results that are known in basic cryptography. We will not go over the proofs of these results - doing so would require a separate one or two semester class. However, much of what we will discuss in class will make use of these basic tools. Much of the discussion here is based on the textbook *Foundations of Cryptography* by Oded Goldreich which has details of the proofs which are omitted here.

1.1 Encryption Schemes One of the most basic problems in cryptography is that of providing secret communication over insecure channels. The typical setting for this problem consists of two players, Alice, the sender, and Bob, the receiver, that are communicating over a channel which an adversary, Eve, can listen in on. The encryption scheme typically consists of a pair of algorithms. The first algorithm, the *encryption* is applied to the plaintext by Alice to create the cyphertext. The second algorithm, called the *decryption* is applied by the Bob to cyphertext in order to recover the plaintext.

For this scheme to provide secret communication, Bob must know something that Eve doesn't know (otherwise Eve could decrypt the message in exactly the same way that Bob does). This extra knowledge is called the *decryption key*. We generally assume that Eve knows the decryption algorithm and the cyphertext, but not the decryption key. The existence of such a secret key is a necessary condition for secret communication.

There are two approaches to evaluating the security of an encryption scheme. The first approach is *information theoretic* which provides very strong guarantees but is generally impractical. This approach is concerned about the "information" about the plaintext that is present in the cyphertext. If the cyphertext contains information about the plaintext, then the encryption scheme is considered insecure. However, it can be shown using Information Theory that a high level of security can only be achieved in this scheme if the decryption key is at least as long as the length of the plaintext. This makes this scheme impractical when wanting to send very long messages. An example encryption scheme that provides information theoretic security is the *one time pad*. In this scheme, there is a perfectly random sequence of bits, r, known to both Alice and Bob but not to Eve. This sequence r is the same length as the plaintext message p. Alice sends Bob the message $m = p \oplus r$ and Bob recovers the plaintext by noting that $p = m \oplus r$. The downside of this scheme is that it requires a large number of perfectly random bits to be known secretly by Alice and Bob but not Eve.

The more "modern" approach to evaluating secrecy is based on *computational complexity*. The basic idea in this approach is that it doesn't matter if the cyphertext contains some information about the plaintext so long as this information can't be extracted efficiently. This new approach offers security even if the key is much shorter than the plaintext. For example, one can use *Pseudorandom Generators* that expand short keys into much longer "pseudo-keys" so that the pseudo-keys are just as secure as the short keys (in the computational sense, making certain assumptions).

The computational complexity approach also allows for algorithms that can't exist under the

information theoretic approach. An example of this is the *public-key encryption scheme*. We note that the encryption algorithm must get in addition to the plaintext message another input which is the *encryption key*. All encryption schemes used prior to the 1980's assumed that the encryption key and the decryption key were the same (*private-key encryption*. In this case, Eve could not know the encryption key and so the key-distribution problem arises - i.e. how to make sure that Alice and Bob can agree on a secret encryption/decryption key. In the computational complexity approach, it's possible for Eve to know the encryption key without compromising security. In such schemes, under certain assumptions, it is infeasible to compute the decryption key even if Eve knows the encryption key.

2 One-way functions

One-way functions, loosely speaking are functions which can be computed in polynomial time but which are computationally infeasible to invert. More precisely they can be defined as follows:

Definition A function $f : \{0,1\}^* \rightarrow \{0,1\}^*$ is called (strongly) *one-way* if the following two conditions hold:

1. *Easy to compute*: There exists a deterministic polynomial-time algorithm A such that on input x algorithm A outputs $f(x)$ (i.e. $A = f(x)$)

2. *Hard to invert*: For every probabilistic polynomial time algorithm, A', every positive polynomial $p()$ and all n sufficiently large,

$$Pr[A'(f(U_n, 1^n) \in f^{-1}(f(U_n))] < \frac{1}{p(N)} \tag{2.1}$$

In the above, U_n is a random variable distributed over $[0,1]^n$. A' is given the auxiliary input 1^n simply to rule out the possibility that a function will be one-way simply because it drastically shrinks its input. The second condition in the above is basically saying that there is no (possibly randomized) algorithm A' with the following property. A' is able to compute with non-negligible probability the quantity x when it's given the value $f(x)$ and x is a random binary string of length n.

It is currently not know if one-way functions exist and in fact the existence of one-way functions implies that NP is not contained in BPP which would imply that $P \neq NP$ (it's not known if the reverse implication is true). Recall that BPP is the class of problems for which there exist randomized Monte Carlo algorithms.

Assume that there exists some one-way function f. Consider the language $L = \{(y,p) :$ there exists a string $x \leq p$ and $f(x) = y.$ $\}$. Clearly this language is in NP since x is a witness to the fact that (y,p) is in L. However, if the language were in BPP, then one could do a binary search over all values of p to determine x (there are an exponential number of values of p but the logarithm of this will just be polynomial). Thus, with non-negligible probability, for any string y we could find a string x such that $f(x) = y$. This contradicts the assumption that f is a one-way function.

3 Pseudorandom generators

Loosely speaking, a pseudo-random generator is a deterministic algorithm that expands short random seeds into much longer bit sequences that appear to be random. In other words, it is computationally infeasible to tell the difference between the output of the pseudorandom generator and a real sequence of random bits. More precisely (from notes by Rafail Ostrovsky):

Definition A deterministic algorithm $G(-,-)$ is a pseudorandom generator if:

1. $G(x, Q)$ runs in time polynomial in $|x|, Q(|x|)$ where Q is a polynomial
2. $G(x, Q)$ outputs strings of length $Q(|x|)$ for all x
3. For every polynomial $Q()$, the induced distribution $\{G(x, Q)\}$ is polynomial-time indistinguishable from $\{U_{Q(|x|)}\}$ where $\{U_{Q(|x|)}\}$ is the uniform distribution on strings of length $Q(|x|)$.

A key result in cryptography is that a pseudorandom generator exists iff a one-way function exists.

from notes by Ostrovsky: The existence of a pseudorandom generator allows us to construct a Here we will only show that any pseudo-random generator is a one-way function. Suppose that a pseudo-random generator G : 0, 1 n ? 0, 1l(n), l(n) n + 1, is not a one-way function. Then a pseudo-random string of length l(n) can be inverted with non-negligible probability. However, recall that there are only 2 n seeds of length n and there are 2l(n) strings of length l(n), which means that most truly random strings of length l(n) cannot be inverted. This is due to the fact that there are at most 2 l(n) 2 n times as many outputs as seeds. Based on the assumption that l(n) n + 1, there are at least twice as many outputs as seeds. If a pseudo-random string of length l(n) can be inverted with nonnegligible probability, a distinguisher between truly random and pseudo-random strings can be constructed. If we have such a distinguisher, we can try to invert a given string: if we are able to invert it, it is pseudo-random; if not, then it is truly random. However, a pseudo-random generator, by definition, has an output that cannot be distinguished from truly random in polynomial time. Therefore, if it is a pseudo-random generator, its output cannot be inverted in polynomial time and it is a one-way function. The work of H?astad, Impagliazzo, Levin, and Luby showed that any one-way function can be used to construct a pseudo-random generator.

4 Pseudorandom Functions

Pseudorandom functions create outputs which seem random but the functions themselves are deterministic. A truly random function $U : 0,1^m \to 0,1^n$ is basically a lookup table with 2^n entries of length $l(n)$. Pseudorandom functions are generally assumed to have keys k, so a pseudorandom function, F will map $0,1^k \times 0,1^m \to 0,1^n$. We will use a computational model to define precisely the notion of being unable to distinguish a function F_k from a random function U. In particular, we will assume that the adversary is given as a "black box" either a functions F_k or a truly random function U. The adversary is allowed to call the black box a polynomial number of times but can determine absolutely nothing else about the box.[1] We will call the adversarial algorithm A and will use $A^{F_s()}$, (or $A^{U()}$) to denote the output of A when given $F_s()$ (respectively $U()$) as a black box.

Definition A function $F : 0,1^k \times 0,1^m \to 0,1^n$ is pseudorandom if for all randomized algorithms A running in polynomial time in n, the following is true.

$$|Pr(s \leftarrow \{0,1\}^k : A^{F_s()} = 1) - Pr(U \leftarrow \mathbf{Rand}^{m \to n} : A^{U()} = 1)| \leq 1/p(n) \tag{4.2}$$

where $p(n)$ is some polynomial in n, and $U \leftarrow \mathbf{Rand}^{m \to n}$ denotes setting U to a truly random function.

Essentially this definition is saying that no randomized algorithm running in polynomial time is able to distinguish the function F from a truly random function U with non-negligible probability.

We state the following theorem without proof.

THEOREM 4.1. *Pseudorandom functions exist iff one-way functions exist*

[1] Typically, this "black box" is referred to as an *oracle*

5 Private-key Encryption Based on Pseudorandom functions

Key generation consists of selecting a seed denoted s, for the pseudorandom function, denoted f_s. To encrypt a message $x \in \{0,1\}^n$ the encryption algorithm uniformly selects a string $r \in \{0,1\}^n$ and produces the ciphertext $(r, x \oplus f_s(r))$. To decrypt the cyphertext, (r, y), using key s, the decryption algorithm just computes $y \oplus f_s(r)$.

The proof of the security of this encryption scheme consists of two steps.

1. Prove that an idealized version of the scheme in which one uses a uniformly selected function $f : 0, 1^n \to 0, 1^n$ rather than the pseudorandom function is secure.

2. Conclude that the real scheme is secure since otherwise, one could distinguish a pseudorandom function from a truly random one.

Note that we can get rid of the randomness, r, if we allow the encryption algorithm to be history-dependent (i.e. use a counter instead of r).

5.1 Block Ciphers In practice, people use block ciphers. Block cyphers are keyed functions as described above.

todo

6 Message Authentication Based on Pseudorandom Functions

In the message authentication problem (often referred to as the MAC problem), Alice wants to send Bob a message, m, (possibly in plaintext) over an insecure channel. We want to make sure that Bob call tell if the message m is actually from Alice or if it is a forgery from Eve. Assume Alice and Bob share a secret key s to a pseudo-random function F, i.e. they both know F_s. A message authentication code consists of two algorithms: the MAC algorithm which takes a secret key k and a message m and returns a tag tag; and the verification algorithm $VRFY$ which takes a key s, a message m and a tag, tag and returns 1 if the tag is valid and 0 otherwise.

How do we define security for such a scheme? Roughly the adversary should not be able to forge a valid tag on any message that was not sent by Alice. In general, we'd like to prevent the adversary from fooling the receiver even after the adversary sees multiple tags computed on the messages. We'll actually assume the adversary is even more powerful in that it actually gets to see multiple tags for messages of the adversaries own choice.

We'll define an oracle MAC which takes a message m as input and returns as output the tag for that message using secret s. The adversary can call this oracle as much as it likes but can determine nothing else from it except for what it outputs [2]. The adversary breaks the scheme if it can forge a valid tag for some message that it never submitted to the oracle i.e. it can output a pair (m', t') such that $VRFY(m', t') = 1$ and m' is not in the set of messages the adversary submitted to the oracle.

Definition A message authentication scheme (MAC) is secure if for all randomized algorithms A, running in polynomial time in the message lengths, we have that

$Pr(s \leftarrow 0, 1^k; (m', t') \leftarrow A^{MAC_s()} : VRFY(m', t') = 1$ and m' is not in the set of messages that A submitted t

where $p(n)$ is some polynomial in n.

[2]Note that the adversary knows the algorithm used by the black box, but is assumed not be able to use external factors like e.g. runtime, heat dissipation,etc in order to determine s. The adversary only knows the algorithm and the outputs for every input given

Example of message authentication protocol which is not secure is to just exclusive-or the message with s. Why is this not secure?

Following is a protocol which is secure assuming that F is a pseudorandom function and Alice and Bob share a secret s. For a message m, $MAC(m) = F_s(m)$ and $VRFY(m,t)$ is 1 iff $F_s(m) = t$. It is possible to prove this scheme is secure as defined above if F is a pseudorandom function. We will not do that here but the intuition behind the proof is as follows. If the adversary were able to break the scheme in polynomial time, this means that the adversary would be able to find an input for F_s, which it never fed to the oracle, for which it can predict the output. The adversary can then be sure that F_s is not a random function. But this then implies that the adversary is able to differentiate between F_s and random functions in polytime which contradicts our assumption that F is a pseudorandom function.

6.0.1 Secret Societies

Imagine a secret society where the members all know some secret s. There's a group

7 Public Key Cryptography

describe public key encryption *describe notion of security*

Say what minimal assumptions needed to build public-key

All efficient public-key encryption schemes [3]and digital signature schemes

define random-oracle model

[3]with the possible exception of the El Gamal encryption scheme which we will not discuss in class