Copyright ©1996 by Phillip Rogaway. All rights reserved.

ECS 227: Modern Cryptography
Lecture 1

Introduction

Phillip Rogaway
9 January 1996

Reading for Lecture 1:

[**DiHe76**] Whitfield Diffie and Martin Hellman, "New Directions in Cryptography." *IEEE Transactions on Information Theory*, vol IT-22, No. 6, November 1976.

[**Ri90**] Ronald Rivest, "Cryptography." From the *Handbook of Theoretical Computer Science*, edited by J. van Leeuwen, Elsevier Science Publishers B.V., 1990.

1 Introduction

1.1 What is Cryptography?

For unknown reasons, people like to try to "define" cryptography. One nice description, from 1976, was that:

> Cryptography is the study of "mathematical" systems for solving two kinds of security problems: privacy and authentication. [DiHe76]

Nowadays, cryptography is often understood to include goals even beyond privacy and authentication (e.g., the construction of a pseudorandom generators). But certainly privacy and authentication are at the heart of things.

One things which the above description omits is the central role of the *adversary*. Rivest says simply:

> Cryptography is about *communication in the presence of adversaries.* [Ri90]

Indeed, there is no cryptography in the absence of an adversary, and any rigorous treatment of cryptography elevates her to being a central character. One way to be sure that you've left cryptography is to realize that there isn't any adversary in sight.

1.2 Classical Cryptography

This course is called *modern cryptography* but what I really mean is cryptography in the *provable-security tradition.* To appreciate provable-security you should have some sense for how *classical cryptography* has operated.

Classical cryptography is an attack-responsive tradition in which (A) cryptographic problems are recognized; then (B) cryptographic solutions are offered; then (C) one searches for faults or weakness in these cryptographic solutions; and then (D) one modifies the cryptographic methods accordingly. The third step is called *cryptanalysis.* The process ought to continue forever. In practice, one eventually regards a solution as adequate, implements the solution, and ships products containing the solution. Hopefully, further cryptographic issues will not materialize. Often they do, and when this happens the company which owns the product may find it impractical to fix the fielded protocol.

Doing classical cryptography well takes great cleverness. Sometimes one finds problems with cryptographic primitives in the form of subtle mathematical relationships which allow one to subvert the protocol's aims. Sometimes, instead, one "jumps out of the system," showing that some essential cryptographic issue was overlooked in the design, application, or implementation of the cryptography.

1.3 Connections to Complexity Theory

Is the *ad. hoc.* nature of classical cryptography essential? One might think so, because of the following fallacious reasoning:

> Formalized in a natural manner, a correct solution to most cryptographic goals implies the existence of an object called a *one-way function.* The existence of a one-way function, in turn, implies P $\neq$ NP. We can not really expect proper foundations for a field whose most basic of goals *already* implies complexity-theoretic results stronger than what one really expects to prove.

The error in the above reasoning is indicated by the fact that complexity theory itself has made valuable contributions in computer science *despite* the unresolved nature of its most basic questions. The reason has been the power of

reductions. When we show that some problem Π is NP-Complete, we say something interesting about Π even though we have not managed to show P $\neq$ NP. Here we will do the same for cryptography: we will make (basic) cryptographic assumptions and achieve (non-basic) goals by *reductions.* We will be doing this for practical problems, and coming up with protocol protocols. Let us explain a bit more.

1.4 Provable Security

Provable security again begins with (1) recognizing some real-world cryptographic problem. But next (2) we formulate a *definition* for the problem at hand. Next we provide (3) a protocol, and then we (4) prove, (3.5) under a complexity theoretic assumption, that the protocol meets its definition (2). Now, if we want we (5) implement. No revisions to the protocol should be needed.

The definition step (2) can be quite difficult. In some cases, it is so difficult as to be infeasible for us humans to accomplish this in one swoop. In that case we split the process into two. First (1.5) we describe a *model* for the problem; and only then do we do a definition. The model is "where" the problem is defined. An example may be useful.

Suppose we want to define the protocol goal of how to compute the average of our salaries in a way that reveals only the correctly computed average. The *model* specifies the details of the communication, e.g.: at each beat of a global clock, each party i computes a message for player j which is some arbitrary probabilistic function f_{ij} of i's computational state and the messages received so far; the player's computation state is adjusted according to some function f_i of his previous computational state and the messages received so far; etc..

Something fundamental is missing from the model above. Do you see what it is? The adversary! One adversary we might imagine (a simple "rushing" adversary) could intercept each message to each player that is transmitted; change this message as the adversary wishes; and then deliver the revised message. Even more powerful adversaries might be considered, e.g., adversary's who can do all of the above, as well as *corrupting* some number of the players (thereby learning their internal computation state). The model must exactly specify the adversary's capabilities, and how a protocol runs in the presence of an adversary.

Once the model is pinned down, a protocol with its adversary gives rise to a probablity space on protocol executions. Typically these runs are classified as *good* (for the adversary) or *bad* (for the adversary). This classification is the heart of the definition of the goal, (2).

Successful adversaries are ones which succeed (i) often and (ii) without spending an excessive amount of computational resources. Desirable protocols are those for which there are no successful adversaries. That is, if the amount of resources (time, space, etc.) that an adversary spends is reasonable, then her chance of success is too small to care about.

Showing a protocol good, in the sense above, would seem to be a monumental task. After all, we are trying to defeat a universal quantifier, showing that *every* computationally reasonable adversary is ineffectual. Despite this difficult, cryptographers manage to prove quite amazing results.

We will be interested in making *quantitative* claims of security. These will have the following form.

> Suppose an adversary E can defeat (break the definition of) a protocol P with success probability ϵ, using computational resources R. Then there is some other adversary E' which can break the underlying cryptographic primitive with success probablity ϵ', using resources R'.

Here ϵ' and R' are functions of ϵ and R. We want that ϵ' be close to ϵ and R' be close to R. We regard the demonstrated value of the protocol as begin given by the functional dependencies of ϵ', R' on ϵ, R.

1.5 A Menagerie of Cryptographic Problems

Now that the approach has been vaguely sketched, let's look at the sort of *problems* cryptographers like to think about. The rest of this lecture I'll spend going through a few examples of such problems. I won't pay any attention to possible protocols for these problems, nor will I care about the problems' definitions. Be clear that what's given below are simple *descriptions* of cryptographic problems (step 1, above), not definitions (step 2). The latter must be completely unambiguous, partitioning the world into solutions and non-solutions, or quantifying a candidate solution. The former, in contrast, can be useful under far less strict criteria.

1.5.1 Message Privacy

The days in which cryptography meant *encryption* —transformations to preserve *privacy*— are long over. Nonetheless, encryption is the classical problem of cryptography, so we follow tradition and begin there. In this problem Alice (A) wants to send a secret message *msg* to Bob (B) over a wire on which an adversary Eve (E) is listening. Bob should understand the message; Eve shouldn't.

To make this goal achievable, in any sense at all, there will have to be some *asymmetry* between Eve and Bob. For if Eve and Bob had identical initial information and capabilities, and receive identical communications from Alice, then everything Bob could figure out Eve could figure out, too. Cryptography is about ways to exploit asymmetry in players information or capabilities.

Information asymmetry can be achieved in a number of ways. In *symmetric encryption* Alice and Bob share a key a which Eve lacks. Possession/non-possession of the key is the information asymmetry which symmetric encryption

exploits. Alice sends Bob a message by computing $y \leftarrow \mathcal{E}_a(msg)$. The *encryption function* $\mathcal{E}$ may be probabilistic or stateful. *Probabilistic* means that the outcome of computing the function depends on random coin tosses. *Stateful* means that the outcome of computing the function depends on the function values which have already been computed Bob recovers *msg* by *decrypting* the incoming message, setting $msg' \leftarrow \mathcal{D}_a(y)$. Of course one expects that, when $y \leftarrow \mathcal{E}_a(msg)$ and then $msg' \leftarrow \mathcal{D}_a(y)$, one has that $msg' = msg$.

There are other ways to set up the information asymmetry between Bob and Eve. In *asymmetric encryption* Bob possesses a *pair* of keys —a *public key*, PK_B, and a *secret key,* SK_B. Bob's public key is made publicly known and bound to his identity. When Alice wants to send a secret message *msg* to Bob, she computes $y \leftarrow \mathcal{E}_{\mathrm{PK}_B}(msg)$. Again, $\mathcal{E}$ may be probabilistic or stateful. When Bob receives a message y, he computes $\mathcal{D}_{\mathrm{SK}_B}(y)$. Of course one expects that, when $y \leftarrow \mathcal{E}_{\mathrm{PK}_B}(msg)$ and then $msg' \leftarrow \mathcal{D}_{\mathrm{SK}_B}(y)$, one has that $msg' = msg$.

The *trust model* specifies who, initially, has what keys. We have just described two different trust models for achieving the same basic aim.

1.5.2 Message Authenticity

I didn't have time to talk about this in class, but here is a brief description for the notes. In this problem Bob receives some message which is claimed to have originated with Alice. The channel on which this message has flowed is (like the Internet) quite insecure. Thus Bob wants to distinguish the case in which the message really did originate with Alice from the case in which the message originated with some adversary, Eve.

Once again, information asymmetry will be needed to solve this problem. The symmetric and asymmetric trust models described above are equally applicable here. In a *message authentication mechanism* Alice and Bob share a secret key, a. Using key a, Alice transforms message *msg* in such a way that (1) Bob, using a, can recover *msg* (usually Alice sends *msg* together with a short *tag*); and (2) Bob, using a, can be convinced that *msg* really did originate with Alice. In a *digital signature mechanism* Alice, this time, has a public key PK_A and a corresponding secret key SK_A. Alice transforms message *msg* in such a way that (1) Bob, using PK_A, can recover *msg* (usually Alice sends *msg* together with a short *signature*); and (2) Bob, using PK_A, can be convinced that *msg* really did originate with Alice (the one party who is supposed to hold SK_B).

1.5.3 Pseudorandom Generator

Computers are remarkably deterministic. But for lots of applications, "random numbers" are quite useful. These applications involve simulation, efficient algorithms, and cryptography itself.

How can a completely deterministic machine generate "random" numbers? It can't. But a machine can do the next best thing: it can *stretch* a little bit of

randomness into a lot of "pseudorandomness."

Suppose we wire to our computer a Geiger counter that generates a "random" bit every second. We run our computer for a little while and now it has 200 "random" bits. We won't worry about the "philosophical" question as to whether these bits are random in any *real* sense. We'll simply assume that these bits are completely unpredictable to anything "beyond" the computer which has gathered this data—mathematically, we'll treat these bits as random.

A *pseudorandom generator* (PRG) stretches a short (eg. 200-bit) "truly random" string into a much longer (a million bits, say) string which "looks" random.

Is this problem cryptography? If so, where's the adversary?!

The adversary shows up when you define the goal. We will be quite ambitious and say that the stretched string should look random to any "reasonable" algorithm— any computationally feasible method whose function in life is to make an intelligent guess "a million random bits" OR "the output of a pseudorandom generator applied to a random 200-bit string." This *distinguisher* is the adversary. She's trying to recognize that you've used a PRG instead of flipping a million fair coins.

1.5.4 Telephone Coin Flip

Alice and Bob are arguing on the phone. They want to decide who among has to show up to AI-class tomorrow morning. Neither wants to go. But one of them better go to take notes for the two of them.

Alice offers a simple solution to end their bickering. "Bob," she says, "I've got a penny in my pocket. I'm going to toss it in the air right now. You call *heads* or *tails*. If you get it right, I'll go to AI class and take notes for us. If you get it wrong, you'll have to go."

Bob's not as bright as Alice, but something troubles him about this arrangement ...

The *telephone coin flip* problem is to come up with a *communications protocol* so that, to the maximal extent possible, neither Alice nor Bob can cheat the other; and, at the same time, each of them learn the outcome of a fair coin toss.

1.5.5 Dating Problem

Alice and Bob are colleagues. Bob might —or might not— want to go out on a date with Alice. Alice might —or might not— what to go out on a date with Bob. The age-old problem (which keeps many people apart) is a desire not to have *your* interest known when the other is *not* interested. That is, Bob would rather not even ask Alice if she's going to say no, and Alice would rather not even ask Bob if he's going to say no. But if no one ever asks, how is anyone to ever know?

Solving the the dating problem means finding a protocol for Alice and Bob to run for which (1) if Alice and Bob *both* want to go out with the other, then they both learn this; (2) if Alice prefers *not* to go on a date with Bob, she doesn't learn whether or not Bob would like to go on a date with her; (3) if Bob prefers *not* to go on a date with Alice, he doesn't learn whether or not Alice would like to go on a date with him.

You will see that, in these last examples, we've gotten really quite far from encryption. This is part of the charm of modern cryptography— how it has expanded to include all sorts of interesting (and sometimes even practical) problems. Issues related to privacy are ultimately very *human* concerns. As a result, cryptography has a unique feel among topics in mathematics and computer science.