Cryptography

Lecture Notes from CS276, Spring 2009

Luca Trevisan
Stanford University

Foreword

These are scribed notes from a graduate course on Cryptography offered at the University of California, Berkeley, in the Spring of 2009. The notes have been only minimally edited, and there may be several errors and imprecisions. We use a definition of security against a chosen cyphertext attack (CCA-security) that is weaker than the standard one, and that allows attacks that are forbidden by the standard definition. The weaker definition that we use here, however, is much easier to define and reason about.

I wish to thank the students who attended this course for their enthusiasm and hard work. Thanks to Anand Bhaskar, Siu-Man Chan, Siu-On Chan, Alexandra Constantin, James Cook, Anindya De, Milosh Drezgich, Matt Finifter, Ian Haken, Steve Hanna, Nick Jalbert, Manohar Jonnalagedda, Mark Landry, Anupam Prakash, Bharath Ramsundar, Jonah Sherman, Cynthia Sturton, Madhur Tulsiani, Guoming Wang, and Joel Weinberger for scribing some of the notes.

While offering this course and writing these notes, I was supported by the National Science Foundation, under grant CCF 0729137. Any opinions, findings and conclusions or recommendations expressed in these notes are my own and do not necessarily reflect the views of the National Science Foundation.

San Francisco, May 19, 2011.

Luca Trevisan

Contents

Lecture 1

Introduction

This course assumes CS170, or equivalent, as a prerequisite. We will assume that the reader is familiar with the notions of algorithm and running time, as well as with basic notions of algebra (for example arithmetic in finite fields), discrete math and probability.

General information about the class, including prerequisites, grading, and recommended references, are available on the class home page.

Cryptography is the mathematical foundation on which one builds secure systems. It studies ways of securely storing, transmitting, and processing information. Understanding what cryptographic primitives can do, and how they can be composed together, is necessary to build secure systems, but not sufficient. Several additional considerations go into the design of secure systems, and they are covered in various Berkeley graduate courses on security.

In this course we will see a number of rigorous definitions of security, some of them requiring seemingly outlandish safety, even against entirely implausible attacks, and we shall see how if any cryptography at all is possible, then it is also possible to satisfy such extremely strong notions of security. For example, we shall look at a notion of security for encryption in which an adversary should not be able to learn any information about a message given the ciphertext, even if the adversary is allowed to get encodings of any messages of his choice, and *decodings* of any ciphertexts of his choices, with the only exception of the one he is trying to decode.

We shall also see extremely powerful (but also surprisingly simple and elegant) ways to define security for protocols involving several untrusted participants.

Learning to think rigorously about security, and seeing what kind of strength is possible, at least in principle, is one of the main goals of this course. We will also see a number of constructions, some interesting for the general point they make (that certain weak primitives are sufficient to make very strong constructions), some efficient enough to have made their way in commercial products.

1.1 Alice, Bob, Eve, and the others

Most of this class will be devoted to the following simplified setting: Alice and Bob communicate over an insecure channel, such as the internet or a cell phone. An eavesdropper, Eve, is able to see the whole communication and to inject her own messages in the channel.

Alice and Bob hence want to find a way to encode their communication so as to achieve:

- **Privacy:** Eve should have no information about the content of the messages exchanged between Alice and Bob;
- **Authentication:** Eve should not be able to impersonate Alice, and every time that Bob receives a message from Alice, he should be sure of the identity of the sender. (Same for messages in the other direction.)

For example, if Alice is your laptop and Bob is your wireless router, you might want to make sure that your neighbor Eve cannot see what you are doing

on the internet, and cannot connect using your router.

For this to be possible, Alice and Bob must have some secret information that Eve ignores, otherwise Eve could simply run the same algorithms that Alice does, and thus be able to read the messages received by Alice and to communicate with Bob impersonating Alice.

In the classical *symmetric-key cryptography* setting, Alice and Bob have met before and agreed on a secret *key*, which they use to encode and decode message, to produce authentication information and to verify the validity of the authentication information.

In the *public-key* setting, Alice has a *private key* known only to her, and a *public key* known to everybody, including Eve; Bob too has his own private key and a public key known to everybody. In this setting, private and authenticated communication is possible without Alice and Bob having to meet to agree on a shared secret key.

This gives rise to four possible problems (symmetric-key encryption, symmetric-key authentication, public-key encrpytion, and public-key authentication, or *signatures*), and we shall spend time on each of them. This will account for more than half of the course.

The last part of the course will deal with a fully general set-up in which any number of parties, including any number of (possibly colluding) bad guys, execute a distributed protocol over a communication network.

In between, we shall consider some important protocol design problems, which will play a role in the fully general constructions. These will be *commitment schemes*, *zero-knowledge proofs* and *oblivious transfer*.

1.2 The Pre-history of Encryption

The task of encoding a message to preserve privacy is called *encryption* (the decoding of the message is called *decrpytion*), and methods for symmetric-key encryption have been studied for literally *thousands* of years.

Various *substitution* ciphers were invented in cultures having an alphabetical writing system. The secret key is a permutation of the set of letters of the alphabet, encryption is done by applying the permutation to each letter of the message, and decryption is done by applying the inverse permutation. Examples are

- the Atbash ciphers used for Hebrew, in which the first letter of the alphabet is replaced with the last, the second letter with the second-to-last, and so on. It is used in the book of Jeremiah
- the cipher used by Julius Caesar, in which each letter is shifted by three positions in the alphabet.

There are reports of similar methods used in Greece. If we identify the alphabet with the integers $\{0, \ldots, k-1\}$, where k is the size of the alphabet, then the Atbash code is the mapping $x \to k-1-x$ and Caesar's code is $x \to x+3 \bmod k$. In general, a substitution code of the form $x \to x+i \bmod k$ is trivially breakable because of the very small number of possible keys that one has to try. Reportedly, former Mafia boss Bernardo Provenzano used Caesar's code to communicate with associates while he was a fugitive. (It didn't work too well for him.)

The obvious flaw of such kind of substitution ciphers is the very small number of possible keys, so that an adversary can simply try all of them.

Substitution codes in which the permutation is allowed to be arbitrary were used through the middle ages and modern times. In a 26-letter alphabet, the number of keys is 26!, which is too large for a brute-force attack. Such systems, however, suffer from easy total breaks because of the facts that, in any given language, different letters appear with different frequencies, so that Eve can immediately make good guesses for what are the encryptions of the most common letters, and work out the whole code with some trial and errors. This was noticed already in the 9th century A.D. by Arab scholar al-Kindy. Sherlock Holmes breaks a substitution cipher in *The Adventure of the Dancing Men*.

For fun, try decoding the following message. (A permutation over the English alphabet has been applied; spaces have been removed before encoding.)

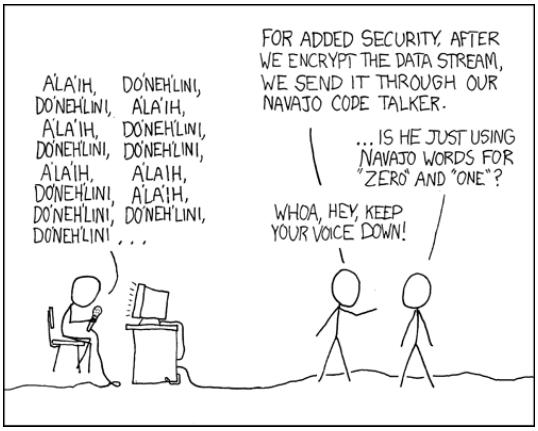
IKNHQHNWKZHTHNHPZKTPKAZYASNKOOAVHNPSAETKOHQHNCH
HZSKBZRHYKBRCBRNHIBOHYRKCHXZKSXHYKBRAZYIKNHQHNWK
ZHETKTAOORBVCFHYCBRORKKYNPDTRCASXBLAZYIKNHQHNWK
ZHETKEKNXOTANYAZYZHQHNDPQHOBLRTPOKZHPOIKNWKBWKB
XZKEETARRTHWOAWAOKTPKDKHOOKDKHORTHZARPKZEHFFRT
POZARPKZOSKVPZDCASXAZYOKPORTPOSAVLAPDZRTHLHKLFHK
IKTPKTAQHOAPYPRFKBYFWAZYSFHANFWEHNHDKPZDKZEHNHDK
PZDORNKZDAZYEHNHDKPZDAFFRTHEAWWKBXZKERTHWSAFFKT
PKACHFFEHRTHNORARHPROACARRFHDNKBZYORARHPROAORA
RHRTARXZKEOTKERKLPSXALNHOPYHZRAZYZKSAZYPYARHPZNH
SHZRTPORKNWYHVKSNARKNNHLBCFPSAZTAOEKZRTHETPRHTK
BOHEPRTKBREPZZPZDRTHKTPKLNPVANW

Other substitution ciphers were studied, in which the code is based on a permutation over Σ^t, where Σ is the alphabet and t a small integers. (For example, the code would specify a permutation over 5-tuples of characters.) Even such systems suffer from (more sophisticated) frequency analysis.

Various tricks have been conceived to prevent frequency analysis, such as changing the permutation at each step, for example by combining it with a cyclic shift permutation. (The German Enigma machines used during WWII used multiple permutations, and applied different shift on each application.)

More generally, however, most classic methods suffer from the problem of being *deterministic* encryption schemes: If the same message is sent twice, the encryptions will be the same. This can be disastrous when the code is used with a (known) small set of possible messages

This xkcd cartoon makes this point very aptly.



(The context of the cartoon is that, reportedly, during WWII, some messages were encrypted by translating them into the Navajo language, the idea being that there was no Navajo speaker outside of North America. As the comic shows, even though this could be a very hard permutation to invert without the right secret information, this is useless if the set of encrypted messages is very small.)

Look also at the pictures of the two encodings of the Linux penguin on the Wikipedia page on block ciphers.

Here is an approach that has large key space, which prevents single-character frequency analysis, and which is probabilistic.

Alice and Bob have agreed on a permutation P of the English alphabet $\Sigma = \{A, \ldots, Z\}$, and they think of it as a group, for example by identifying Σ with $\mathbb{Z}/26\mathbb{Z}$, the integers mod 26.

When Alice has a message $m_1 \cdots m_k$ to send, she first picks a random letter r, and then she produces an encryption $c_0, c_1, \ldots, c_k$ by setting $c_0 = r$ and $c_i := P(c_{i-1} + m_i)$. Then Bob will decode $c_0, \ldots, c_k$ by setting $m_i := P^{(-1)}(c_i) - c_{i-1}$.

Unfortunately, this method suffers from *two-character* frequency analysis. You might try to

amuse yourselves by decoding the following ciphertext (encoded with the above described method):

HTBTOOWCHEZPWDVTBYQWHFDBLEDZTESGVFO
SKPOTWILEJQBLSOYZGLMVALTQGVTBYQPLHAKZ
BMGMGDWSTEMHNBVHMZXERHJQBEHNKPOMJDP
DWJUBSPIXYNNRSJQHAKXMOTOBIMZTWEJHHCFD
BMUETCIXOWZTWFIACZLRVLTQPDBDMFPUSPFYW
XFZXXVLTQPABJFHXAFTNUBBJSTFHBKOMGYXGKC
YXVSFRNEDMQVBSHBPLHMDOOYMVWJSEEKPILOB
AMKMXPPTBXZCNNIDPSNRJKMRNKDFQZOMRNFQZ
OMRNF

As we shall see later, this idea has merit if used with an *exponentially big* permutation, and this fact will be useful in the design of actual secure encryption schemes.

1.3 Perfect Security and One-Time Pad

Note that if Alice only ever sends one one-letter message m, then just sending $P(m)$ is completely secure: regardless of what the message m is, Eve will just see a random letter $P(m)$. That is, the distribution (over the choice of the secret key P) of encodings of a message m is *the same* for all messages m, and thus, from the point of view of Eve, the encryption is *statistically independent* of the message.

This is an ideal notion of security: basically Eve might as well not be listening to the communication, because the communication gives no information about the message. The same security can be obtained using a key of $\log 26$ bits (instead of $\log 26!$ as necessary to store a random permutation) by Alice and Bob sharing a random letter r, and having Alice send $m + r$.

In general, is Alice wants to send a message $m \in \Sigma^k$, and Alice and Bob share a random secret $r \in \Sigma^k$, then it is perfectly secure as above to send $m_1 + r_1, \ldots, m_k + r_k$.

This encoding, however, can be used *only once* (think of what happens when several messages are encoded using this process with the same secret key) and it is called *one-time pad*. It has, reportedly, been used in several military and diplomatic applications.

The inconvenience of one-time pad is that Alice and Bob need to agree in advance on a key as large as the total length of all messages they are ever going to exchange. Obviously, your laptop cannot use one-time pad to communicate with your base station.

Shannon demonstrated that perfect security requires this enormous key length. Without getting into the precise result, the point is that if you have an n-bit message and you use a k-bit key, $k < n$, then Eve, after seeing the ciphertext, knows that the original message is one of 2^k possible messages, whereas without seeing the ciphertext she only knew that it was one of 2^n possible messages.

When the original message is written, say, in English, the consequence of short key length can be more striking. English has, more or less, one bit of entropy per letter which means

(*very* roughly speaking) that there are only about 2^n meaningful n-letter English sentences, or only a $(1/13)^n$ fraction of all $(26)^n$ possible n-letter strings. Given a ciphertext encoded with a k-bit key, Eve knows that the original message is one of 2^k possible messages. Chances are, however, that only about $2^k \cdot (13)^{-n}$ such messages are meaningful English sentences. If k is small enough compared to n, Eve can uniquely reconstruct the original message. (This is why, in the two examples given above, you have enough information to actually reconstruct the entire original message.)

When $n >> k$, for example if we use an 128-bit key to encrypt a 4GB movie, virtually all the information of the original message is available in the encryption. A brute-force way to use that information, however, would require to try all possible keys, which would be infeasible even with moderate key lengths. Above, we have seen two examples of encryption in which the key space is fairly large, but efficient algorithms can reconstruct the plaintext. Are there always methods to *efficiently* break any cryptosystem?

We don't know. This is equivalent to the question of whether one-way functions exist, which is probably an extremely hard question to settle. (If, as believed, one-way functions do exist, proving their existence would imply a proof that $P \neq NP$.)

We shall be able, however, to prove the following dichotomy: either one-way functions do not exist, in which case any approach to essentially any cryptographic problem is breakable (with exceptions related to the one-time pad), or one-way functions exist, and then all symmetric-key cryptographic problems have solutions with extravagantly strong security guarantees.

Next, we'll see how to formally define security for symmetric-key encryption, and how to achieve it using various primitives.

Lecture 2

Notions of Security

In the last lecture we saw that

- all classical encryption schemes which allow the encryption of arbitrarily long messages have fatal flaws;
- it is possible to encrypt with perfect security using one-time pad, but the scheme can be used only once, and the key has to be as long as the message;
- if one wants perfect security, one needs a key as long as the total length of all messages that are going to be sent.

Our goal for the next few lectures will be to study schemes that allow the sending of messages that are essentially arbitrarily long, using a fixed key, and having a security that is essentially as good as the perfect security of one-time pad.

Today we introduce a notion of security (*semantic security*) that is extremely strong. When it is met *there is no point for an adversary to eavesdrop the channel*, regardless of what messages are being sent, of what she already knows about the message, and what goal she is trying to accomplish.

2.1 Semantic Security

First, let us fix the model in which we are going to work. For the time being, we are going to be very modest, and we shall only try to construct an encryption scheme that, like one-time pad, is designed for only one use. We just want the key to be reasonably short and the message to be of reasonably large length.

We shall also restrict ourselves to *passive* adversaries, meaning that Eve is able to see the communication between Alice and Bob, but she cannot inject her own messages in the channel, and she cannot prevent messages from being delivered.

The definition of *correctness* for an *encryption scheme* is straightforward.

Definition 1 (Symmetric-Key Encryption Scheme – Finite case) *A symmetric-key encryption scheme with key-length k, plain-text length m and ciphertext-length c is is a pair of probabilistic algorithms (Enc, Dec), such that $Enc : \{0,1\}^k \times \{0,1\}^m \to \{0,1\}^c$, $Dec : \{0,1\}^k \times \{0,1\}^c \to \{0,1\}^m$, and for every key $K \in \{0,1\}^k$ and every message M,*

$$\mathbb{P}[Dec(K, Enc(K, M)) = M] = 1 \tag{2.1}$$

where the probability is taken over the randomness of the algorithms

Definition 2 (Symmetric-Key Encryption Scheme – Variable Key Length Case) *A symmetric-key encryption scheme with variable key-length is is a pair of polynomial-time probabilistic algorithms (Enc, Dec) and a function $m(k) > k$, such that for every security parameter k, for every key $K \in \{0,1\}^k$ and every message $M \in \{0,1\}^{m(k)}$,*

$$\mathbb{P}[Dec(k, K, Enc(k, K, M)) = M] = 1 \tag{2.2}$$

(Although it is redundant to give the algorithm the parameter k, we do so because this will emphasize the similarity with the public-key setting that we shall study later.)

It will be more tricky to satisfactorily formalize the notion of *security*.

One super-strong notion of security, which is true for the one-time pad is the following:

Definition 3 (Perfect Security) *A symmetric-key encryption scheme (Enc, Dec) is perfectly secure if, for every two messages M, M', the distributions $Enc(K, M)$ and $Enc(K, M')$ are identical, where we consider the distribution over the randomness of the algorithm $Enc()$ and over the choice of $K \sim \{0,1\}^k$.*

The informal discussion from the previous lecture gives a hint to how to solve the following

Exercise 1 *Prove that if (Enc, Dec) is perfectly secure, then $k \geq m$.*

Before we move on, let us observe two limitations that will be present in any possible definitions of security involving a key of length k which is much smaller than the message length. Eve can always employ one of the following two trivial attacks:

1. In time 2^k, Eve can enumerate all keys and produce a list of 2^k plaintexts, one of which is correct. Further considerations can help her prune the list, and in certain attack models (which we shall consider later), she can figure out with near certainty which one is right, recover the key and totally break the system.
2. Eve can make a random guess of what the key is, and be correct with probability 2^{-k}.

Already if $k = 128$, however, neither line of attack is worrisome for Alice and Bob. Even if Eve has access to the fastest of super-computers, Alice and Bob will be long dead of old

age before Eve is done with the enumeration of all 2^{128} keys; and Alice and Bob are going to both going to be stricken by lighting, and then both hit by meteors, with much higher probability than 2^{-128}.

The point, however, is that *any* definition of security will have to involve a bound on Eve's running time, and allow for a low probability of break. If the bound on Eve's running time is enormous, and the bound on the probability of a break is minuscule, then the definition is as satisfactory as if the former was infinite and the latter was zero.

All the definitions that we shall consider involve a bound on the complexity of Eve, which means that we need to fix a model of computation to measure this complexity. We shall use (non-uniform) *circuit complexity* to measure the complexity of Eve, that is measure the number of gates in a boolean circuit implementing Eve's functionality.

If you are not familiar with circuit complexity, the following other convention is essentially equivalent: we measure the running time of Eve (for example on a RAM, a model of computation that captures the way standard computers work) and we *add the length of the program* that Eve is running. The reason is that, without this convention, we would never be able to talk about the complexity of computing finite functions. Every function of an 128-bit input, for example, is very efficiently computable by a program which is nothing but a series of 2^{128} if-then-elses.

Finally we come to our first definition of security:

Definition 4 (Message Indistinguishability – concrete version) *We say that an encryption scheme (Enc, Dec) is (t, ϵ) message indistinguishable if for every two messages M, M', and for every boolean function T of complexity $\leq t$, we have*

$$|\,\mathbb{P}[T(Enc(K, M) = 1] - \mathbb{P}[T(Enc(K, M')) = 1]| \leq \epsilon \tag{2.3}$$

where the probability is taken over the randomness of $Enc()$ and the choice of $K \sim \{0,1\}^k$.

(Typical parameters that are considered in practice are $t = 2^{80}$ and $\epsilon = 2^{-60}$.)

When we have a family of ciphers that allow varying key lengths, the following asymptotic definition is standard.

Definition 5 (Negligible functions) *A function $\nu : \mathbb{N} \to \mathbb{R}^+$ is negligible if for every polynomial p and for every sufficiently large n*

$$\nu(n) \leq \frac{1}{p(n)}$$

Definition 6 (Message Indistinguishability – asymptotic definition) *We say that a variable key length encryption scheme (Enc, Dec) is message indistinguishable if for every polynomial p there is a negligible function ν such that for all sufficiently large k the scheme is $(p(k), \nu(k))$-message indistinguishable when the security parameter is k.*

The motivation for the asymptotic definition, is that we take polynomial time to be an upper bound to the amount of steps that any efficient computation can take, and to the "number of events" that can take place. This is why we bound Eve's running time by a polynomial. The motivation for the definition of negligible functions is that if an event happens with negligible probability, then the expected number of experiments that it takes for the event to happen is superpolynomial, so it will "never" happen. Of course, in practice, we would want the security parameters of a variable-key scheme to be exponential, rather than merely super-polynomial.

Why do we use message indistinguishability as a formalization of security?

A first observation is that if we take $\epsilon = 0$ and put no limit on t, then message-indistinguishability becomes perfect security, so at least we are dealing with a notion whose "limit" is perfect security.

A more convincing explanation is that message indistinguishability is equivalent to *semantic security*, a notion that we describe below and that, intuitively, says that Eve might as well not look at the channel.

What does it mean that "Eve might as well not look at the channel"? Let us summarize Eve's information and goals. Alice has a message M that is sent to Bob over the channel. The message comes from some distribution X (for example, it is written in English, in a certain style, it is about a certain subject, and so on), and let's assume that Eve knows X. Eve might also know more about the specific message being sent, because of a variety of reasons; call $I(M)$ the information that Eve has about the message. Finally, Eve has a goal in eavesdropping the conversation, which is to learn some information $f(M)$ about the message. Perhaps she wants to reconstruct the message in its entirety, but it could also be that she is only interested in a single bit. (Does M contain the string "I hate Eve"? Is M a confidential report stating that company Y is going to miss its earning estimate? And so on.)

Why is Eve even bothering tapping the channel? Because via some cryptoanalytic algorithm A, which runs in a reasonable amount of time, she thinks she has a good chance to accomplish. But the probability of accomplishing her goal would have been essentially the same *without* tapping the channel, then there is no point.

Definition 7 (Semantic Security – Concrete definition) *An encryption scheme (Enc, Dec) is (t, o, ϵ) semantically secure if for every distribution X over messages, every functions $I : \{0,1\}^m \rightarrow \{0,1\}^*$ and $f : \{0,1\}^m \rightarrow \{0,1\}^*$ (of arbitrary complexity) and every function A of complexity $t_A \leq t$, there is a function A' of complexity $\leq t_A + o$ such that*

$$|\,\mathbb{P}[A(Enc(K, M), I(m)) = f(M)] - \mathbb{P}[A'(I(m)) = f(M)]| \leq \epsilon$$

Think, as before, of $t = 2^{80}$ and $\epsilon = 2^{-60}$, and suppose o is quite small (so that a computation of complexity o can be performed in a few seconds or less), and notice how the above definition captures the previous informal discussion.

Now let's see that semantic security is *equivalent* to message indistinguishability.

Lemma 8 (Semantic Security Implies Message Indistinguishability) *If* (Enc, Dec) *is* (t, o, ϵ) *semantically secure, then it is* $(t, 2\epsilon)$ *message indistinguishable.*

Note that semantic security implies message indistinguishability *regardless of the overhead parameter* o.

PROOF: We prove that if (Enc, Dec) is *not* $(t, 2\epsilon)$ message indistinguishable then it is *not* (t, o, ϵ) semantically secure regardless of how large is o.

If (Enc, Dec) is not $(t, 2\epsilon)$ message indistinguishable, then there are two messages M_0, M_1 and an algorithm T of complexity $\leq t$ such that

$$\mathbb{P}[T(Enc(K, M_1)) = 1] - \mathbb{P}[T(Enc(K, M_0)) = 1] > 2\epsilon \tag{2.4}$$

Pick a bit b uniformly at random in $\{0, 1\}$; then we have

$$\mathbb{P}[T(Enc(K, M_b)) = b] > \frac{1}{2} + \epsilon \tag{2.5}$$

And now take A to be T, X to be the distribution M_b for a random b, and define $f(M_b) = b$, and $I(M)$ to be empty. Then

$$\mathbb{P}[A(I(M), Enc(K, M)) = f(M)] > \frac{1}{2} + \epsilon \tag{2.6}$$

On the other hand, for every A', regardless of complexity

$$\mathbb{P}[A'(I(M), Enc(K, M)) = f(M)] = \frac{1}{2} \tag{2.7}$$

and so we contradict semantic security. □

Lemma 9 (Message Indistinguishability Implies Semantic Security) *If* (Enc, Dec) *is* (t, ϵ) *message indistinguishable and Enc has complexity* $\leq p$*, then* (Enc, Dec) *is* $(t - \ell_f, p, \epsilon)$ *semantically secure, where* ℓ_f *is the maximum length of* $f(M)$ *over* $M \in \{0, 1\}^m$.

PROOF: Fix a distribution X, an information function I, a goal function f, and a cryptoanalytic algorithm A of complexity $\leq t - \ell_f$.

Take $A'(I(M)) = A(I(M), Enc(K, \mathbf{0}))$, so that the complexity of A' is equal to the complexity of A plus the complexity of Enc.

For every message M, we have

$$\mathbb{P}[A(I(M), Enc(K, M)) = f(M)] \leq \mathbb{P}[A(I(M), Enc(K, \mathbf{0})) = f(M)] + \epsilon \tag{2.8}$$

Otherwise defining $T(C) = 1 \Leftrightarrow A(I(M), C) = f(M)$ would contradict the indistinguishabiltiy.

Averaging over M in X

$$\mathop{\mathbb{P}}_{M\sim X,K\in\{0,1\}^n}[A(I(M),Enc(K,M))=f(M)] \leq \mathop{\mathbb{P}}_{M\sim X,K\in\{0,1\}^n}[A(I(M),Enc(K,\mathbf{0}))=f(M)]+\epsilon \tag{2.9}$$

and so

$$\mathop{\mathbb{P}}_{M\sim X,K\in\{0,1\}^n}[A(I(M),Enc(K,M))=f(M)] \leq \mathop{\mathbb{P}}_{M\sim X,K\in\{0,1\}^n}[A'(I(M))=f(M)]+\epsilon \tag{2.10}$$

□

It is also possible to define an asymptotic version of semantic security, and to show that it is equivalent to the asymptotic version of message indistinguishability.

Definition 10 (Semantic Security – Asymptotic Definition) *An encryption scheme (Enc, Dec) is semantically secure if for every polynomial p there exists a polynomial q and a negligible function ν such that (Enc, Dec) is $(p(k), q(k), \nu(k))$ semantically secure for all sufficiently large k.*

Exercise 2 *Prove that a variable key-length encryption scheme (Enc, Dec) is asymptotically semantically secure if and only if it is asymptotically message indistinguishable,*

2.2 Security for Multiple Encryptions: Plain Version

In the real world, we often need to send more than just one message. Consequently, we have to create new definitions of security for such situations, where we use the same key to send multiple messages. There are in fact multiple possible definitions of security in this scenario. Today we shall only introduce the simplest definition.

Definition 11 (Message indistinguishability for multiple encryptions) (Enc, Dec) *is (t,ϵ)-message indistinguishable for c encryptions if for every $2c$ messages $M_1,\ldots,M_c$, $M'_1,\ldots,M'_c$ and every T of complexity $\leq t$ we have*

$$\begin{aligned}&|\,\mathbb{P}[T(Enc(K,M_1),\ldots,Enc(K,M_c))=1]\\ &-\mathbb{P}[T(Enc(K,M'_1),\ldots,Enc(K,M'_c))=1]|\leq\epsilon\end{aligned}$$

Similarly, we define semantic security, and the asymptotic versions.

Exercise 3 *Prove that no encryption scheme (Enc, Dec) in which $Enc()$ is deterministic (such as the scheme for one-time encryption described above) can be secure even for 2 encryptions.*

Encryption in some versions of Microsoft Office is deterministic and thus fails to satisfy this definition. (This is just a symptom of bigger problems; the schemes in those versions of Office are considered completely broken.)

If we allow the encryption algorithm to keep *state* information, then a pseudorandom generator is sufficient to meet this definition. Indeed, usually pseudorandom generators designed for such applications, including RC4, are optimized for this kind of "stateful multiple encryption."

2.3 Security against Chosen Plaintext Attack

In realistic scenarios, an adversary has knowledge of plaintext-ciphertext pairs. A broadly (but not fully) general way to capture this knowledge is to look at a model in which the adversary is able to see *encryptions of arbitrary messages of her choice*. An attack in this model is called a Chosen Plaintext Attack (CPA). This model at least captures the situation when the messages which were initially secret have been made public in course of time and hence some plaintext ciphertext pairs are available to the adversary. Using new primitives called *pseudorandom functions* and *pseudorandom permutations*, it is possible to construct encryption schemes that satisfy this notion of security. In fact, an even stronger notion of security where adversary is allowed to see decryptions of certain chosen messages can also be constructed using pseudorandom functions but it will take us some time to develop the right tools to analyze a construction meeting this level of security. How do we construct pseudorandom functions and permutations? It is possible to construct them from pseudorandom generators (and hence from one-way functions), and there are ad-hoc constructions which are believed to be secure. For the time being, we define security under CPA and show how it generalizes the notion of multiple encryption security given in the last section.

If O is a, possibly randomized, procedure, and A is an algorithm, we denote by $A^O(x)$ the computation of algorithm A given x as an input and given the ability to execute O. We charge just one unit of time for every execution of O, and we refer to A as having *oracle access* to O.

Definition 12 (Message indistinguishability against CPA) *(Enc, Dec) is (t, ϵ)-message indistinguishable against CPA if for every 2 messages M, M' and every T of complexity $\leq t$ we have*

$$|\,\mathbb{P}[T^{Enc(K,\cdot)}(Enc(K, M)) = 1] - \mathbb{P}[T^{Enc(K,\cdot)}(Enc(K, M')) = 1]| \leq \epsilon$$

We now prove that it is a generalization of security for multiple encryptions (as defined in the last section)

Lemma 13 *Suppose (Enc, Dec) is (t, ϵ)-message indistinguishable against CPA. Then for every c it is $(t - cm, c\epsilon)$-message indistinguishable for c encryptions.*

PROOF: We prove by contradiction i.e. we assume that there exist a pair of c messages $(M_1, M_2, \ldots, M_{c-1}, M_c)$ and $(M'_1, M'_2, \ldots, M'_{c-1}, M'_c)$ such that there is a procedure T' of complexity $(t - cm)$ which can distinguish between these two with probability greater than $c\epsilon$. We shall prove existence of two messages M and M' and an oracle procedure of complexity $\leq t$ such that

$$|\,\mathbb{P}[T^{Enc(K,\cdot)}(Enc(K,M)) = 1] - \mathbb{P}[T^{Enc(K,\cdot)}(Enc(K,M')) = 1]| > \epsilon$$

We shall assume a circuit model rather than the Turing machine model of computation here. We first start with a simple case i.e. let there exist pair of c messages $M_1, M_2, \ldots, M_{c-1}, M_c$ and $M'_1, M'_2, \ldots, M'_{c-1}, M'_c$ such that the sequence differs exactly at one position say i i.e. $M_j = M'_j$ for $j \neq i$ and $M_i \neq M'_i$. By assumption, we can say that

$$|\,\mathbb{P}[T'(Enc(K,M_1), \ldots, Enc(K,M_c) = 1] - \mathbb{P}[T'^{(}Enc(K,M'_1), \ldots, Enc(K,M'_c)) = 1]| > c\epsilon$$

The machine T on being given input C simulates machine T' as

$$T'(Enc(K,M_1), \ldots, Enc(K,M_{i-1}), C, Enc(K,M_{i+1}), \ldots, Enc(K,M_c))$$

using the oracle to know $Enc(K,M_j)$ for $j \neq i$. Clearly by assumption, T can distinguish between $Enc(K,M_i)$ and $Enc(K,M'_i)$ with probability more than $c\epsilon > \epsilon$. Also, the hard-wiring of the messages M_j for $j \neq i$ increases the circuit complexity by at most cm (there are c of them and each is of length at most m). Hence T is a circuit of complexity at most t. To move to the general case where the sequences differ at more than one place, we use the hybrid argument which is a staple of proofs in cryptography. We shall use it here as follows. Consider c-tuple D_a defined as -

$$D_a = Enc(K,M_1), \ldots, Enc(K,M_{c-a}), Enc(K,M'_{c-a+1}), \ldots, Enc(K,M'_c)$$

Then, we have that $|\,\mathbb{P}[T'(D_0) = 1] - \mathbb{P}[T'(D_c) = 1]| > c\epsilon$ which can be rewritten as $|\sum_{a=0}^{c-1} \mathbb{P}[T'(D_a) = 1] - \mathbb{P}[T'(D_{a+1}) = 1]| > c\epsilon$. By triangle inequality, we get that there exists some j such that $|\,\mathbb{P}[T'(D_j)] - \mathbb{P}[T'(D_{j+1})]| > \epsilon$. However, note that D_j and D_{j+1} differ at exactly one position, a case which we have already solved. The only difference is that the distinguishing probability is ϵ rather than $c\epsilon$ because of introduction of the hybrid argument. $\square$

Lecture 3

Pseudorandom Generators

Summary

We discuss the notion of *pseudorandom generator*, and see that it is precisely the primitive that is needed in order to have message-indistinguishable (and hence semantically secure) one-time encryption.

But how do we construct a pseudorandom generator? We can't if $P = NP$, so the security of any construction will have to rely on an unproved assumption which is at least as strong as $P \neq NP$. We shall see, later on, how to construct a pseudorandom generator based on well-established assumptions, such as the hardness of integer factorization, and we shall see that the weakest assumption under which we can construct pseudorandom generators is the *existence of one-way functions*.

Today, we shall instead look at RC4, a simple candidate pseudorandom generator designed by Ron Rivest. RC4 is very efficient, and widely used in practice – for example in the WEP standard for wireless communication. It is known to be insecure in its simplest instantiation (which makes WEP insecure too), but there are variants that may be secure.

This gives a complete overview of one-time symmetric-key encryption: from a rigorous definition of security to a practical construction that may plausibly satisfy the definition.

3.1 Pseudorandom Generators And One-Time Encryption

Intuitively, a Pseudorandom Generator is a function that takes a short random string and stretches it to a longer string which is almost random, in the sense that reasonably complex algorithms cannot differentiate the new string from truly random strings with more than negligible probability.

Definition 14 (Pseudorandom Generator) *A function $G : \{0,1\}^k \to \{0,1\}^m$ is a (t, ϵ)-secure pseudorandom generator if for every boolean function T of complexity at most*

t we have

$$\left| \mathbb{P}_{x \sim U_k}[T(G(x)) = 1] - \mathbb{P}_{x \sim U_m}[T(x) = 1] \right| \leq \epsilon \tag{3.1}$$

(We use the notation U_n for the uniform distribution over $\{0,1\}^n$.)

The definition is interesting when $m > k$ (otherwise the generator can simply output the first m bits of the input, and satisfy the definition with $\epsilon = 0$ and arbitrarily large t). Typical parameters we may be interested in are $k = 128$, $m = 2^{20}$, $t = 2^{60}$ and $\epsilon = 2^{-40}$, that is we want k to be very small, m to be large, t to be huge, and ϵ to be tiny. There are some unavoidable trade-offs between these parameters.

Lemma 15 *If $G : \{0,1\}^k \to \{0,1\}^m$ is $(t, 2^{-k-1})$ pseudorandom with $t = O(m)$, then $k \geq m - 1$.*

Proof: Pick an arbitrary $y \in \{0,1\}^k$. Define

$$T_y(x) = 1 \Leftrightarrow x = G(y)$$

It is clear that we may implement T with an algorithm of complexity $O(m)$: all this algorithm has to do is store the value of $G(y)$ (which takes space $O(m)$) and compare its input to the stored value (which takes time $O(m)$) for total complexity of $O(m)$. Now, note that

$$\mathbb{P}_{x \sim U_k}[T(G(x)) = 1] \geq \frac{1}{2^k}$$

since $G(x) = G(y)$ at least when $x = y$. Similarly, note that $\mathbb{P}_{x \sim U_m}[T(x) = 1] = \frac{1}{2^m}$ since $T(x) = 1$ only when $x = G(y)$. Now, by the pseudorandomness of G, we have that $\frac{1}{2^k} - \frac{1}{2^m} \leq \frac{1}{2^{k+1}}$. With some rearranging, this expression implies that

$$\frac{1}{2^{k+1}} \leq \frac{1}{2^m}$$

which then implies $m \leq k + 1$ and consequently $k \geq m - 1$ □

Exercise 4 *Prove that if $G : \{0,1\}^k \to \{0,1\}^m$ is (t, ϵ) pseudorandom, and $k < m$, then*

$$t \cdot \frac{1}{\epsilon} \leq O(m \cdot 2^k)$$

Suppose we have a pseudorandom generator as above. Consider the following encryption scheme:

- Given a key $K \in \{0,1\}^k$ and a message $M \in \{0,1\}^m$,

$$Enc(K, M) := M \oplus G(K)$$

- Given a ciphertext $C \in \{0,1\}^m$ and a key $K \in \{0,1\}^k$,

$$Dec(K, C) = C \oplus G(K)$$

(The XOR operation is applied bit-wise.)

It's clear by construction that the encryption scheme is correct. Regarding the security, we have

Lemma 16 *If G is (t, ϵ)-pseudorandom, then (Enc, Dec) as defined above is $(t - m, 2\epsilon)$-message indistinguishable for one-time encryption.*

PROOF: Suppose that G is not $(t-m, 2\epsilon)$-message indistinguishable for one-time encryption. Then $\exists$ messages M_1, M_2 and $\exists$ algorithm T of complexity at most $t - m$ such that

$$\left| \mathop{\mathbb{P}}_{K \sim U_k}[T(Enc(K, M_1)) = 1] - \mathop{\mathbb{P}}_{K \sim U_k}[T(Enc(K, M_2)) = 1] \right| > 2\epsilon$$

By using the definition of Enc we obtain

$$\left| \mathop{\mathbb{P}}_{K \sim U_k}[T(G(K) \oplus M_1)) = 1] - \mathop{\mathbb{P}}_{K \sim U_k}[T(G(K) \oplus M_2)) = 1] \right| > 2\epsilon$$

Now, we can add and subtract the term $\mathbb{P}_{R \sim U_m}[T(R) = 1]$ and use the triangle inequality to obtain that $|\mathbb{P}_{K \sim U_k}[T(G(K) \oplus M_1) = 1] - \mathbb{P}_{R \sim U_m}[T(R) = 1]|$ added to $|\mathbb{P}_{R \sim U_m}[T(R) = 1] - \mathbb{P}_{K \sim U_k}[T(G(K) \oplus M_2)$ is greater than 2ϵ. At least one of the two terms in the previous expression must be greater that ϵ. Suppose without loss of generality that the first term is greater than ϵ

$$\left| \mathop{\mathbb{P}}_{K \sim U_k}[T(G(K) \oplus M_1)) = 1] - \mathop{\mathbb{P}}_{R \sim U_m}[T(R) = 1] \right| > \epsilon$$

Now define $T'(X) = T(X \oplus M_1)$. Then since $H(X) = X \oplus M_1$ is a bijection, $\mathbb{P}_{R \sim U_m}[T'(R) = 1] = \mathbb{P}_{R \sim U_m}[T(R) = 1]$. Consequently,

$$\left| \mathop{\mathbb{P}}_{K \sim U_k}[T'(G(K)) = 1] - \mathop{\mathbb{P}}_{R \sim U_m}[T'(R) = 1] \right| > \epsilon$$

Thus, since the complexity of T is at most $t - m$ and T' is T plus an xor operation (which takes time m), T' is of complexity at most t. Thus, G is not (t, ϵ)-pseudorandom since there exists an algorithm T' of complexity at most t that can distinguish between G's output and random strings with probability greater than ϵ. Contradiction. Thus (Enc, Dec) is $(t - m, 2\epsilon)$-message indistinguishable. $\square$

3.2 Description of RC4

We now present the description of RC4, a very simple candidate pseudorandom generator. This was proposed by Ron Rivest. Though there are some theoretical considerations in the choice of encryption scheme, we shall not be going into it. Below we give a slightly generalized description of RC4.

Fix a modulus s, which is 256 in RC4, and let $\mathbb{Z}_s$ be the finite group of s elements $\{0, \ldots, s-1\}$ together with the operation of addition mod s. (The notation $\mathbb{Z}/s\mathbb{Z}$ is more common in math.)

The generator has two phases:

The first phase is intended to construct a "pseudorandom" permutation. Before, we go into the actual construction used in RC4, we first give a description of how to construct a nearly random permutation given a sufficiently long seed. Note that the problem is non-trivial because even if we are given a very long random string and we interpret the string as a function in the obvious way, then it may not be a permutation. Hence, we need to try something more clever. A nearly random permutation may be created in the following way. Let K be a random element in $(\{0,1\}^8)^{256}$ which we may interpret as an array of 256 numbers each in the range $[0, 255]$. In particular, let $K(a)$ represent the a^{th} element. To create a permutation over 256 elements, do the following (id represents the identity permutation i.e. $id(x) = x$)

- $P := id$
- For $a \in \mathbb{Z}_{256}$
  - Swap $P(a)$ and $P(K(a))$

What distribution over permutations do we generate with the above process? The answer is not completely understood, but the final permutation is believed to be close to uniformly distributed.

However, the above process is inefficient in the amount of randomness required to create a random permutation. We now describe a process which is more randomness efficient i.e. uses a smaller key to construct a permutation (which shall now be "pseudorandom", although this is not meant as a technical term; the final permutation will be distinguishable from a truly random permutation). The seed $K \in \{0,1\}^k = (\{0,1\}^{\log_2 s})^t$ (interpreted as an array over elements in $\mathbb{Z}_s$ of length t) is converted into a permutation $P : \mathbb{Z}_s \to \mathbb{Z}_s$ as follows (the variables a, b are in $\mathbb{Z}_s$ and so addition is performed mod s):

- $P := id$
- $b := 0$
- for a in $\{0, \ldots s-1\}$:
  - $b := b + P(a) + K[a \bmod t]$

  - swap $(P(a), P(b))$

(Note that if $k = s \log_2 s$ then the first phase has the following simpler description: for each $a \in \mathbb{Z}_s$, swap $P(a)$ with a random location, as in the simplified random process.)

In the second phase, the permutation is used to produce the output of the generator as follows:

- $a := 0$; $b := 0$
- for $i := 1$ to m :
  - $a := a + 1$
  - $b = b + P(a)$
  - output $P(P(a) + P(b))$
  - swap $(P(a), P(b))$

In RC4, s is 256, as said before, which allows extremely fast implementations, and k is around 100.

The construction as above is known to be insecure: the second byte has probability 2^{-7} instead of 2^{-8} of being the all-zero byte which violates the definition of pseudorandom generator.

There are other problems besides this bias, and it is possible to reconstruct the key and completely break the generator given a not-too-long sequence of output bits. WEP uses RC4 as described above, and is considered completely broken.

If one discards an initial prefix of the output, however, no strong attack is known. A conservative recommendation is to drop the first 4096 bits.

Lecture 4

Encryption Using Pseudorandom Functions

Summary

Having introduced the notion of CPA security in the past lecture, we shall now see constructions that achieve it. Such constructions shall require either *pseudorandom functions* or *pseudorandom permutations*. We shall see later how to construct such objects.

4.1 Pseudorandom Functions

To understand the definition of a pseudorandom function, it's good to think of it as a pseudorandom generator whose output is *exponentially long*, and such that each bit of the output is efficiently computable given the seed. The security is against efficient adversaries that are allowed to look at at any subset of the exponentially many output bits.

Definition 17 (Pseudorandom Function) *A function $F : \{0,1\}^k \times \{0,1\}^m \to \{0,1\}^m$ is a (t, ϵ)-secure pseudorandom function if for every oracle algorithm T that has complexity at most t we have*

$$|\mathop{\mathbb{P}}_{K\in\{0,1\}^k}[T^{F_K}() = 1] - \mathop{\mathbb{P}}_{R:\{0,1\}^m\to\{0,1\}^m}[T^R() = 1]| \leq \epsilon$$

Intuitively, this means that an adversary wouldn't be able to distinguish outputs from a purely random function and a pseudorandom function (upto a certain ϵ additive error). Typical parameters are $k = m = 128$, in which case security as high as $(2^{60}, 2^{-40})$ is conjectured to be possible.

As usual, it is possible to give an asymptotic definition, in which $\epsilon(k)$ is required to be negligible, $t(k)$ is allowed to be any polynomial, and F is required to be computable in polynomial time.

4.2 Encryption Using Pseudorandom Functions

Suppose $F : \{0,1\}^k \times \{0,1\}^m \to \{0,1\}^m$ is a pseudorandom function. We define the following encryption scheme.

- $Enc(K, M)$: pick a random $r \in \{0,1\}^m$, output $(r, F_K(r) \oplus M)$
- $Dec(K, (C_0, C_1)) := F_K(C_0) \oplus C_1$

This construction achieves CPA security.

Theorem 18 *Suppose F is a (t, ϵ) secure pseudorandom function. Then the above scheme is $(\frac{t}{O(m)}, 2\epsilon + t \cdot 2^{-m})$-secure against CPA.*

The proof of Theorem 18 will introduce another key idea that will often reappear in this course: to first pretend that our pseudorandom object is truly random, and perform our analysis accordingly. Then extend the analysis from the pseudorandom case to the truly random case.

Let us therefore consider a modified scheme $(\overline{Enc}, \overline{Dec})$, where instead of performing $F_K(r) \oplus M$, we do $R(r) \oplus M$, where $R : \{0,1\}^m \to \{0,1\}^m$ is a truly random function. We need to look at how secure this scheme is. In fact, we will actually prove that

Lemma 19 *$(\overline{Enc}, \overline{Dec})$ is $(t, \frac{t}{2^m})-CPA$ secure.*

Proof:

In the computation $T^{\overline{Enc}}(\overline{Enc}(r, C))$ of algorithm T given oracle $\overline{Enc}$ and input the ciphertext (r, C), let us define REPEAT to be the event where T gets the messages $(r_1, C_1), \ldots, (r_t, C_t)$ from the oracle, such that r equals one of the r_i.

Then we have

$$\begin{aligned} \mathbb{P}[T^{\overline{Enc}}(\overline{Enc}(M)) = 1] &= \mathbb{P}[T^{\overline{Enc}}(\overline{Enc}(M)) = 1 \wedge REPEAT] \\ &\quad + \mathbb{P}[T^{\overline{Enc}}(\overline{Enc}(M)) = 1 \wedge \neg REPEAT] \end{aligned}$$

similarly,

$$\begin{aligned} \mathbb{P}[T^{\overline{Enc}}(\overline{Enc}(M')) = 1] &= \mathbb{P}[T^{\overline{Enc}}(\overline{Enc}(K, M')) = 1 \wedge REPEAT] \\ &\quad + \mathbb{P}[T^{\overline{Enc}}(\overline{Enc}(M')) = 1 \wedge \neg REPEAT] \end{aligned}$$

so

$$
\begin{aligned}
&|\,\mathbb{P}[T^{\overline{Enc}}(\overline{Enc}(K,M))=1]-\mathbb{P}[T^{\overline{Enc}}(\overline{Enc}(K,M'))=1]|\leq\\
&\quad|\,\mathbb{P}[T^{\overline{Enc}}(\overline{Enc}(M))=1\wedge REPEAT]-\mathbb{P}[T^{\overline{Enc}}(\overline{Enc}(M'))=1\wedge REPEAT]|+\\
&\quad|\,\mathbb{P}[T^{\overline{Enc}}(\overline{Enc}(M))=1\wedge\neg REPEAT]-\mathbb{P}[T^{\overline{Enc}}(\overline{Enc}(M'))=1\wedge\neg REPEAT]|
\end{aligned}
$$

Now the first difference is the difference between two numbers which are both between 0 and $P[REPEAT]$, so it is at most $P[REPEAT]$, which is at most $\frac{t}{2^m}$.

The second difference is zero, because with a purely random function there is a 1-1 mapping between every random choice (of $R, r, r_1, \ldots, r_t$) which makes the first event happen and every random choice that makes the second event happen. □

We have shown that with a purely random function, the above encryption scheme is CPA-secure. We can now turn our eyes to the pseudorandom scheme (Enc, Dec), and prove Theorem 18.

PROOF: Consider the following four probabilities, for messages M, M', and algorithm T :

1. $\mathbb{P}_K[T^{Enc(K,\cdot)}(Enc(K,M))=1]$
2. $\mathbb{P}_K[T^{Enc(K,\cdot)}(Enc(K,M'))=1]$
3. $\mathbb{P}_R[T^{\overline{Enc}(\cdot)}(\overline{Enc}(M))=1]$
4. $\mathbb{P}_R[T^{\overline{Enc}(\cdot)}(\overline{Enc}(M'))=1]$

From the previous proof, we have $|3-4|\leq\frac{t}{2^m}$. If we are able to show that $|1-3|\leq\epsilon$, $|2-4|\leq\epsilon$, then we have $|1-2|\leq 2\epsilon+\frac{t}{2^m}$.

So, it remains to show that

$$
|\,\mathop{\mathbb{P}}_{K}[T^{Enc(K,\cdot)}(Enc(K,M))=1]-\mathop{\mathbb{P}}_{R}[T^{\overline{Enc}(\cdot)}(\overline{Enc}(M))=1]|\leq\epsilon \tag{4.1}
$$

Suppose, by contradiction, this is not the case. We will show that such a contradiction implies that F is not secure, by constructing an oracle algorithm T' that distinguishes F from a truly random function.

For an oracle G, we define T'^G to be the following algorithm:

- pick a random $r\in\{0,1\}^m$ and compute $C:=(r,G(r)\oplus M)$
- simulate $T(C)$; every time C makes an oracle query M_i, pick a random r_i and respond to the query with $(r_i, G(r_i)\oplus M)$

Note that if T' is given the oracle F_K, then the computation T'^{F_K} is exactly the same as the computation $T^{Enc}(Enc(M))$, and if T' is given the oracle R, where R is a random function, then the computation $T^{\overline{Enc}}(\overline{Enc}(M))$.

Thus, we have

$$\mathop{\mathbb{P}}_{K\in\{0,1\}^k}[T'^{F_K}() = 1] = \mathop{\mathbb{P}}_{K}[T^{Enc(K,\cdot)}(Enc(K,M)) = 1] \tag{4.2}$$

$$\mathop{\mathbb{P}}_{R:\{0,1\}^m\to\{0,1\}^m}[T'^{R}() = 1] = \mathop{\mathbb{P}}_{R}[T^{\overline{Enc}(\cdot)}(\overline{Enc}(M)) = 1] \tag{4.3}$$

which means that

$$\left|\mathop{\mathbb{P}}_{K\in\{0,1\}^k}[T'^{F_K}() = 1] - \mathop{\mathbb{P}}_{R:\{0,1\}^m\to\{0,1\}^m}[T'^{R}() = 1]\right| > \epsilon \tag{4.4}$$

The complexity of T' is at most the complexity of T times $O(m)$ (the time needed to translate between oracle queries of T and oracle queries of T'), and so if T has complexity $t/O(m)$ then T' has complexity $\leq t$. This means that (4.4) contradicts the assumption that F is $(t,\epsilon)-$secure. □

4.3 The Randomized Counter Mode

Recall that a pseudorandom function is a function $F\colon \{0,1\}^k \times \{0,1\}^m \to \{0,1\}^m$ which looks approximately like a random function $R\colon \{0,1\}^m \to \{0,1\}^m$. With the encryption method from the previous lecture (in which the ciphertext is a random $r \in \{0,1\}^m$ followed by $F_K(r)\oplus M$) the encryption of a message is twice as long as the original message. We now define an encryption method which continues to use a pseudorandom function, but whose ciphertext overhead is marginal.

Suppose we have a pseudorandom function $F\colon \{0,1\}^k \times \{0,1\}^m \to \{0,1\}^m$. We describe an encryption scheme that works for messages of variable length. We assume without loss of generality that the length of the message is a multiple of m, and we write a plaintext M of length cm as $M_1,\ldots,M_c$, a sequence of c blocks of length m.

- $Enc(K, M_1,\ldots,M_c)$:
  - pick a random $r \in \{0,1\}^m$
  - output

$$(r, F_K(r)\oplus M_1, F_K(r+1)\oplus M_2,\ldots,F_K(r+(c-1))\oplus M_c)$$

- $Dec(K, C_0,\ldots,C_c) := C_1 \oplus F_K(C_0),\ldots,C_c \oplus F_K(C_0+(c-1))$

(When r is a binary string in $\{0,1\}^m$ and i is an integer, $r+i$ means the binary representation of the sum mod 2^m of r (seen as an integer) and i.)
Observe that the ciphertext length is $(c+1)m$ which is a negligable overhead when $c \gg m$.

Theorem 20 *Suppose F is a (t, ϵ)-secure pseudorandom function; then, when used to encrypt messages of length cm, the above scheme is $(t - O(cm), O(\epsilon + ct/2^m))$-CPA secure.*

Example 21 *Consider the values which these variables might take in the transmission of a large (e.g. > 4GB) file. If we let $m = 128$, $t = 2^{60}$, $\epsilon = 2^{-60}$, $c = 2^{30}$, then we end up with an approximately $(2^{59}, 2^{-38})$-CPA secure transmission.*

PROOF: Recall the proof from last time in which we defined $\overline{Enc}(R, \cdot)$, where R is a truly random function. Given messages M, M' and a cryptanalytic algorithm T, we considered:

- (a) $\mathbb{P}_K[T^{Enc(K,\cdot)}(Enc(K, M)) = 1]$
- (b) $\mathbb{P}_R[T^{\overline{Enc}(R,\cdot)}(\overline{Enc}(R, M)) = 1]$
- (c) $\mathbb{P}_R[T^{\overline{Enc}(R,\cdot)}(\overline{Enc}(R, M')) = 1]$
- (d) $\mathbb{P}_K[T^{Enc(K,\cdot)}(Enc(K, M')) = 1]$

We were able to show in the previous proof that $|(a) - (b)| \leq \epsilon$, $|(c) - (d)| \leq \epsilon$, and $|(b) - (c)| \leq t/2^m$, thus showing that $|(a) - (d)| \leq 2\epsilon + t/2^m$. Our proof will follow similarly.

We will first show that for any M

$$\left| \mathop{\mathbb{P}}_{K}[T^{Enc(K,\cdot)}(Enc(K, M)) = 1] - \mathop{\mathbb{P}}_{R}[T^{\overline{Enc}(R,\cdot)}(\overline{Enc}(R, M)) = 1] \right| \leq \epsilon$$

hence showing $|(a) - (b)| \leq \epsilon$ and $|(c) - (d)| \leq \epsilon$. Suppose for a contradiction that this is not the case, i.e. $\exists M = (M_1, \ldots, M_c)$ and $\exists T$ where T is of complexity $\leq t - O(cm)$ such that

$$\left| \mathop{\mathbb{P}}_{K}[T^{Enc(K,\cdot)}(Enc(K, M)) = 1] - \mathop{\mathbb{P}}_{R}[T^{\overline{Enc}(R,\cdot)}(\overline{Enc}(R, M)) = 1] \right| > \epsilon$$

Define $T'^{O(\cdot)}(\cdot)$ as a program which simulates $T(O(M))$. (Note that T' has complexity $\leq t$). Noting that $T'^{Enc(K,\cdot)}() = T^{Enc(K,\cdot)}(Enc(K, M))$ and $T'^{\overline{Enc}(R,\cdot)}() = T^{\overline{Enc}(R,\cdot)}(\overline{Enc}(R, M))$, this program T' would be a counterexample to F being (t, ϵ)-secure.

Now we want to show that $\forall M = M_1, \ldots, M_c$, $\forall M' = M'_1, \ldots, M'_c$, and $\forall T$ such that the complexity of T is $\leq t - O(cm)$,

$$\left| \mathbb{P}_R[T^{\overline{Enc}(R,\cdot)}(\overline{Enc}(R, M)) = 1] - \mathbb{P}_R[T^{\overline{Enc}(R,\cdot)}(\overline{Enc}(R, M')) = 1] \right| \leq 2ct/2^m$$

As in the previous proof, we consider the requests T may make to the oracle $\overline{Enc}(R, \cdot)$. The returned values from the oracle would be $r_k, R(r_k) \oplus M_1^k, R(r_k + 1) \oplus M_2^k, \ldots, R(r_k + (c-1)) \oplus M_c^k$, where k ranges between 1 and the number of requests to the oracle. Since T has complexity limited by t, we can assume $1 \leq k \leq t$. As before, if none of the $r_k + i$ overlap with $r + j$ (for $1 \leq i, j \leq c$) then T only sees a random stream of bits from the oracle. Otherwise, if $r_k + i = r + j$ for some i, j, then T can recover, and hence distinguish,

M_j and M'_j. Hence the probability of T distinguishing M, M' is ϵ plus the probability of a collision.

Note that the kth oracle request will have a collision with some $r+j$ iff $r-c < r_k \leq r+(c-1)$. If we have $r \leq r_k \leq r+(c-1)$ then obviously there is a collision, and otherwise $r-c < r_k < r$ so $r-1 < r_k + (c-1) \leq r + (c-1)$ so there is a collision with $r_k + (c-1)$. If r_k is outside this range, then there is no way a collision can occur. Since r_k is chosen randomly from the space of 2^m, there is a $(2c-1)/2^m$ probability that the kth oracle request has a collision. Hence $2ct/2^m$ is an upper bound on the probability that there is a collision in at least one the oracle requests.

Combining these results, we see that $|(a) - (d)| \leq 2(\epsilon + ct/2^m) = O(\epsilon + ct/2^m)$, i.e.

$$\left|\mathbb{P}_K[T^{Enc(K,\cdot)}(Enc(K,M)) = 1] - \mathbb{P}_K[T^{Enc(K,\cdot)}(Enc(K,M')) = 1]\right| = O(\epsilon + ct/2^m)$$

□

Lecture 5

Encryption Using Pseudorandom Permutations

Summary

We give the definition of *pseudorandom permutation*, which is a rigorous formalization of the notion of *block cipher* from applied cryptography, and see two ways of using block ciphers to perform encryption. One is totally insecure (ECB), the other (CBC) achieves CPA security.

5.1 Pseudorandom Permutations

5.1.1 Some Motivation

Suppose the message stream has known messages, such as a protocol which always has a common header. For example, suppose Eve knows that Bob is sending an email to Alice, and that the first block of the message M_1 is the sender's email. That is, suppose Eve knows that M_1 ="bob@cs.berkeley.edu". If Eve can insert or modify messages on the channel, then upon seeing the ciphertext $C_0, \ldots, C_c$ she could then send to Alice the stream $C_0, C_1 \oplus$ "bob@cs.berkeley.edu" $\oplus$ "eve@cs.berkeley.edu" $, C_2, \ldots, C_c$. The result is that the message received by Alice would appear to be sent from "eve@cs.berkeley.edu", but remain otherwise unchanged.

5.1.2 Definition

Denote by $\mathcal{P}_n$ the set of permutations $P\colon \{0,1\}^n \to \{0,1\}^n$.

Definition 22 *A pair of functions $F\colon \{0,1\}^k \times \{0,1\}^n \to \{0,1\}^n$, $I\colon \{0,1\}^k \times \{0,1\}^n \to \{0,1\}^n$ is a (t,ϵ)-secure pseudorandom permutation if:*

- *For every $r \in \{0,1\}^k$, the functions $F_r(\cdot)$ and $I_r(\cdot)$ are permutations (i.e. bijections) and are inverses of each other.*
- *For every oracle algorithm T that has complexity at most t*

$$\left|\mathbb{P}_K[T^{F_K,I_K}() = 1] - \mathbb{P}_{P\in\mathcal{P}_n}[T^{P,P^{-1}}() = 1]\right| \leq \epsilon$$

That is, to any algorithm T that doesn't know K, the functions F_K, I_K look like a random permutation and its inverse.

In applied cryptography literature, pseudorandom permutations are called *block ciphers*.

How do we construct pseudorandom permutations? There are a number of block cipher proposals, including the AES standard, that have been studied extensively and are considered safe for the time being. We shall prove later that any construction of pseudorandom functions can be turned into a construction of pseudorandom permutations; also, every construction of pseudorandom generators can be turned into a pseudorandom function, and every one-way function can be used to construct a pseudorandom generator. Ultimately, this will mean that it is possible to construct a block cipher whose security relies, for example, on the hardness of factoring random integers. Such a construction, however, would not be practical.

5.2 The AES Pseudorandom Permutation

AES is a pseudorandom permutation with $m = 128$ and $k = 128$, 192 or 256. It was the winner of a competition run by NIST between 1997 and 2000 to create a new encryption standard to replace DES (which had $k = 56$ and was nearly broken by that time).

The conjectured security of practical pseudorandom permutations such as AES does not rely on the hardness of a well-defined computational problem, but rather on a combination of design principles and of an understanding of current attack strategies and of methods to defy them.

AES keeps a state, which is initially equal to the input, which is a 4×4 matrix of bytes. The state is processed in 4 stages. This processing is done 10, 12, or 14 times (depending on key length), and the final state is the output.

1. In the first stage, a 128-bit string derived from the key (and dependent on the current round) is added to the state. This is the only stage that depends on the key;
2. A fixed bijection (which is part of the specification of AES) $p : \{0,1\}^8 \to \{0,1\}^8$ is applied to each byte of the state
3. The rows of the matrix are shifted (row i is shifted $i - 1$ places)
4. An invertible linear transformation (over the field $GF(256)$) is applied to the matrix

The general structure is common to other conjecturally secure pseudorandom permutations:

- There is one (or more) small "random-like" permutations that are hard-wired in the construction, such as $p : \{0,1\}^8 \to \{0,1\}^8$ in AES. Traditionally, those hard-wired functions are called "S-boxes."
- A "key-scheduler" produces several "pseudorandom" strings from the key. (Usually, the scheduler is not a true pseudorandom generator, but does something very simple.)
- The construction proceeds in several rounds. At each round there is some combination of:
  - "Confuse:" apply the hard-wired S-boxes locally to the input (Stage 2 in AES)
  - "Diffuse:" rearrange bits so as to obscure the local nature of the application of the S-boxes (Stages 3 and 4 in AES)
  - "Randomize:" use a string produced by the key-scheduler to add key-dependent randomness to the input (Stage 1 in AES)

5.3 Encryption Using Pseudorandom Permutations

Here are two ways of using Pseudorandom Functions and Permutations to perform encryption. Both are used in practice.

5.3.1 ECB Mode

The Electronic Code-Book mode of encryption works as follows

- $Enc(K, M) := F_K(M)$
- $Dec(K, M) := I_K(M)$

Exercise 5 *Show that ECB is message-indistinguishable for one-time encryption but not for two encryptions.*

5.3.2 CBC Mode

In its simplest instantiation the Cipher Block-Chaining mode works as follows:

- $Enc(K, M)$: pick a random string $r \in \{0,1\}^n$, output $(r, F_K(r \oplus M))$
- $Dec(K, (C_0, C_1)) := C_0 \oplus I_K(C_1)$

Note that this similar to (but a bit different from) the scheme based on pseudorandom functions that we saw last time. In CBC, we take advantage of the fact that F_K is now a permutation that is efficiently invertible given the secret key, and so we are allowed to put the $\oplus M$ inside the computation of F_K.

There is a generalization in which one can use the same random string to send several messages. (It requires synchronization and state information.)

- $Enc(K, M_1, \ldots, M_c)$:
  - pick a random string $C_0 \in \{0,1\}^n$
  - output $(C_0, C_1, \ldots, C_c)$ where $C_i := F_K(C_{i-1} \oplus M_i)$
- $Dec(K, C_0, C_1, \ldots, C_c) := M_1, \ldots, M_c$ where $M_i := I_K(C_i) \oplus C_{i-1}$

Exercise 6 *This mode achieves CPA security.*

Note that CBC overcomes the above problem in which Eve knows a particular block of the message being sent, for if Eve modified C_1 in the encryption that Bob was sending to Alice (as in the example above) then the change would be noticeable because $C_2, \ldots, C_c$ would not decrypt correctly.

Lecture 6

Authentication

Summary

Today we start to talk about *message authentication codes* (MACs). The goal of a MAC is to guarantee to the recipient the integrity of a message and the identity of the sender. We provide a very strong definition of security (*existential unforgeability under adaptive chosen message attack*) and show how to achieve it using pseudorandom functions.

Our solution will be secure, but inefficient in terms of length of the required authentication information.

Next time we shall see a more space-efficient authentication scheme, and we shall prove that given a CPA-secure encryption scheme and a secure MAC, one can get a CCA-secure encryption scheme. (That is, an encryption scheme secure against an *adaptive chosen ciphertext and plaintext attack*.)

6.1 Message Authentication

The goal of message authentication is for two parties (say, Alice and Bob) who share a secret key to ensure the integrity and authenticity of the messages they exchange. When Alice wants to send a message to Bob, she also computes a *tag*, using the secret key, which she appends to the message. When Bob receives the message, he *verifies* the validity of the tag, again using the secret key.

The syntax of an authentication scheme is the following.

Definition 23 (Authentication Scheme) *An authentication scheme is a pair of algorithms* $(Tag, Verify)$*, where* $Tag(\cdot,\cdot)$ *takes in input a key* $K \in \{0,1\}^k$ *and a message* M *and outputs a tag* T*, and* $Verify(\cdot,\cdot,\cdot)$ *takes in input a key, a message, and a tag, and outputs a boolean answers. We require that for every key* K*, and very message* M

$$Verify(K, M, Tag(K, M)) = True$$

if $Tag(\cdot,\cdot)$ is deterministic, and we require

$$\mathbb{P}[Verify(K, M, Tag(K, M)) = True] = 1$$

if $Tag(\cdot,\cdot)$ is randomized.

In defining security, we want to ensure that an adversary who does not know the private key is unable to produce a valid tag. Usually, an adversary may attempt to forge a tag for a message after having seen other tagged messages, so our definition of security must ensure that seeing tagged messages does not help in producing a forgery. We provide a very strong definition of security by making sure that the adversary is able to tag *no* new messages, even after having seen tags of any other messages of *her choice*.

Definition 24 (Existential unforgeability under chosen message attack) *We say that an authentication scheme $(Tag, Verify)$ is (t, ϵ)-secure if for every algorithm A of complexity at most t*

$$\mathop{\mathbb{P}}_{K}[A^{Tag(K,\cdot)} = (M, T) : (\mathrm{M}, \mathrm{T}) \text{ is a forge}] \leq \epsilon$$

where a pair (M, T) is a "forge" if $Verify(K, M, T) = True$ and M is none of the messages that A queried to the tag oracle.

This definition rules out any possible attack by an active adversary except a *replay* attack, in which the adversary stores a tagged message it sees on the channel, and later sends a copy of it. We still are guaranteed that any message we see was sent at some time by the right party. To protect against *replay* attacks, we could include a timestamp with the message, and reject messages that are too old. We'll assume that *replay* attacks are handled at a higher level and will not worry about them.

6.2 Construction for Short Messages

Suppose $F : \{0,1\}^k \times \{0,1\}^m \to \{0,1\}^m$ is a pseudorandom function. A simple scheme is to use the pseudorandom function as a tag:

- $Tag(K, M) := F_K(M)$
- $Verify(K, M, T) := True$ if $T = F_K(M)$, $False$ otherwise

This construction works only for short messages (of the same length as the input of the pseudorandom function), but is secure.

Theorem 25 *If F is a (t, ϵ)-secure pseudorandom function, then the above construction is a $(t - O(m), \epsilon + 2^{-m})$-secure authentication scheme.*

PROOF: First, let $R : \{0,1\}^m \to \{0,1\}^m$ be a truly random function, and $A^{R(\cdot)}()$ an algorithm with oracle access to $R(\cdot)$ and complexity at most t. Then

$$\mathop{\mathbb{P}}_{R(\cdot)}[A^{R(\cdot)}() = (M,T) : M,T\, is\, a\, forgery] = \mathop{\mathbb{P}}_{R(\cdot)}[R(M) = T] = 2^{-m}.$$

Now, define an algorithm $A'^{O(\cdot)}$ that returns 1 iff $O(M) = T$, where M, T are the values computed by $A^{O(\cdot)}()$. Then

$$\begin{aligned} &|\,\mathbb{P}[A^{R(\cdot)}\, is\, a\, forgery] - \mathbb{P}[A^{F_K(\cdot)}\, is\, a\, forgery]| \\ &= |\,\mathbb{P}[A'^{R(\cdot)} = 1] - \mathbb{P}[A'^{F_K(\cdot)} = 1]| \\ &\leq \epsilon \end{aligned}$$

Where the last inequality is due to the definition of a pseudo-random function. From this it follows that

$$\begin{aligned} \mathbb{P}[A^{F_K(\cdot)}\, is\, a\, forgery] &\leq \mathbb{P}[A^{R(\cdot)}()\, is\, a\, forgery] \\ &\quad +|\,\mathbb{P}[A^{R(\cdot)}\, is\, a\, forgery] - \mathbb{P}[A^{F_K(\cdot)}\, is\, a\, forgery]| \\ &\leq 2^{-m} + \epsilon \end{aligned}$$

□

6.3 Construction for Messages of Arbitrary Length

Suppose we now have a longer message M, which we write as $M := M_1, \ldots, M_\ell$ with each block M_i being of the same length as the input of a given pseudorandom function.

There are various simple constructions we described in class that do not work. Here are some examples:

Example 26 *$Tag(K, M) := F_K(M_1), \ldots, F_K(M_\ell)$. This authentication scheme allows the advisary to rearrange, repeate, or remove blocks of the message. Therefore it is insecure.*

Example 27 *$Tag(K, M) := F_K(1, M_1), \ldots, F_K(\ell, M_\ell)$. This authentication scheme prevents the advisary from reordering blocks of the message, but it still allows the advisary to truncate the message or to interleave blocks from two previously seen messages.*

Example 28 *$Tag(K, M) := r, F_K(r, 1, M_1), \ldots, F_K(r, \ell, M_\ell)$. This scheme adds a randomized message identifier, and it prevents interleaving blocks from different messages, but it still fails to protect the message from being truncated by the advisary.*

The following construction works:

Let $F : \{0,1\}^k \times \{0,1\}^m \to \{0,1\}^m$ be a pseudorandom function, M be the message we want to tag, and write $M = M_1, \ldots, M_\ell$ where each block M_i is $m/4$ bits long.

- $Tag(K, M)$:
  - Pick a random $r \in \{0,1\}^{m/4}$,
  - output $r, T_1, \ldots, T_\ell$, where

$$T_i := F_K(r, \ell, i, M_i)$$

- $Verify(K, (M_1, \ldots, M_\ell), (r, T_1, \ldots, T_\ell))$:
  - Output $True$ if and only if $T_i = F_K(r, \ell, i, M_i)$

Theorem 29 *If F is (t, ϵ)-secure, the above scheme is $(\Omega(t), \epsilon + t^2 \cdot 2^{-m/4} + 2^{-m})$-secure.*

PROOF: Define (T, V) as the authentication scheme above. Define $(\bar{T}, \bar{V})$ in the same way, except using a truly random function R in place of F_K. Let A be an algorithm of complexity at most t.

Consider $A^{\bar{T}}$. Note that A can make at most t oracle queries. Define `FORGE` as the event in which $A^{\bar{T}}$ never queries M, and produces a tag T for which $\bar{V}(M, T) = yes$. Define `REP` as the event in which, for two different oracle queries, A receives tags with same r.

Now,

$$\begin{aligned}
\mathbb{P}[\texttt{REP}] &= \mathbb{P}[\exists \textit{ a repetition among } r_1, \ldots, r_t] \\
&\leq \sum_i \sum_j \mathbb{P}[r_i = r_j] \\
&= t^2 2^{-m/4}
\end{aligned}$$

Consider the event $\texttt{FORGE} \wedge \neg\texttt{REP}$. Suppose our oracle queries, and the resulting random strings, were:

$$\begin{aligned}
M_1^1, \ldots, M_{l_1}^1 &\to r^1 \\
M_1^2, \ldots, M_{l_2}^2 &\to r^2 \\
&\ldots
\end{aligned}$$

$$\begin{aligned}
M_1^1, \ldots, M_{l_1}^1 &\to r^1 \\
M_1^2, \ldots, M_{l_2}^2 &\to r^2 \\
&\ldots
\end{aligned}$$

Then we know $i \neq j \Rightarrow r^i \neq r^j$. Now, the algorithm outputs message

$$M_1, \ldots, M_\ell$$

with a valid tag

$$r, T_1, \ldots, T_\ell$$

Then there are the following cases:

- Case1: $r \neq r^i \forall i$. Then the algorithm computed $T_1 = R(r, \ell, 1, M_1)$ without having seen it before.
- Case2: r was seen before, so it occured exactly once, in the Tag for the j^{th} query.
  - Case 2a: $\ell_j \neq \ell$. Then we computed $T_1 = R(r, \ell, 1, M_1)$ without having seen it before.
  - Case 2b: $\ell_j = \ell$. We know $M \neq M^j$, so $\exists i : M_i^j \neq M_i$. thus we computed $T_i = R(r, \ell, i, M_i)$ without having seen it before

Thus, in the event $\texttt{FORGE} \wedge \neg\texttt{REP}$, we constructed some $T_i = R(r, \ell, i, M_i)$ without sending (r, ℓ, i, M_i) to the oracle. Since R is truely random, this can only occur with prbability 2^{-m}.

Now,

$$\begin{aligned}
\mathbb{P}[A^{\bar{T}}\, is\, a\, forgery] &= \mathbb{P}[\texttt{FORGE}] \\
&= \mathbb{P}[\texttt{FORGE} \wedge \texttt{REP}] + \mathbb{P}[\texttt{FORGE} \wedge \neg\texttt{REP}] \\
&\leq \mathbb{P}[\texttt{REP}] + \mathbb{P}[\texttt{FORGE} \wedge \neg\texttt{REP}] \\
&\leq t^2 2^{-m/4} + 2^{-m}
\end{aligned}$$

So finally we have

$$\begin{aligned}
\mathbb{P}[A^T() \, is\, a\, forgery] &\leq |\, \mathbb{P}[A^T() \, is\, a\, forgery] - \mathbb{P}[A^{\bar{T}}() \, is\, a\, forgery]| + \mathbb{P}[A^{\bar{T}}() \, is\, a\, forgery] \\
&\leq \epsilon + t^2 2^{-m/4} + 2^{-m}
\end{aligned}$$

□

Lecture 7

CCA-Secure Encryption

Summary

Last time we described a secure MAC (message authentication code) based on pseudorandom functions. Its disadvantage was the length of the tag, which grew with the length of the message.

Today we describe the CBC-MAC, also based on pseudorandom functions, which has the advantage of short tags. We skip its security analysis.

Next, we show that combining a CPA-secure encryption with a secure MAC gives a CCA-secure encryption scheme.

7.1 CBC-MAC

Suppose we have a pseudorandom function $F : \{0,1\}^k \times \{0,1\}^m \to \{0,1\}^m$.

Last time we described a provably secure MAC in which a message M is broken up into blocks $M_1, \ldots, M_\ell$, each of length $m/4$, and the tag of M is the sequence

$$(r, F_K(r, \ell, 1, M_1), F_K(r, \ell, 2, M_2), \ldots, F_K(r, \ell, \ell, M_\ell))$$

where r is a random string and K is the key of the authentication scheme. Jonah suggested a more compact scheme, in which M is broken into blocks $M_1, \ldots, M_\ell$ of length $m/3$ and the tag is

$$(r, F_K(r, 0, 1, M_1), F_K(r, 0, 2, M_2), \ldots, F_K(r, 1, \ell, M_\ell))$$

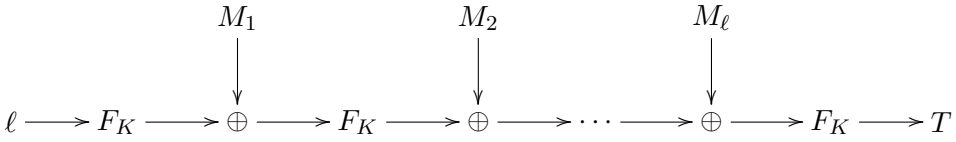
for a random string r. That is, the length of the message is not explicitly authenticated in each block, but we authenticate a single bit that says whether this is, or isn't, the last block of the message.

Exercise 7 *Prove that if F is (t, ϵ)-secure then this scheme is $(t/O(\ell m), \epsilon + t^2 \cdot 2^{-m/3} + 2^{-m})$-secure, where ℓ is an upper bound to the number of blocks of the message that we are going to authenticate.*

A main disadvantage of such schemes is the length of the final tag.

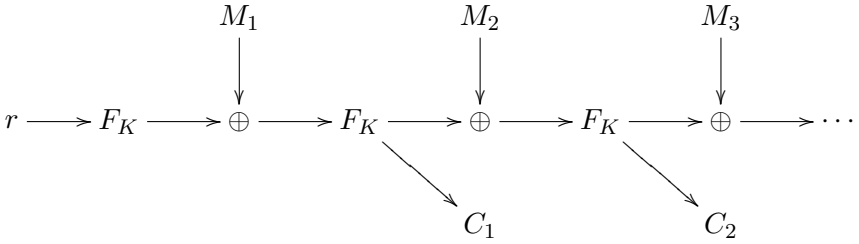
The CBC-MAC scheme has the advantage of producing a tag whose length is only m.

CBC-MAC scheme:



- $Tag(K, M_1, \ldots, M_\ell)$:
  - $T_0 := F_K(\ell)$
  - for $i := 1$ to ℓ: $T_i := F_K(T_{i-1} \oplus M_i)$
  - return T_ℓ
- $Verify(K, M, T)$: check that $Tag(K, M) == T$

This scheme is similar to in structure to CBC encryption:



We will not prove CBC-MAC to be secure, but the general approach is to show that all the inputs to F_K are distinct with high probability.

7.2 Combining MAC and Encryption

Suppose that we have an encryption scheme (E, D) and a MAC (T, V). We can combine them to produce the following encryption scheme, in which a key is made of pair (K_1, K_2) where K_1 is a key for (E, D) and K_2 is a key for (T, V):

- $E'((K_1, K_2), M)$:
  - $C := E(K_1, M)$
  - $T := T(K_2, C)$
  - return (C, T)

- $D'((K_1, K_2), (C, T))$:
  - if $V(K_2, C, T)$: return $D(K_1, C)$
  - else return ERROR

The scheme (E', D') is an encrypt-then-authenticate scheme in which we first encrypt the plaintext with key K_1 and then authenticate the ciphertext with key K_2. The decryption aborts if given an incorrectly tagged ciphertext.

The idea of this scheme is that an adversary mounting a CCA attack (and hence having access to both an encryption oracle and a decryption oracle) has *no use* for the decryption oracle, because the adversary *already knows* the answer that the decryption oracle is going to provide for each oracle query:

1. if the adversary queries a ciphertext previously obtained from the encryption oracle, then it already knows the corresponding plaintext
2. if the adversary queries a ciphertext not previously obtained from the encryption oracle, then almost surely (assuming the security of the MAC), the tag in the ciphertext will be incorrect, and the oracle answer is going to be "ERROR"

This intuition is formalized in the proof of the following theorem.

Theorem 30 *If (E, D) is (t, ϵ) CPA secure, and (T, V) is $(t, \epsilon/t)$ secure, then (E', D') is $(t/(r + O(\ell)), 3\epsilon)$ CCA secure, where r is an upper bound to the running time of the encryption algorithm E and the tag algorithm T, and ℓ is an upper bound to the length of the messages that we encrypt.*

PROOF: Suppose (E', D') is not CCA-secure. Then there exist an algorithm A' of complexity $t' \leq t/(r + O(\ell))$ and two messages M_1 and M_2 such that

$$\begin{aligned} &|\Pr[A'^{E'_{(K_1,K_2)}, D'_{(K_1,K_2)}}(E'_{(K_1,K_2)}(M_1)) = 1] \\ &\quad - \Pr[A'^{E'_{(K_1,K_2)}, D'_{(K_1,K_2)}}(E'_{(K_1,K_2)}(M_2)) = 1]| > 3\epsilon. \end{aligned} \tag{\#}$$

Without loss of generality, we assume A' never queries D' on any ciphertext previously returned by E'. We can make this assumption because we can modify A' to keep a record of all the queries it makes to E', and to use the record to avoid redundant queries to D'.

We now wish to convert A' to a new algorithm A_1 such that

$$\forall M \ \Pr_K[A_1^{E_K}(E_K(M)) = 1] \approx \Pr_{K_1,K_2}[A'^{E'_{(K_1,K_2)}, D'_{(K_1,K_2)}}(E'_{(K_1,K_2)}(M))].$$

Note that A' is given the oracles E' and D', but A_1 is given as an oracle just the original CPA-secure encryption algorithm E.

Define

- $A_1^E(C)$:
  - pick a random key K_2'
  - $T := T(K_2', C)$
  - simulate $A'^{O_1,O_2}(C,T)$ with these oracles:
    - $O_1(M)$ returns $E'((K_1,K_2'),M)$;
    - O_2 always returns ERROR.

A_1 has to run the tagging algorithm T, which has complexity r, every time A' makes an oracle call. Since A' has complexity at most t/r, A_1 has complexity at most t.

Now, assuming the attack A' works, we can apply the triangle inequality to (#) to obtain:

$$\begin{aligned} 3\epsilon < &|\Pr[A'^{E',D'}(E'(M_1))=1] - \Pr[A_1^E(E(M_1))=1]| && \text{(A)}\\ &+|\Pr[A_1^E(E(M_1)=1] - \Pr[A_1^E(E(M_2)=1]| && \text{(B)}\\ &+|\Pr[A_1^E(E(M_2)=1] - \Pr[A'^{E',D'}(E'(M_2)=1]|. && \text{(C)} \end{aligned}$$

One of (A), (B) and (C) must be greater than ϵ.

If (B) $> \epsilon$, then algorithm A_1 breaks the CPA-security of E. We assumed E was CPA-secure, so one of (A) and (B) must be greater than ϵ. In either case, there exists a message M with the property that

$$|\Pr_K[A_1^E(E(M))=1] - \Pr_{K_1,K_2}[A'^{E',D'}(E'(M))=1]| > \epsilon. \tag{7.1}$$

If when A_1 is simulating A', A' never makes a call to D' which results in an output other than "ERROR", then A_1 behaves exactly as A' would with the same key K_2. So (7.1) implies that with probability greater than ϵ, $A'^{E',O'}(E'(M))$ makes a call to the decryption oracle resulting in an output other than "ERROR". This means A' manages to generate valid messages that it has never seen before, and we can use this fact to define an algorithm A_2 that breaks the Message Authentication Code (T,V).

$A'^{E',D'}$ makes at most t oracle queries to D', and with probability ϵ, at least one of those results in an output other than "ERROR". There must exist a number i such that with probability at least ϵ/t, the i-th query $A'^{E',D'}$ makes to D' is the first one that does not result in an error. Then define algorithm A_2 as follows.

- A_2^T (no input)
  - choose a random key K_1
  - $C := E(K_1, M)$
  - $T := T(C)$
  - simulate $A'^{O_1,O_2}(C,T)$, with the following two oracles...
    - $O_1(M_1)$:
      - $C := E(K_1, M_1)$

- $T := T(C)$
- return (C, T)

$*$ O_2 always returns ERROR

...until A' makes its i-th query to O_2

– Let (C_i, T_i) be the i-th query A' made to O_2.

– return (C_i, T_i)

Note that A_2 has complexity at most t, and by our analysis of algorithm A', A_2^T produces a correct tage for a message it has never seen before with probability at least ϵ/t.

Since we assumed (T, V) was $(T, \epsilon/t)$-secure, we have reached a contradiction: (E', D') is therefore CCA secure. $\square$

Lecture 8

Collision-Resistant Hash Functions

Summary

Last time, we showed that combining a CPA-secure encryption with a secure MAC gives a CCA-secure encryption scheme. Today we shall see that such a combination has to be done carefully, or the security guarantee on the combined encryption scheme will not hold.

We then begin to talk about *cryptographic hash functions*, and their construction via the *Merkle-Damgård transform*.

8.1 Combining Encryption and Authentication

8.1.1 Encrypt-Then-Authenticate

Let (E, D) be an encryption scheme and (Tag, V) be a MAC.

Last time we considered their *encrypt-then-authenticate* combination defined as follows:

Construction (E_1, D_1)

- Key: a pair (K_1, K_2) where K_1 is a key for (E, D) and K_2 is a key for (Tag, V)
- $E_1((K_1, K_2), M)$:
  - $C := E(K_1, M)$
  - $T := Tag(K_2, C)$
  - return (C, T)
- $D_1((K_1, K_2), (C, T))$:
  - if $V(K_2, C, T)$ then return $D(K_1, C)$
  - else return 'ERROR'

and we proved that if (E, D) is CPA-secure and (Tag, V) is existentially unforgeable under a chosen message attack then (E_1, D_1) is CCA-secure.

Such a result is not provable if (E, D) and (Tag, V) are combined in different ways.

8.1.2 Encrypt-And-Authenticate

Consider the following alternative composition:

Construction (E_2, D_2)

- Key: a pair (K_1, K_2) where K_1 is a key for (E, D) and K_2 is a key for (Tag, V)
- $E_2((K_1, K_2), M) := E(K_1, M), Tag(K_2, M)$
- $D_2((K_1, K_2), (C, T))$:
  - $M := D(K_1, C)$
  - if $V(K_2, M, T)$ then return M
  - else return 'ERROR'

The problem with this construction is that a MAC (Tag, V) can be secure even if $Tag()$ is deterministic. (E.g. CBC-MAC.) But if the construction (E_2, D_2) is instantiated with a deterministic $Tag()$, then it cannot even guarantee security for 2 encryptions (much less CPA-security or CCA security).

A more theoretical problem with this construction is that a MAC (Tag, V) can be secure even if $Tag(K, M)$ *completely gives away* M, and in such a case (E_2, D_2) is completely broken.

8.1.3 Authenticate-Then-Encrypt

Finally, consider the following scheme:

Construction (E_3, D_3)

- Key: a pair (K_1, K_2) where K_1 is a key for (E, D) and K_2 is a key for (Tag, V)
- $E_3((K_1, K_2), M)$:
  - $T := Tag(K_2, M)$
  - return $E(K_1, (M, T))$
- $D_3((K_1, K_2), C$:
  - $(M, T) := D(K_1, C)$
  - if $V(K_2, M, T)$ then return M

– else return 'ERROR'

The problem with this construction is rather subtle.

First of all, the major problem of the construction (E_2, D_2), in which we lost even security for two encryptions, does not occur.

Exercise 8 *Show that if (E, D) is (t, ϵ) CPA-secure and E, D, Tag, V all have running time at most r, then (E_3, D_3) is $(t/O(r), \epsilon)$ CPA secure*

It is possible, however, that (E, D) is CPA-secure and (Tag, V) is existentially unforgeable under chosen message attack, and yet (E_3, D_3) is not CCA-secure.

Suppose that (Tag, V) is such that, in $Tag(K, M)$, the first bit is ignored in the verification. This seems reasonable in the case that some padding is needed to fill out a network protocol, for example. Further, suppose (E, D) is counter mode with a pseudo-random function, F_{K_1}.

Take the encryption of $M_1, ..., M_l$ with keys K_1 and K_2 for E and V, respectively. Pick a random r. Then, by the definition of the encryption scheme in counter mode, the encryption of E, T will be:

$$r, F_{K_1}(r) \oplus M_1, F_{K_1}(r+1) \oplus M_2, ..., F_{K_1}(r+l-1) \oplus M_l, F_{K_1}(r+l) \oplus T_1, F_{K_1}(r+l+1) \oplus T_2, ...$$

Consider this CCA attack. Let $e_1 = (1, 0, ..., 0)$. The attacker sees

$$r, C_1, C_2, ..., C_l, C_{l+1}, ..., C_{l+c}$$

Take $e_1 \oplus C_{l+1}$ (or any of the other tags). Clearly, this is not the original message encryption, as a bit has been changed, so the Oracle will give you a decryption of it in a CCA attack. Since all that was modified in the ciphertext was the first bit, which was padding, the attacker has just used the Oracle to get the original message.

8.2 Cryptographic Hash Functions

8.2.1 Definition and Birthday Attack

Definition 31 (Collision-Resistant Hash Function) *A function $H : \{0,1\}^k \times \{0,1\}^L \to \{0,1\}^\ell$ is a (t, ϵ) secure collision resistant hash function if $L > \ell$ and for every algorithm A of complexity $\leq t$ we have*

$$\mathbb{P}[A(s) = (x, x') : H^s(x) = H^s(x')] \leq \epsilon \tag{8.1}$$

The idea is that, for every key (or *seed*) $s \in \{0,1\}^k$ we have a length-decreasing function $H^s : \{0,1\}^L \to \{0,1\}^\ell$. By the pigeon-hole principle, such functions cannot be injective. An

efficient algorithm, however, cannot find *collisions* (pairs of inputs that produce the same output) even given the seed s.

The main *security parameter* in a construction of collision-resistent hash functions (that is, the parameter that one needs to increase in order to hope to achieve larger t and smaller ϵ) is the output length ℓ.

It is easy to see that if H has running time r and output length ℓ then we can find collisions in time $O(r \cdot 2^\ell)$ by computing values of H in any order until we find a collision. (By the pigeon-hole principle, a collision will be found in $2^\ell + 1$ attempts or fewer.)

If, specifically, we attack H by trying a sequence of *randomly chosen* inputs until we find a collision, then we can show that with only $2^{\ell/2}$ attempts we already have a constant probability of finding a collision. (The presence of a collision can be tested by sorting the outputs. Overall, this takes time $O(2^{\ell/2} \cdot \ell + 2^{\ell/2} \cdot r) = O(r \cdot 2^{\ell/2})$.) The calculation can be generalized to the following:

Consider A given $H^s(\cdot)$. Define A as picking m random strings $x_1 ... x_m$ and generating their respective outputs from H^s. We want to check the probability $\mathbb{P}$[collision] that $H^s(x_1) ... H^s(x_m)$ contains a repeated value. However, it is easier to begin by checking the probability of whether the m choices *do not* contain a collision, $\mathbb{P}$[no collision]:

$$\mathbb{P}[\text{no collision}] = 1 \cdot \left(1 - \frac{1}{2^\ell}\right) \cdot \left(1 - \frac{2}{2^\ell}\right) \cdots \left(1 - \frac{(\mathrm{m}-1)}{2^\ell}\right)$$

$$\begin{aligned}
\mathbb{P}[\text{collision}] &= 1 - \mathbb{P}[\text{no collision}] \\
&\approx e^{-\mathbb{P}[\text{no collision}]} \\
&= e^{-\frac{1}{2^\ell}} \cdot e^{-\frac{2}{2^\ell}} \cdot \ldots \cdot e^{-\frac{(m-1)}{2^\ell}} \\
&= e^{-\frac{1+2+\ldots+(m-1)}{2^\ell}} \\
&\approx e^{-\frac{m^2}{2^{\ell+1}}}
\end{aligned}$$

Note that this is a constant if $m \approx 2^{\ell/2}$.

(This calculation is called the "birthday paradox," because it establishes that in a set of n elements uniformly distributed, if you pick elements $x_1 ... x_{\sqrt{n}}$, where x_i is uniformally distributed in the set and x_i is independent, then there is a constant probability that $\exists i, j : x_i = x_j$. Specifically, in the case of birthdays, if there are 23 people in a room, then there is a probability over $\frac{1}{2}$ that two of the people in the room have the same birthday.)

Exercise 9 *If $H : \{0,1\}^k \times \{0,1\}^L \to \{0,1\}^\ell$ is a (t, ϵ) secure collision resistant hash function computable in time r, then*

$$\frac{t^2}{\epsilon} \leq O(r \cdot 2^\ell) \tag{8.2}$$

This should be contrasted with the case of pseudorandom generators and pseudorandom functions, where the security parameter is the seed (or key) length k, and the only known

generic attack is the one described in a previous exercise which gives

$$\frac{t}{\epsilon} \leq O(m2^{\ell}) \tag{8.3}$$

This means that if one wants a (t, ϵ) secure pseudorandom generator or function where, say, $t = 2^{80}$ and $\epsilon = 2^{-40}$, then it is somewhat plausible that a key of 128 bits might suffice. The same level of security for a collision-resistant hash function, however, requires a key of at least about 200 bits.

To make matters worse, attacks which are significantly faster than the generic birthday attack have been found for the two constructions of hash functions which are most used in practice: MD5 and SHA-1. MD5, which has $\ell = 128$, is completely broken by such new attacks. Implementing the new attacks on SHA-1 (which has $\ell = 160$) is not yet feasible but is about 1,000 times faster than the birthday attack.

There is a process under way to define new standards for collision-resistant hash functions.

8.2.2 The Merkle-Damgård Transform

In practice, it is convenient to have collision-resistant hash functions $H : \{0,1\}^k \times \{0,1\}^* \rightarrow \{0,1\}^{\ell}$ in which the input is allowed to be (essentially) arbitrarily long.

The Merkle-Damgård transform is a generic way to transform a hash function that has $L = 2\ell$ into one that is able to handle messages of arbitrary length.

Let $H : \{0,1\}^k \times \{0,1\}^{2\ell} \rightarrow \{0,1\}^{\ell}$ a hash functions that compresses by a factor of two.

Then $H^s_{MD}(M)$ is defined as follows (IV is a fixed ℓ-bit string, for example the all-zero string):

- Let L be the length of M
- divide M into blocks $M_1, \ldots, M_B$ of length ℓ, where $B = \lceil L/\ell \rceil$
- $h_0 := IV$
- for $i := 1$ to B : $h_i := H^s(M_i, h_{i-1})$
- return $H^s(L, h_B)$

We can provide the following security analysis:

Theorem 32 *If H is (t, ϵ)-secure and has running time r, then H_{MD} has security $(t - O(rL/\ell), \epsilon)$ when used to hash messages of length up to L.*

Proof:

Suppose that A, given s, has a probability ϵ of finding messages M, M' where $M = M_1, ..., M_B, M' = M'_1, ..., M'_B$ and $H^s_{MD}(M) = H^s_{MD}(M')$, where $B = L/\ell, B' = L'/\ell$. Let us construct an algorithm A' which:

1. Given s, runs A to find collisions for M, M' in H^s_{MD}. Assume that running A takes time t_{MD}.

2. Computes both $H^s_{MD}(M) and H^s_{MD}(M')$. We know that this takes time on the order of $r \cdot 2L/\ell$, where r is the time of running H^s and assuming without loss of generality that $L \geq L'$ since the Merkle-Damgård Transform, by definition, runs H^s once for each of the L/ℓ blocks of M and M'.

3. Finds a collision of H^s that is guaranteed to exist (we will prove this below) in the Merkle-Damgård algorithm. This will take the time of running the algorithm once for each message M, M', which is on the order of $r \cdot 2L/\ell$, as above.

In order to show that there must be a collision in H^s if there is a collision in H^s_{MD}, recall that the last computation of the MD algorithm hashes $M_{B+1} = L$. We need to consider two cases:

1. $L \neq L'$: By the definition of the problem, we have $H^s_{MD}(M) = H^s_{MD}(M')$. The last step of the alrogithm for M and M' generates, respectively, $H^s(L, h_B) = H^s_{MD}(M)$ and $H^s(L', h'_{B'}) = H^s_{MD}(M') = H^s_{MD}(M)$. This is a collision of H^s on different inputs.

2. $L = L'$: This implies that $B = B'$ and $h_{B+1} = h'_{B'+1}$, as well. Because $M \neq M'$ but $|M| \neq |M'|$, $\exists i, 0 \leq i \leq B : h_i \neq h'_i$.

   Let $j \leq B+1$ be the largest index where $h_j \neq h'_j$. If $j = B+1$, then h_{j+1} and h'_{j+1} are two (different) colliding strings because $H^s(h_j) = h_{j+1} = H^s_{MD}(M) = H^s_{MD}(M') = h'_{j+1} = H^s(h'_j)$.

   If $j \leq B$, then that implies $h_{j+1} = h'_{j+1}$. Thus, h_j and h'_j are two different strings whose hashes collide.

This shows that a collision in H^s_{MD} implies a collision in H^s, thus guaranteeing for our proof that by executing each stage of the Merkle-Damgård Transform, we can find a collision in H^s if there is a collision in H^s_{MD}.

Returning to algorithm A', we now have established the steps necessary to find collisions of a hash function H^s by finding collisions in H^s_{MD}. Note that by the definition of the problem, because it uses algorithm A which finds collisions with probability ϵ, A' will also find collisions with probability ϵ. By the definition of a collision-resistant hash function, we also know that it must have time complexity t.

Examining the steps of the algorithm, we know that the time complexity of A is $t = t_{MD} + 2r \cdot L/\ell + 2r \cdot L/\ell = 4r \cdot L/\ell = t_{MD} + O(rL/\ell)$. From this, we solve for t_{MD} and get $t_{MD} = t - O(rL/\ell)$. Thus, we conclude that if A finds a collision with probability ϵ, it is $(t - O(RL/\ell), \epsilon)$ secure.

□

8.3 Hash Functions and Authentication

Suppose $H : \{0,1\}^k \times \{0,1\}^{2\ell} \to \{0,1\}^\ell$ is a collision-resistant hash function and $H_{MD} : \{0,1\}^k \times \{0,1\}^* \to \{0,1\}^\ell$ is the hash function derived from the Merkle-Damgård transform.

Two popular schemes for message authentication involve using a key $K \in \{0,1\}^\ell$, and authenticating a message M as either

1. $H^s_{MD}(K, M)$ or
2. $H^s_{MD}(M, K)$.

The first scheme has a serious problem: given the tag of a message $M_1, \ldots, M_B$ of length ℓ, we can compute the tag of any message that starts with $M_1, \ldots, M_B, \ell$.

The NMAC scheme works roughly as a scheme of the second type, but with two keys, the first used in place of IV in the Merkle-Damgård transform, the second put at the end of the message.

This is secure assuming that $H^s(K, M)$ (where H is the fixed-length hash function) is a secure MAC for fixed length messages.

An alternative scheme (which is easier to implement given current cryptographic libraries) is HMAC, which uses $H^s(IV, K \oplus ipad)$ as a first key and $H^s(IV, K \oplus opad)$ as a second key, where *ipad* and *opad* are two fixed strings. This is secure if, in addition to the assumptions that make NMAC secure, we have that the mapping

$$s, K \to s, H^s(IV, K \oplus ipad), H^s(IV, K \oplus opad)$$

is a pseudorandom generator.

Lecture 9

One-Way Functions and Hardcore Predicates

Summary

Today we begin a tour of the theory of one-way functions and pseudorandomness.

The highlight of the theory is a proof that if *one-way functions* exist (with good asymptotic security) then pseudorandom permutations exist (with good asymptotic security). We have seen that pseudorandom permutations suffice to do encryption and authentication with extravagantly high levels of security (respectively, CCA security and existential unforgeability under chosen message attack), and it is easy to see that if one-way functions do not exist, then every encryption and authentication scheme suffers from a total break.

Thus the conclusion is a strong "dichotomy" result, saying that either cryptography is fundamentally impossible, or extravagantly high security is possible.

Unfortunately the proof of this result involves a rather inefficient reduction, so the concrete parameters for which the dichotomy holds are rather unrealistic. (One would probably end up with a system requiring gigabyte-long keys and days of processing time for each encryption, with the guarantee that if it is not CCA secure then every 128-bit key scheme suffers a total break.) Nonetheless it is one of the great unifying achievements of the asymptotic theory, and it remains possible that a more effective proof will be found.

In this lecture and the next few ones we shall prove the weaker statement that if *one-way permutations* exist then pseudorandom permutations exist. This will be done in a series of four steps each involving reasonable concrete bounds. A number of combinatorial and number-theoretic problems which are believed to be intractable give us highly plausible candidate one-way permutations. Overall, we can show that if any of those well-defined and well-understood problems are hard, then we can get secure encryption and authentication with schemes that are slow but not entirely impractical. If, for example, solving discrete log with a modulus of the order of $2^{1,000}$ is hard, then there is a CCA-secure encryption scheme requiring a $4,000$-bit key and fast enough to carry email, instant messages and probably

voice communication. (Though probably too slow to encrypt disk access or video playback.)

9.1 One-way Functions and One-way Permutations

A one-way function f is a function such that, for a random x, it is hard to find a pre-image of $f(x)$.

Definition 33 (One-way Function) *A function $f : \{0,1\}^n \to \{0,1\}^m$ is (t,ϵ)-one way if for every algorithm A of complexity $\leq t$ we have*

$$\mathop{\mathbb{P}}_{x\sim\{0,1\}^n}[A(f(x)) = x' : f(x) = f(x')] \leq \epsilon$$

In the asymptotic theory, one is interested in one-way functions that are defined for all input lengths and are efficiently computable. Recall that a function $\nu : \mathbb{N} \to \mathbb{R}$ is called *negligible* if for every polynomial p we have $\lim_{n\to\infty} \nu(n)/p(n) = 0$.

Definition 34 (One-way Function – Asymptotic Definition) *A function $f : \{0,1\}^* \to \{0,1\}^*$ is one-way if*

1. *f is polynomial time computable and*
2. *for every polynomial $p()$ there is a negligible function ν such that for all large enough n the function $f_n(x) := (n, f(x))$ is $(t(n), \nu(n))$-one way.*

Example 35 (Subset Sum) *On input $x \in \{0,1\}^n$, where $n = k \cdot (k+1)$, $SS_k(x)$ parses x as a sequence of k integers, each k-bit long, plus a subset $I \subseteq \{1,\ldots,k\}$.*

The output is

$$SS_k(x_1,\ldots,x_k,I) := x_1,\ldots,x_k,\sum_{i\in I} x_i$$

Some variants of subset-sum have been broken, but it is plausible that SS_k is a (t,ϵ)-one way function with t and $1/\epsilon$ super-polynomial in k, maybe even as large as $2^{k^{\Omega(1)}}$.

Exercise 10 *Let $f : \{0,1\}^n \to \{0,1\}^m$ be a (t,ϵ)-secure one-way function. Show that*

$$\frac{t}{\epsilon} \leq O((m+n)\cdot 2^n)$$

Definition 36 (One-way Permutation) *If $f : \{0,1\}^n \to \{0,1\}^n$ is a bijective (t,ϵ)-one way function, then we call f a (t,ϵ)-one-way permutation.*

If f is an (asymptotic) one-way function, and for every n f is a bijection from $\{0,1\}^n$ into $\{0,1\}^n$, then we say that f is an (asymptotic) one-way permutation.

There is a non-trivial general attack against one-way permutations.

Exercise 11 *Let $f : \{0,1\}^n \to \{0,1\}^m$ be a (t,ϵ)-secure one-way permutation. Show that*

$$\frac{t^2}{\epsilon} \leq O((m+n)^2 \cdot 2^n)$$

This means that we should generally expect the input length of a secure one-way permutation to be at least 200 bits or so. (Stronger attacks than the generic one are known for the candidates that we shall consider, and their input length is usually 1000 bits or more.)

Example 37 (Modular Exponentiation) *Let p be a prime, and $\mathbb{Z}_p^*$ be the group whose elements are $\{1,\ldots,p-1\}$ and whose operation is multiplication* mod$p$*. It is a fact (which we shall not prove) that $\mathbb{Z}_p^*$ is cyclic, meaning that there is an element g such that the mapping*

$$EXP_{g,p}(x) := g^x \bmod p$$

is a permutation on $\mathbb{Z}_p^*$. Such an element g is called a generator, and in fact most elements of $\mathbb{Z}_p^*$ are generators. $EXP_{g,p}$ is conjectured to be one-way for most choices of p and g.

The problem of inverting $EXP_{g,p}$ is called the discrete logarithm problem.

The best known algorithm for the discrete logarithm is conjectured to run in time $2^{O((logp)^{1/3})}$. It is plausible that for most p and most g the discrete logarithm is a (t,ϵ) one way permutation with t and ϵ^{-1} of the order of $2^{(\log p)^{\Omega(1)}}$.

Problems like exponentiation do not fit well in the asymptotic definition, because of the extra parameters g, p. (Technically, they do not fit our definitions at all because the input is an element of $\mathbb{Z}_p^*$ instead of a bit string, but this is a fairly trivial issue of data representation.) This leads to the definition of *family of one-way functions (and permutations).*

9.2 A Preview of What is Ahead

Our proof that a pseudorandom permutation can be constructed from any one-way permutation will proceed via the following steps:

1. We shall prove that for any one-way permutation f we can construct a *hard-core predicate* P, that is a predicate P such that $P(x)$ is easy to compute given x, but it is hard to compute given $f(x)$.
2. From a one-way function with a hard-core predicate, we shall show how to construct a pseudorandom generator with *one-bit expansion*, mapping ℓ bits into $\ell + 1$.
3. From a pseudorandom generator with one-bit expansion, we shall show how to get generators with essentially *arbitrary expansion.*

4. From a length-doubling generator mapping ℓ bits into 2ℓ, we shall show how to get *pseudorandom functions*.

5. For a pseudorandom function, we shall show how to get *pseudorandom permutations*.

9.3 Hard-Core Predicate

Definition 38 (Hard-Core Predicate) *A boolean function $P : \{0,1\}^n \to \{0,1\}$ is (t, ϵ)-hard core for a permutation $f : \{0,1\}^n \to \{0,1\}^n$ if for every algorithm A of complexity $\leq t$*

$$\mathop{\mathbb{P}}_{x \sim \{0,1\}^n}[A(f(x)) = P(x)] \leq \frac{1}{2} + \epsilon$$

Note that only one-way permutations can have efficiently computable hard-core predicates.

Exercise 12 *Suppose that P is a (t, ϵ)-hard core predicate for a permutation $f : \{0,1\}^n \to \{0,1\}^n$, and P is computable in time r. Show that f is $(t - r, 2\epsilon)$-one way.*

It is known that if $Exp_{g,p}$ is one-way, then every bit of x is hard-core.

Our main theorem will be that a random XOR is hard-core for every one-way permutation.

9.4 The Goldreich-Levin Theorem

We will use the following notation for "inner product" modulo 2:

$$\langle x, r \rangle := \sum_i x_i r_i \bmod 2 \tag{9.1}$$

Theorem 39 (Goldreich and Levin) *Suppose that A is an algorithm of complexity t such that*

$$\mathop{\mathbb{P}}_{x,r}[A(f(x), r) = \langle x, r \rangle] \geq \frac{1}{2} + \epsilon \tag{9.2}$$

Then there is an algorithm A' of complexity at most $O(t\epsilon^{-4} n^{O(1)})$ such that

$$\mathop{\mathbb{P}}_{x}[A'(f(x)) = x] \geq \Omega(\epsilon)$$

We begin by establishing the following weaker result.

Theorem 40 (Goldreich and Levin – Weak Version) *Suppose that A is an algorithm of complexity t such that*

$$\mathop{\mathbb{P}}_{x,r}[A(f(x),r) = \langle x, r\rangle] \geq \frac{15}{16} \tag{9.3}$$

Then there is an algorithm A' of complexity at most $O(tn\log n + n^2\log n)$ such that

$$\mathop{\mathbb{P}}_{x}[A'(f(x)) = x] \geq \frac{1}{3}$$

Before getting into the proof of Theorem 40, it is useful to think of the "super-weak" version of the Goldreich-Levin theorem, in which the right-hand-side in (9.3) is 1. Then inverting f is very easy. Call $e_i \in \{0,1\}^n$ the vector that has 1 in the i-th position and zeroes everywhere else, thus $\langle x, e_i\rangle = x_i$. Now, given $y = f(x)$ and an algorithm A for which the right-hand-side of (9.3) is 1, we have $x_i = A(y, e_i)$ for every i, and so we can compute x given $f(x)$ via n invocations of A. In order to prove the Goldreich-Levin theorem we will do something similar, but we will have to deal with the fact that we only have an algorithm that approximately computes inner products.

We derive the Weak Goldreich-Levin Theorem from the following reconstruction algorithm.

Lemma 41 (Goldreich-Levin Algorithm – Weak Version) *There is an algorithm GLW that given oracle access to a function $H : \{0,1\}^n \to \{0,1\}^n$ such that, for some $x \in \{0,1\}^n$,*

$$\mathop{\mathbb{P}}_{r\sim\{0,1\}^n}[H(r) = \langle x, r\rangle] \geq \frac{7}{8}$$

runs in time $O(n^2\log n)$, makes $O(\log n)$ queries into H, and with $1 - o(1)$ probability outputs x.

Before proving Lemma 41, we need to state the following version of the *Chernoff Bound*.

Lemma 42 (Chernoff Bound) *Let $X_1, \ldots, X_n$ be mutually independent $0/1$ random variables. Then, for every $\epsilon > 0$, we have*

$$\mathbb{P}\left[\sum_{i=1}^n X_i > \mathbb{E}\sum_{i=1}^n X_i + \epsilon n\right] \leq e^{-2\epsilon^2 n} \tag{9.4}$$

Proof: We only give a sketch. Let $Y_i := X_i - \mathbb{E}\, X_i$. Then we want to prove that

$$\mathbb{P}\left[\sum_i Y_i > \epsilon n\right] \leq e^{-2\epsilon^2 n}$$

For every fixed λ, Markov's inequality gives us

$$\begin{aligned}\mathbb{P}\left[\sum_i Y_i > \epsilon n\right] &= \mathbb{P}\left[e^{\lambda \sum_i Y_i} > e^{\lambda \epsilon n}\right] \\ &\leq \frac{\mathbb{E}\, e^{\lambda \sum_i Y_i}}{e^{\lambda \epsilon n}}\end{aligned}$$

We can use independence to write

$$\mathbb{E}\, e^{\lambda \sum_i Y_i} = \prod_i \mathbb{E}\, e^{\lambda Y_i}$$

and some calculus shows that for every Y_i we have

$$\mathbb{E}\, e^{\lambda Y_i} \leq e^{\lambda^2/8}$$

So we get

$$\mathbb{P}\left[\sum_i Y_i > \epsilon n\right] \leq e^{\lambda^2 n/8 - \lambda \epsilon n} \tag{9.5}$$

Equation (9.5) holds for every $\lambda > 0$, and in particular for $\lambda := 4\epsilon$ giving us

$$\mathbb{P}\left[\sum_i Y_i > \epsilon n\right] \leq e^{-2\epsilon^2 n}$$

as desired. □

We can proceed with the design and the analysis of the algorithm of Lemma 41.

PROOF:[Of Lemma 41] The idea of the algorithm is that we would like to compute $\langle x, e_i \rangle$ for $i = 1, \ldots, n$, but we cannot do so by simply evaluating $H(e_i)$, because it is entirely possible that H is incorrect on those inputs. If, however, we were just interested in computing $\langle x, r \rangle$ for a random r, then we would be in good shape, because $H(r)$ would be correct with resonably large probability. We thus want to *reduce the task of computing $\langle x, y \rangle$ on a specific y, to the task of computing $\langle x, r \rangle$ for a random r.* We can do so by observing the following identity: for every y and every r, we have

$$\langle x, y \rangle = \langle x, r + y \rangle - \langle x, r \rangle$$

where all operations are mod 2. (And bit-wise, when involving vectors.) So, in order to compute $\langle x, y \rangle$ we can pick a random r, and then compute $H(r+y) - H(r)$. If r is uniformly distributed, then $H(r + y)$ and $H(r)$ are uniformly distributed, and we have

$$\begin{aligned}
& \mathop{\mathbb{P}}_{r\sim\{0,1\}^n}[H(r+y) - H(r) = \langle x, y\rangle] \\
\geq & \mathop{\mathbb{P}}_{r\sim\{0,1\}^n}[H(r+y) = \langle x, r+y\rangle \wedge H(r) = \langle x, r\rangle] \\
= & \; 1 - \mathop{\mathbb{P}}_{r\sim\{0,1\}^n}[H(r+y) \neq \langle x, r+y\rangle \vee H(r) \neq \langle x, r\rangle] \\
\geq & \; 1 - \mathop{\mathbb{P}}_{r\sim\{0,1\}^n}[H(r+y) \neq \langle x, r+y\rangle] - \mathop{\mathbb{P}}_{r\sim\{0,1\}^n}[H(r) \neq \langle x, r\rangle] \\
\geq & \; \frac{3}{4}
\end{aligned}$$

Suppose now that we pick independently several random vectors $r_1, \ldots, r_k$, and that we compute $Y_j := H(r_j + y) - H(r_j)$ for $j = 1, \ldots, k$ and we take the majority value of the Y_j as our estimate for $\langle x, y\rangle$. By the above analysis, each Y_j equals $\langle x, y\rangle$ with probability at least $3/4$; furthermore, the events $Y_j = \langle x, y\rangle$ are mutually independent. We can then invoke the Chernoff bound to deduce that the probability that the majority value is wrong is at most $e^{-k/8}$. (If the majority vote of the Y_j is wrong, it means that at least $k/2$ or the Y_j are wrong, even though the expected number of wrong ones is at most $k/4$, implying a deviation of $k/4$ from the expectation; we can invoke the Chernoff bound with $\epsilon = 1/4$.) The algorithm GLW is thus as follows:

- Algorithm GLW
- for $i := 1$ to n
  - for $j := 1$ to $16 \log n$
    - pick a random $r_j \in \{0,1\}^n$
  - $x_i :=$ majority$\{H(r_j + e_i) - H(e_i) : j = 1, \ldots, 16 \log n\}$
- return x

For every i, the probability fails to compute $\langle x, e_i\rangle = x_i$ is at most $e^{-2\log n} = 1/n^2$. So the probability that the algorithm fails to return x is at most $1/n = o(1)$. The algorithm takes time $O(n^2 \log n)$ and makes $32n \log n$ oracle queries into H. □

In order to derive Theorem 40 from Lemma 41 we will need the following variant of Markov's inequality.

Lemma 43 *Let X be a discrete bounded non-negative random variable ranging over $[0, 1]$. Then for every $0 \leq t \leq \mathbb{E}\, X$,*

$$\mathbb{P}[X \geq t] \geq \frac{\mathbb{E}\, X - t}{1 - t} \tag{9.6}$$

PROOF: Let R be the set of values taken by X with non-zero probability. Then

$$\begin{aligned}
\mathbb{E}\, X &= \sum_{v \in R} v \cdot \mathbb{P}[X = v] \\
&= \sum_{v \in R: v < t} v \cdot \mathbb{P}[X = v] + \sum_{v \in R: v \geq t} v \cdot \mathbb{P}[X = v] \\
&\leq \sum_{v \in R: v < t} t \cdot \mathbb{P}[X = v] + \sum_{v \in R: v \geq t} 1 \cdot \mathbb{P}[X = v] \\
&= t \cdot \mathbb{P}[X < t] + \mathbb{P}[X \geq t] \\
&= t - t \cdot \mathbb{P}[X \geq t] + \mathbb{P}[X \geq t]
\end{aligned}$$

So we have $\mathbb{P}[X \geq t] \cdot (1 - t) \geq \mathbb{E}\, X - t$. □

We can now prove Theorem 40.

PROOF:[Of Theorem 40] The assumption of the Theorem can be rewritten as

$$\mathop{\mathbb{E}}_{x}\left[\mathop{\mathbb{P}}_{r}[A(f(x), r) = \langle x, r\rangle]\right] \geq \frac{15}{16}$$

From Lemma 43 we have

$$\mathop{\mathbb{P}}_{x}\left[\mathop{\mathbb{P}}_{r}[A(f(x), r) = \langle x, r\rangle] \geq \frac{7}{8}\right] \geq \frac{1}{2}$$

Call an x "good" if it satisfies $\mathbb{P}_r[A(f(x), r) = \langle x, r\rangle]$.

The inverter A', on input $y = f(x)$, runs the algorithm GLW using the oracle $A'(y, \cdot)$. If x is good, then the algorithm finds x with probability at least $1 - o(1)$. At least half of the choices of x are good, so overall the algorithm inverts $f()$ on at least a $.5 - o(1) > 1/3$ fraction of inputs. The running time of the algorithm if $O(n^2 \log n)$ plus the cost of $32n \log n$ calls to $A(y, \cdot)$, each costing time t. □

We need, however, the full Goldreich-Levin Theorem in order to construct a pseudorandom generator, and so it seems that we have to prove a strengthening of Lemma **??** in which the right-hand-side in (9.7) is $\frac{1}{2} + \epsilon$.

Unfortunately such a stronger version of Lemma **??** is just false: for any two different $x, x' \in \{0,1\}^n$ we can construct an H such that

$$\mathop{\mathbb{P}}_{r \sim \{0,1\}^n}[H(r) = \langle x, r\rangle] = \frac{3}{4}$$

and

$$\mathop{\mathbb{P}}_{r \sim \{0,1\}^n}[H(r) = \langle x', r\rangle] = \frac{3}{4}$$

so no algorithm can be guaranteed to find x given an arbitrary function H such that $\mathbb{P}[H(r) = \langle x, r\rangle] = \frac{3}{4}$, because x need not be uniquely defined by H.

We can, however, prove the following:

Lemma 44 (Goldreich-Levin Algorithm) *Suppose we have access to a function $H : \{0,1\}^n \to \{0,1\}$ such that, for some unknown x, we have*

$$\mathop{\mathbb{P}}_{r\in\{0,1\}^n}[H(r) = \langle x, r\rangle] \geq \frac{1}{2} + \epsilon \tag{9.7}$$

where $x \in \{0,1\}^n$ is an unknown string, and $\epsilon > 0$ is given.

Then there is an algorithm GL that runs in time $O(n^2\epsilon^{-4}\log n)$, makes $O(n\epsilon^{-4}\log n)$ oracle queries into H, and outputs a set $L \subseteq \{0,1\}^n$ such that $|L| = O(\epsilon^{-2})$ and with probability at least $1/2$, $x \in L$.

The Goldreich-Levin algorithm GL has other interpretations (an algorithm that learns the Fourier coefficients of H, an algorithm that decodes the Hadamard code is sub-linear time) and various applications outside cryptography.

The Goldreich-Levin Theorem is an easy consequence of Lemma 44. Let A' take input y and then run the algorithm of Lemma 44 with $H(r) = A(y, r)$, yielding a list L. A' then checks if $f(x) = y$ for any $x \in L$, and outputs it if one is found.

From the assumption that

$$\mathop{\mathbb{P}}_{x,r}[A(f(x), r) = \langle x, r\rangle] \geq \frac{1}{2} + \epsilon$$

it follows by Markov's inequality (See Lemma 9 in the last lecture) that

$$\mathop{\mathbb{P}}_{x}\left[\mathop{\mathbb{P}}_{r}[A(f(x), r) = \langle x, r\rangle] \geq \frac{1}{2} + \frac{\epsilon}{2}\right] \geq \frac{\epsilon}{2}$$

Let us call an x such that $\mathbb{P}_r[A(f(x), r) = \langle x, r\rangle] \geq \frac{1}{2} + \frac{\epsilon}{2}$ a *good* x. If we pick x at random and give $f(x)$ to the above algorithm, there is a probability at least $\epsilon/2$ that x is good and, if so, there is a probability at least $1/2$ that x is in the list. Therefore, there is a probability at least $\epsilon/4$ that the algorithm inverts $f()$, where the probability is over the choices of x and over the internal randomness of the algorithm.

9.5 The Goldreich-Levin Algorithm

In this section we prove Lemma 44.

We are given an oracle $H()$ such that $H(r) = \langle x, r\rangle$ for an $1/2 + \epsilon$ fraction of the r. Our goal will be to use $H()$ to simulate an oracle that has agreement $7/8$ with $\langle x, r\rangle$, so that

we can use the algorithm of Lemma **??** the previous section to find x. We perform this "reduction" by "guessing" the value of $\langle x, r\rangle$ at a few points.

We first choose k random points $r_1 \dots r_k \in \{0,1\}^n$ where $k = O(1/\epsilon^2)$. For the moment, let us suppose that we have "magically" obtained the values $\langle x, r_1\rangle, \dots, \langle x, r_k\rangle$. Then define $H'(r)$ as the majority value of:

$$H(r + r_j) - \langle x, r_j\rangle \quad j = 1, 2, \dots, k \tag{9.8}$$

For each j, the above expression equals $\langle x, r\rangle$ with probability at least $\frac{1}{2} + \epsilon$ (over the choices of r_j) and by choosing $k = O(1/\epsilon^2)$ we can ensure that

$$\mathop{\mathbb{P}}_{r, r_1, \dots, r_k} \left[H'(r) = \langle x, r\rangle\right] \geq \frac{31}{32}. \tag{9.9}$$

from which it follows that

$$\mathop{\mathbb{P}}_{r_1, \dots, r_k} \left[\mathop{\mathbb{P}}_{r} \left[H'(r) = \langle x, r\rangle\right] \geq \frac{7}{8}\right] \geq \frac{3}{4}. \tag{9.10}$$

Consider the following algorithm.

function GL-First-Attempt
  pick $r_1, \dots, r_k \in \{0,1\}^n$ where $k = O(1/\epsilon^2)$
  for all $b_1, \dots, b_k \in \{0,1\}$ **do**
    define $H'_{b_1 \dots b_k}(r)$ as majority of: $H(r + r_j) - b_j$
    apply Algorithm GLW to $H'_{b_1 \dots b_t}$
    add result to list
  end for
  return list
end function

The idea behind this program is that we do not in fact know the values $\langle x, r_j\rangle$, but we can "guess" them by considering all choices for the bits b_j. If $H(r)$ agrees with $\langle x, r\rangle$ for at least a $1/2 + \epsilon$ fraction of the rs, then there is a probability at least $3/4$ that in one of the iteration we invoke algorithm GLW with a simulated oracle that has agreement $7/8$ with $\langle x, r\rangle$. Therefore, the final list contains x with probability at least $3/4 - 1/n > 1/2$.

The obvious problem with this algorithm is that its running time is exponential in $k = O(1/\epsilon^2)$ and the resulting list may also be exponentially larger than the $O(1/\epsilon^2)$ bound promised by the Lemma.

To overcome these problems, consider the following similar algorithm.

function GL
  pick $r_1, \dots, r_t \in \{0,1\}^n$ where $t = \log O(1/\epsilon^2)$

  define $r_S := \sum_{j \in S} r_j$ for each non-empty $S \subseteq \{1, \dots, t\}$

  for all $b_1, \dots, b_t \in \{0,1\}$ **do**
    define $b_S := \sum_{j \in S} b_j$ for each non-empty $S \subseteq \{1, \dots, t\}$

define $H'_{b_1 \ldots b_t}(r)$ as majority over non-empty $S \subseteq \{1, \ldots, t\}$ of $H(r + r_S) - b_S$

run Algorithm GLW with oracle $H'_{b_1 \ldots b_t}$

add result to list

end for
return list
end function

Let us now see why this algorithm works. First we define, for any nonempty $S \subseteq \{1, \ldots, t\}$, $r_S = \sum_{j \in S} r_j$. Then, since $r_1, \ldots, r_t \in \{0,1\}^n$ are random, it follows that for any $S \neq T$, r_S and r_T are independent and uniformly distributed. Now consider an x such that $\langle x, r \rangle$ and $H(r)$ agree on a $\frac{1}{2} + \epsilon$ fraction of the values of r. Then for the choice of $\{b_j\}$ where $b_j = \langle x, r_j \rangle$ for all j, we have that

$$b_S = \langle x, r_S \rangle$$

for every non-empty S. In such a case, for every S and every r, there is a probability at least $\frac{1}{2} + \epsilon$, over the choices of the r_j that

$$H(r + r_S) - b_S = \langle x, r \rangle \ ,$$

and these events are pair-wise independent. Note the following simple lemma.

Lemma 45 *Let $R_1, \ldots, R_k$ be a set of pairwise independent $0-1$ random variables, each of which is 1 with probability at least $\frac{1}{2} + \epsilon$. Then $\mathbb{P}[\sum_i R_i \geq k/2] \geq 1 - \frac{1}{4\epsilon^2 k}$.*

PROOF: Let $R = R_1 + \cdots + R_t$. The variance of a 0/1 random variable is at most 1/4, and, because of pairwise independence, $\mathbf{Var}[R] = \mathbf{Var}[R_1 + \ldots + R_k] = \sum_i \mathbf{Var}[R_k] \leq k/4$.

We then have

$$\mathbb{P}[R \leq k/2] \leq \mathbb{P}[|R - \mathbb{E}[R]| \geq \epsilon k] \leq \frac{\mathbf{Var}[R]}{\epsilon^2 k^2} \leq \frac{1}{4\epsilon^2 k}$$

□

Lemma 45 allows us to upper-bound the probability that the majority operation used to compute H' gives the wrong answer. Combining this with our earlier observation that the $\{r_S\}$ are pairwise independent, we see that choosing $t = \log(128/\epsilon^2)$ suffices to ensure that $H'_{b_1 \ldots b_t}(r)$ and $\langle x, r \rangle$ have agreement at least 7/8 with probability at least 3/4. Thus we can use Algorithm $A_{\frac{7}{8}}$ to obtain x with high probability. Choosing t as above ensures that the list generated is of length at most $2^t = 128/\epsilon^2$ and the running time is then $O(n^2 \epsilon^{-4} \log n)$ with $O(n\epsilon^{-4} \log n)$ oracle accesses, due to the $O(1/\epsilon^2)$ iterations of Algorithm GLW, that makes $O(n \log n)$ oracle accesses, and to the fact that one evaluation of $H'()$ requires $O(1/\epsilon^2)$ evaluations of $H()$.

9.6 References

The results of this lecture are from [GL89]. Goldreich and Levin initially presented a different proof. They credit the proof with pairwise independence to Rackoff. Algorithm $A_{\frac{7}{8}}$ is due to Blum, Luby and Rubinfeld [BLR93]. The use of Algorithm GL-First-Attempt as a motivating example might be new to these notes. (Or, actually, to the Fall 2001 notes for my CS278 class.)

Lecture 10

Pseudorandom Generators from One-Way Permutations

Summary

Today we complete the proof that it is possible to construct a pseudorandom generator from a one-way permutation.

10.1 Pseudorandom Generators from One-Way Permutations

Last time we proved the Goldreich-Levin theorem.

Theorem 46 (Goldreich and Levin) *Let $f : \{0,1\}^n \to \{0,1\}^n$ be a (t,ϵ)-one way permutation computable in time $r \leq t$. Then the predicate $x, r \mapsto \langle x, r\rangle$ is $(\Omega(t \cdot \epsilon^2 \cdot n^{-O(1)}, 3\epsilon)$ hard core for the permutation $x, r \mapsto f(x), r$.*

A way to look at this result is the following: suppose f is $(2^{\Omega(n)}, 2^{-\Omega(n)})$ one way and computable in $n^{O(1)}$ time. Then $\langle x, r\rangle$ is a $(2^{\Omega(n)}, 2^{-\Omega(n)})$ hard-core predicate for the permutation $x, r \to f(x), r$.

From now on, we shall assume that we have a one-way permutation $f : \{0,1\}^n \to \{0,1\}^n$ and a predicate $P : \{0,1\}^n \to \{0,1\}$ that is (t,ϵ) hard core for f.

This already gives us a pseudorandom generator with one-bit expansion.

Theorem 47 (Yao) *Let $f : \{0,1\}^n \to \{0,1\}^n$ be a permutation, and suppose $P : \{0,1\}^n \to \{0,1\}$ is (t,ϵ)-hard core for f. Then the mapping*

$$x \mapsto P(x), f(x)$$

is $(t - O(1), \epsilon)$-pseudorandom generator mapping n bits into $n+1$ bits.

Note that f is required to be a permutation rather than just a function. If f is merely a function, it may always begin with 0 and the overall mapping would not be pseudorandom.

For the special case where the predicate P is given by Goldreich-Levin, the mapping would be

$$x \mapsto \langle x, r\rangle, f(x), r$$

PROOF: Suppose the mapping is not $(t-2,\epsilon)$-pseudorandom. There is an algorithm D of complexity $\leq t-2$ such that

$$\left| \mathop{\mathbb{P}}_{x\sim\{0,1\}^n}[D(P(x)f(x))=1] - \mathop{\mathbb{P}}_{\substack{b\sim\{0,1\}\\ x\sim\{0,1\}^n}}[D(bf(x))=1] \right| > \epsilon \tag{10.1}$$

where we have used the fact that since f is permutation, $f(x)$ would be a uniformly random element in $\{0,1\}^n$ when x is such.

We will first remove the absolute sign in (10.1). The new inequality holds for either D or $1-D$ (i.e. the complement of D), and they both have complexity at most $t-1$.

Now define an algorithm A as follows.

On input $y=f(x)$, pick a random bit $r\sim\{0,1\}$. If $D(r,y)=1$, then output r, otherwise output $1-r$.

Algorithm A has complexity at most t. We claim that

$$\mathop{\mathbb{P}}_{x\sim\{0,1\}^n}[A(f(x))=P(x)] > \frac{1}{2}+\epsilon$$

so $P(\cdot)$ is not (t,ϵ)-hard core.

To make explicit the dependence of A on r, we will denote by $A_r(f(x))$ the fact that A picks r as its random bit.

To prove the claim, we expand

$$\begin{aligned}\mathop{\mathbb{P}}_{x,r}[A_r(f(x))=P(x)] &= \mathop{\mathbb{P}}_{x,r}[A_r(f(x))=P(x)\mid r=P(x)]\,\mathbb{P}[r=P(x)] + \\ &\quad \mathop{\mathbb{P}}_{x,r}[A_r(f(x))=P(x)\mid r\neq P(x)]\,\mathbb{P}[r\neq P(x)]\end{aligned}$$

Note that $\mathbb{P}[r=P(x)]=\mathbb{P}[r\neq P(x)]=1/2$ no matter what $P(x)$ is. The above probability thus becomes

$$\frac{1}{2}\mathop{\mathbb{P}}_{x,r}[D(rf(x))=1\mid r=P(x)] + \frac{1}{2}\mathop{\mathbb{P}}_{x,r}[D(rf(x))=0\mid r\neq P(x)] \tag{10.2}$$

The second term is just $\frac{1}{2} - \frac{1}{2}\mathbb{P}_{x,r}[D(rf(x)) = 1 \mid r \neq P(x)]$. Now we add to and subtract from (10.2) the quantity $\frac{1}{2}\mathbb{P}_{x,r}[D(rf(x)) = 1 \mid r = P(x)]$, getting

$$\frac{1}{2} + \mathop{\mathbb{P}}_{x,r}[D(rf(x)) = 1 \mid r = P(x)] -$$
$$\left(\frac{1}{2}\mathbb{P}[D(rf(x)) = 1 \mid r = P(x)] +\right.$$
$$\left.\frac{1}{2}\mathbb{P}[D(rf(x)) = 1 \mid r \neq P(x)]\right)$$

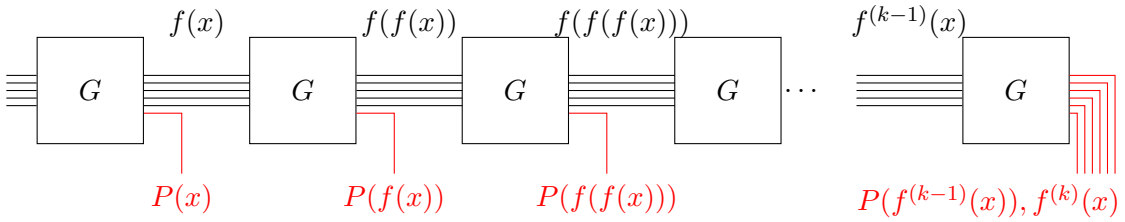
The expression in the bracket is $\mathbb{P}[D(rf(x)) = 1]$, and by our assumption on D, the whole expression is more than $\frac{1}{2} + \epsilon$, as claimed. □

The main idea of the proof is to convert something that distinguishes (i.e. D) to something that outputs (i.e. A). D helps us distinguish good answers and bad answers.

We will amplify the expansion of the generator by the following idea: from an n-bit input, we run the generator to obtain $n+1$ pseudorandom bits. We output one of those $n+1$ bits and feed the other n back into the generator, and so on. Specialized to the above construction, and repeated k times the mapping becomes

$$G_k(x) := P(x), P(f(x)), P(f(f(x)), \ldots, P(f^{(k-1)}(x), f^{(k)}(x) \tag{10.3}$$

This corresponds to the following diagram where all output bits lie at the bottom.



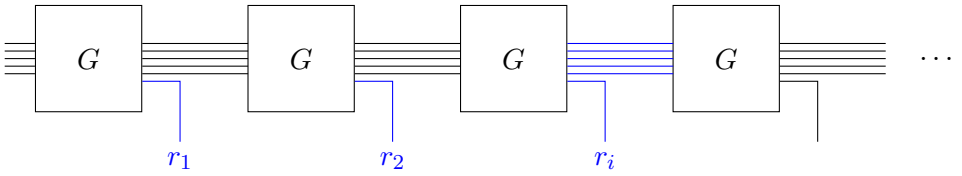
Theorem 48 (Blum-Micali) *Let $f : \{0,1\}^n \to \{0,1\}^n$ be a permutation, and suppose $P : \{0,1\}^n \to \{0,1\}$ is (t,ϵ)-hard core for f and that f, P are computable with complexity r.*

Then $G_k : \{0,1\}^n \to \{0,1\}^{n+k}$ as defined in (10.3) is $(t - O(rk), \epsilon k)$-pseudorandom.

Proof: Suppose G_k is not $(t - O(rk), \epsilon k)$-pseudorandom. Then there is an algorithm D of complexity at most $t - O(rk)$ such that

$$\left| \mathop{\mathbb{P}}_{x \sim \{0,1\}^n}[D(G_k(x)) = 1] - \mathop{\mathbb{P}}_{z \sim \{0,1\}^{n+k}}[D(z) = 1] \right| > \epsilon k$$

We will then use the hybrid argument. We will define a sequence of distributions $H_0, \ldots, H_k$, the first is G_k's output, the last is uniformly random bits, and every two adjacent ones differ only in one invocation of G.



More specifically, define H_i to be the distribution where we intercept the output of the first i copies of G's, replace them with random bits, and run the rest of G_k as usual (see the above figure in which blue lines represent intercepted outputs). Then H_0 is just the distribution of the output of G_k, and H_k is the uniform distribution, as desired. Now

$$\begin{aligned} \epsilon k &< \left| \mathop{\mathbb{P}}_{z \sim H_0}[D(z) = 1] - \mathop{\mathbb{P}}_{z \sim H_k}[D(z) = 1] \right| \\ &= \left| \sum_{i=0}^{k-1} \left(\mathop{\mathbb{P}}_{z \sim H_i}[D(z) = 1] - \mathop{\mathbb{P}}_{z \sim H_{i+1}}[D(z) = 1] \right) \right| \end{aligned}$$

So there is an i such that

$$\left| \mathop{\mathbb{P}}_{z \sim H_i}[D(z) = 1] - \mathop{\mathbb{P}}_{z \sim H_{i+1}}[D(z) = 1] \right| > \epsilon$$

In both H_i and H_{i+1}, the first i bits $r_1, \ldots, r_i$ are random.

We now define a new algorithm D' that takes as input b, y and has output distribution H_i or H_{i+1} in two special cases: if b, y are drawn from $P(x), f(x)$, then D' has output distribution H_i; if b, y are drawn from (random bit)$, f(x)$, then D' has output distribution H_{i+1}. In other words, if b, y are $P(x), f(x)$, D' should output

$$r_1, \ldots, r_i, P(x), P(f(x)), \ldots, P(f^{(k-i-1)}(x)), f^{(k-i)}(x)$$

If b, y are (random bit)$, f(x)$, D' should output

$$r_1, \ldots, r_i, r_{i+1}, P(f(x)), \ldots, P(f^{(k-i-1)}(x)), f^{(k-i)}(x)$$

This suggests that D' on input b, y should pick random bits $r_1, \ldots, r_i$ and output $r_1, \ldots, r_i, b, P(y), \ldots, P(f^{(k-i-2)}(y$

We have

$$\begin{aligned} & \left| \mathbb{P}_{x\sim\{0,1\}^n}[D'(P(x)f(x))=1] - \mathbb{P}_{z\sim\{0,1\}^{n+1}}[D'(z)=1] \right| \\ = & \left| \mathbb{P}_{x\sim H_i}[D'(x)=1] - \mathbb{P}_{x\sim H_{i+1}}[D'(x)=1] \right| \\ > & \epsilon \end{aligned}$$

and $P(\cdot)$ is not (t,ϵ)-hard core. $\square$

Thinking about the following problem is a good preparation for the proof the main result of the next lecture.

Exercise 13 (Tree Composition of Generators) *Let $G : \{0,1\}^n \to \{0,1\}^{2n}$ be a (t,ϵ) pseudorandom generator computable in time r, let $G_0(x)$ be the first n bits of the output of $G(x)$, and let $G_1(x)$ be the last n bits of the output of $G(x)$.*

Define $G' : \{0,1\}^n \to \{0,1\}^{4n}$ as

$$G'(x) = G(G_0(x)), G(G_1(x))$$

Prove that G' is a $(t - O(r), 3\epsilon)$ pseudorandom generator.

Lecture 11

Pseudorandom Functions from Pseudorandom Generators

Summary

Today we show how to construct a pseudorandom function from a pseudorandom generator.

11.1 Pseudorandom generators evaluated on independent seeds

We first prove a simple lemma which we will need. This lemma simply says that if G is a pseudorandom generator with output length m, then if we evaluate G on k independent seeds the resulting function is still a pseudorandom generator with output length km.

Lemma 49 (Generator Evaluated on Independent Seeds) *Let $G : \{0,1\}^n \to \{0,1\}^m$ be a (t, ϵ) pseudorandom generator running in time t_g. Fix a parameter k, and define $G^k : \{0,1\}^{kn} \to \{0,1\}^{km}$ as*

$$G^k(x_1, \ldots, x_k) := G(x_1), G(x_2), \ldots, G(x_k)$$

Then G^k is a $(t - O(km + kt_g), k\epsilon)$ pseudorandom generator.

Proof: We will show that if for some (t, ϵ), G^k is not a (t, ϵ) psedorandom generator, then G cannot be a $(t + O(km + kt_g), \epsilon/k)$ pseudorandom generator.

The proof is by a hybrid argument. If G^k is not a (t, ϵ) pseudorandom generator, then there exists an algorithm D of complexity at most t, which distinguishes the output of G^k on a random seed, from a truly random string of km bits i.e.

$$\left| \mathop{\mathbb{P}}_{x_1,\ldots,x_k} [D(G(x_1), \ldots, G(x_k)) = 1] - \mathop{\mathbb{P}}_{r_1,\ldots,r_k} [D(r_1, \ldots, r_k) = 1] \right| \; > \; \epsilon$$

We can then define the hybrid distributions $H_0, \ldots, H_k$, where in H_i we relplace the first i outputs of the pseudorandom generator G by truly random strings.

$$H_i = (r_1, \ldots, r_i, G(x_{i+1}), \ldots, G(x_n))$$

As before, the above statement which says $|\mathbb{P}_{z\sim H_0}[D(z) = 1] - \mathbb{P}_{z\sim H_k}[D(z) = 1]| > \epsilon$ would imply that there exists an i between 0 and $k-1$ such that

$$\left| \mathop{\mathbb{P}}_{z\sim H_i}[D(z) = 1] - \mathop{\mathbb{P}}_{z\sim H_{i+1}}[D(z) = 1] \right| > \epsilon/k$$

We can now define an algorithm D' which violates the pseudorandom property of the generator G. Given an input $y \in \{0,1\}^m$, D' generates random strings $r_1, \ldots, r_i \in \{0,1\}^m$, $x_{i+2}, \ldots, x_k \in \{0,1\}^n$, and outputs $D(r_1, \ldots, r_i, y, G(x_{i+2}), \ldots, G(x_k))$. It then follows that

$$\mathop{\mathbb{P}}_{x\sim\{0,1\}^n}[D'(G(x)) = 1] \;=\; \mathop{\mathbb{P}}_{z\sim H_i}[D(z) = 1] \;\text{ and }\; \mathop{\mathbb{P}}_{r\sim\{0,1\}^m}[D'(r) = 1] \;=\; \mathop{\mathbb{P}}_{z\sim H_{i+1}}[D(z) = 1]$$

Hence, D' distinguishes the output of G on a random seed x from a truly random string r, with probability at least ϵ/k. Also, the complexity of D' is at most $t + O(km) + O(kt_g)$, where the $O(km)$ term corresponds to generating the random strings and the $O(kt_g)$ terms corresponds to evaluating G on at most k random seeds. □

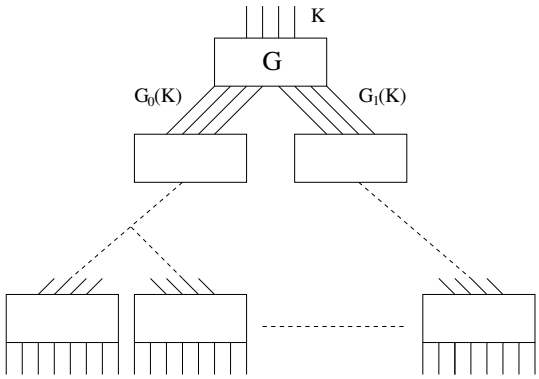
11.2 Construction of Pseudorandom Functions

We now describe the construction of a pseudorandom function from a pseudorandom generator. Let $G : \{0,1\}^n \to \{0,1\}^{2n}$ be a length-doubling pseudorandom generator. Define $G_0 : \{0,1\}^n \to \{0,1\}^n$ such that $G_0(x)$ equals the first n bits of $G(x)$, and define $G_1 : \{0,1\}^n \to \{0,1\}^n$ such that $G_1(x)$ equals the last n bits of $G(x)$.

The the GGM pseudorandom function based on G is defined as follows: for key $K \in \{0,1\}^n$ and input $x \in \{0,1\}^n$:

$$F_K(x) := G_{x_n}(G_{x_{n-1}}(\cdots G_{x_2}(G_{x_1}(K))\cdots)) \tag{11.1}$$

The evaluation of the function F can be visualized by considering a binary tree of depth n, with a copy of the generator G at each node. The root receives the input K and passes the outputs $G_0(K)$ and $G_1(K)$ to its two children. Each node of the tree, receiving an input z, produces the outputs $G_0(z)$ and $G_1(z)$ which are passed to its children if its not a leaf. The input x to the function F_K, then selects a path in this tree from the root to a leaf, and produces the output given by the leaf.



We will prove that if $G : \{0,1\}^n \to \{0,1\}^{2n}$ is a (t, ϵ) pseudorandom generator running in time t_g, then F is a $(t/O(n \cdot t_g), \epsilon \cdot nt)$ secure pseudorandom function.

11.2.1 Considering a tree of small depth

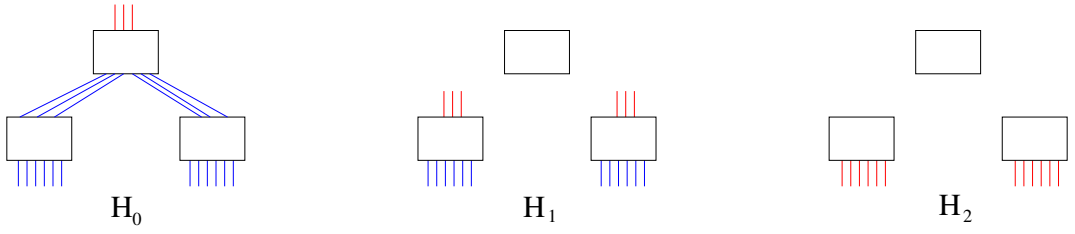
We will first consider a slightly simpler situation which illustrates the main idea. We prove that if G is (t, ϵ) pseudorandom and runs in time t_g, then the concatenated output of all the leaves in a tree with l levels, is $(t - O(2^l \cdot t_g), l2^l \cdot \epsilon)$ pseudorandom. The result is only meaninful when l is much smaller than n.

Theorem 50 *Suppose $G : \{0,1\}^n \to \{0,1\}^{2n}$ is a (t, ϵ) pseudorandom generator and G is computable in time t_g. Fix a constant l and define $F_K : \{0,1\}^l \to \{0,1\}^n$ as $F_K(y) := G_{y_l}(G_{y_{l-1}}(\cdots G_{y_2}(G_{y_1}(K))\cdots))$ Then $\overline{G} : \{0,1\}^n \to \{0,1\}^{2^l \cdot n}$ defined as*

$$\overline{G}(K) := (F_K(0^l), F_K(0^{l-1}1), \ldots, F_K(1^l))$$

is a $(t - O(2^l \cdot t_g), l \cdot 2^l \cdot \epsilon)$ pseudorandom generator.

PROOF: The proof is again by a hybrid argument. The hybrids we consider are easier to describe in terms of the tree with nodes as copies of G. We take H_i to be the distribution of outputs at the leaves, when the input to the nodes at depth i is replaced by truly random bits, ignoring the nodes at depth $i - 1$. Hence, H_0 is simply distributed as $\overline{G}(K)$ for a random K i.e. only the input to the root is random. Also, in H_l we replace the outputs at depth $l - 1$ by truly random strings. Hence, H_l is simply distributed as a random string of length $n \cdot 2^l$. The figure below shows the hybrids for the case $l = 2$, with red color indicating true randomness.



We will prove that $\overline{G}$ is not a (t,ϵ) secure pseudorandom generator, then G is not $(t+O(2^l \cdot t_g), \epsilon/(l \cdot 2^l))$ secure. If we assume that there is an algorithm D of complexity t such that

$$\left| \mathop{\mathbb{P}}_{x \sim \{0,1\}^n} [D(\overline{G}(x)) = 1] - \mathop{\mathbb{P}}_{r \sim \{0,1\}^{2^l \cdot n}} [D(r) = 1] \right| \ > \ \epsilon$$

then we get that there is an i such that $\left| \mathbb{P}_{z \sim H_i}[D(z) = 1] - \mathbb{P}_{z \sim H_{i+1}}[D(z) = 1] \right| \ > \ \epsilon/l$.

We now consider again the difference between H_i and H_{i+1}. In H_i the $2^i \cdot n$ bits which are the inputs to the nodes at depth i are replaced by random bits. These are then used to generate $2^{i+1} \cdot n$ bits which serve as inputs to nodes at depth $i+1$. In H_{i+1}, the inputs to nodes at depth $i+1$ are random.

Let $\overline{G}_{i+1} : \{0,1\}^{2^{i+1} \cdot n} \to \{0,1\}^{2^l \cdot n}$ denote the function which given $2^{i+1} \cdot n$ bits, treats them as inputs to the nodes at depth $i+1$ and evaluates the output at the leaves in the tree for $\overline{G}$. If $r_1, \ldots, r_{2^i} \sim \{0,1\}^{2n}$, then $\overline{G}_{i+1}(r_1, \ldots, r_{2^i})$ is distributed as H_{i+1}. Also, if $x_1, \ldots, x_{2^i} \sim \{0,1\}^n$, then $\overline{G}_{i+1}(G(x_1), \ldots, G(x_{2^i}))$ is distributed as H_i.

Hence, D can be used to create a distinguisher D' which distinguishes G evaluated on 2^i independent seeds, from 2^i random strings of length $2n$. In particular, for $z \in \{0,1\}^{2^{i+1} \cdot n}$, we take $D'(z) = D(G_{i+1}(z))$. This gives

$$\left| \mathop{\mathbb{P}}_{x_1, \ldots, x_{2^i}} [D'(G(x_1), \ldots, G(x_{2^i})) = 1] - \mathop{\mathbb{P}}_{r_1, \ldots, r_{2^i}} [D'(r_1, \ldots, r_{2^i}) = 1] \right| \ > \epsilon/l$$

Hence,D' distinguishes $G^{2^i}(x_1, \ldots, x_{2^i}) = (G(x_1), \ldots, G(x_{2^i}))$ from a random string. Also, G' has complexity $t + O(2^l \cdot t_g)$. However, by Lemma 49, if G^{2^i} is not $(t + O(2^l \cdot t_g), \epsilon/l)$ secure then G is not $(t + O(2^l \cdot t_g + 2^i \cdot n), \epsilon/(l \cdot 2^i))$ secure. Since $i \leq l$, this completes the proof. □

11.2.2 Proving the security of the GGM construction

Recall that the GGM function is defined as

$$F_K(x) := G_{x_n}(G_{x_{n-1}}(\cdots G_{x_2}(G_{x_1}(K)) \cdots)) \tag{11.2}$$

We will prove that

Theorem 51 *If $G : \{0,1\}^n \to \{0,1\}^{2n}$ is a (t,ϵ) pseudorandom generator and G is computable in time t_g, then F is a $(t/O(nt_g), \epsilon \cdot n \cdot t)$ secure pseudorandom function.*

Proof: As before, we assume that F is not a (t,ϵ) secure pseudorandom function, and will show that this implies G is not a $(t \cdot O(nt_g), \epsilon/(n \cdot t))$ pseudorandom generator. The assumption that F is not (t,ϵ) secure, gives that there is an algorithm A of complexity at most t which distinguishes F_K on a random seed K from a random function R, i.e.

$$\left| \mathop{\mathbb{P}}_{K} \left[A^{F_K(\cdot)} = 1 \right] - \mathop{\mathbb{P}}_{R} \left[A^{R(\cdot)} = 1 \right] \right| > \epsilon$$

We consider hybrids $H_0, \ldots, H_n$ as in the proof of Theorem 50. H_0 is the distribution of F_K for $K \sim \{0,1\}^n$ and H_n is the uniform distribution over all functions from $\{0,1\}^n$ to $\{0,1\}^n$. As before, there exists i such that

$$\left| \mathop{\mathbb{P}}_{h \sim H_i} \left[A^{h(\cdot)} = 1 \right] - \mathop{\mathbb{P}}_{h \sim H_{i+1}} \left[A^{h(\cdot)} = 1 \right] \right| > \epsilon/n$$

However, now we can no longer use A to construct a distinguisher for G^{2^i} as in Theorem 50 since i may now be as large as n. The important observation is that since A has complexity t, it can make at most t queries to the function it is given as an oracle. Since the (at most) t queries made by A will be paths in the tree from the root to the leaves, they can contain at most t nodes at depth $i+1$. Hence, to simulate the behavior of A, we only need to generate the value of a function distributed according to H_i or H_{i+1} at t inputs.

We will use this to contruct an algorithm D which distinguishes the output of G^t on t independent seeds from t random strings of length $2n$. D takes as input a string of length $2tn$, which we treat as t pairs $(z_{1,0}, z_{1,1}), \ldots, (z_{t,0}, z_{t,1})$ with each $z_{i,j}$ being of length n. When queried on an input $x \in \{0,1\}^n$, D will pick a pair $(z_{k,0}, z_{k,1})$ according to the first i bits of x (i.e. choose the randomness for the node at depth i which lies on the path), and then choose $z_{k,x_{i+1}}$. In particular, $D((z_{1,0}, z_{1,1}), \ldots, (z_{t,0}, z_{t,1}))$ works as below:

1. Start with counter $k = 0$.
2. Simulate A. When given a query x
   - Check if a pair $P(x_1, \ldots, x_i)$ has already been chosen from the first k pairs.
   - If not, set $P(x_1, \ldots, x_{i+1}) = k+1$ and set $k = k+1$.
   - Answer the query made by A as $G_{x_n}(\cdots G_{i+2}(z_{P(x_1,\ldots,x_{i+1}),x_{i+1}}) \cdots)$.
3. Return the final output given by A.

Then, if all pairs are random strings $r_1, \ldots, r_t$ of length $2n$, the answers received by A are as given by a oracle function distributed according to H_{i+1}. Hence,

$$\mathop{\mathbb{P}}_{r_1,\ldots,r_t}[D(r_1, \ldots, r_t) = 1] = \mathop{\mathbb{P}}_{h \sim H_{i+1}} \left[A^{h(\cdot)} = 1 \right]$$

Similarly, if the t pairs are outputs of the pseudorandom generator G on independent seeds $x_1, \ldots, x_t \in \{0,1\}^n$, then the view of A is the same as in the case with an oracle function distributed according to H_i. This gives

$$\mathop{\mathbb{P}}_{x_1,\ldots,x_t}[D(G(x_1), \ldots, G(x_t)) = 1] = \mathop{\mathbb{P}}_{h \sim H_i} \left[A^{h(\cdot)} = 1 \right]$$

Hence, D distinguishes the output of G^t from a random string with probability ϵ/n. Also, it runs in time $O(t \cdot n \cdot t_g)$. Then Lemma 49 gives that G is not $(O(t \cdot n \cdot t_g), \epsilon/(n \cdot t))$ secure. □

Lecture 12

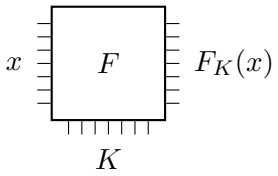
Pseudorandom Permutations from PRFs

Summary

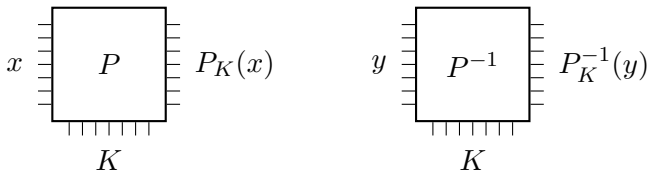
Given one way permutations (of which discrete logarithm is a candidate), we know how to construct pseudorandom functions. Today, we are going to construct pseudorandom permutations (block ciphers) from pseudorandom functions.

12.1 Pseudorandom Permutations

Recall that a pseudorandom function F is an efficient function : $\{0,1\}^k \times \{0,1\}^n \to \{0,1\}^n$, such that every efficient algorithm A cannot distinguish well $F_K(\cdot)$ from $R(\cdot)$, for a randomly chosen key $K \in \{0,1\}^k$ and a random function $R\colon \{0,1\}^n \to \{0,1\}^n$. That is, we want that $A^{F_K(\cdot)}$ behaves like $A^{R(\cdot)}$.



A pseudorandom permutation P is an efficient function : $\{0,1\}^k \times \{0,1\}^n \to \{0,1\}^n$, such that for every key K, the function P_K mapping $x \mapsto P_K(x)$ is a bijection. Moreover, we assume that given K, the mapping $x \mapsto P_K(x)$ is efficiently invertible (i.e. P_K^{-1} is efficient). The security of P states that every efficient algorithm A cannot distinguish well $\langle P_K(\cdot), P_K^{-1}(\cdot)\rangle$ from $\langle \Pi(\cdot), \Pi^{-1}(\cdot)\rangle$, for a randomly chosen key $K \in \{0,1\}^k$ and a random *permutation* $\Pi\colon \{0,1\}^n \to \{0,1\}^n$. That is, we want that $A^{P_K(\cdot), P_K^{-1}(\cdot)}$ behaves like $A^{\Pi(\cdot),\Pi^{-1}(\cdot)}$.



We note that the algorithm A is given access to both an oracle and its (supposed) inverse.

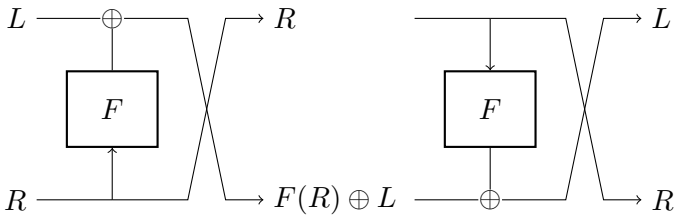
12.2 Feistel Permutations

Given *any* function $F\colon \{0,1\}^m \to \{0,1\}^m$, we can construct a permutation $D_F\colon \{0,1\}^{2m} \to \{0,1\}^{2m}$ using a technique named after Horst Feistel. The definition of D_F is given by

$$D_F(x, y) := y, F(y) \oplus x, \tag{12.1}$$

where x and y are m-bit strings. Note that this is an injective (and hence bijective) function, because its inverse is given by

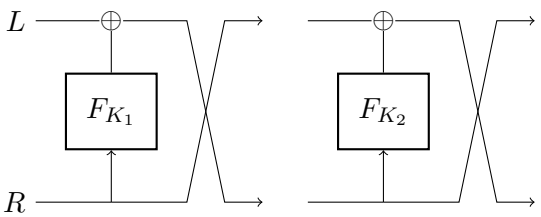
$$D_F^{-1}(z, w) := F(z) \oplus w, z. \tag{12.2}$$



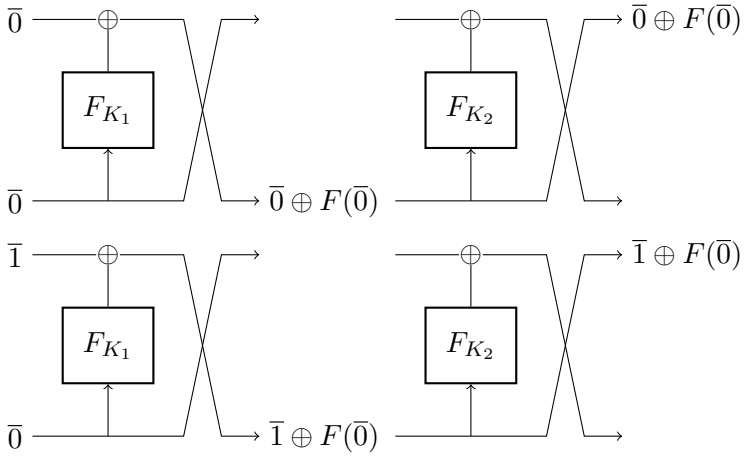
Also, note that D_F and D_F^{-1} are efficiently computable given F.

However, D_F needs not be a pseudorandom permutation even if F is a pseudorandom function, because the output of $D_F(x, y)$ must contain y, which is extremely unlikely for a truly random permutation.

To avoid the above pitfall, we may want to repeat the construction twice. We pick two independent random keys K_1 and K_2, and compose the permutations $P(\cdot) := D_{F_{K_2}}(D_{F_{K_1}}(\cdot))$.



Indeed, the output does not always contain part of the input. However, this construction is still insecure, no matter whether F is pseudorandom or not, as the following example shows.



Here, $\overline{0}$ denotes the all-zero string of length m, $\overline{1}$ denotes the all-one string of length m, and $F(\cdot)$ is $F_{K_1}(\cdot)$. This shows that, restricting to the first half, $P(\overline{00})$ is the complement of $P(\overline{10})$, regardless of F.

What happens if we repeat the construction three times? We still do not get a pseudorandom permutation.

Exercise 14 (Not Easy) *Show that there is an efficient oracle algorithm A such that*

$$\mathop{\mathbb{P}}_{\Pi:\{0,1\}^{2m}\to\{0,1\}^{2m}}[A^{\Pi,\Pi^{-1}} = 1] = 2^{-\Omega(m)}$$

where Π is a random permutation, but for every three functions F_1, F_2, F_3, if we define $P(x) := D_{F_3}(D_{F_2}(D_{F_1}(x)))$ we have

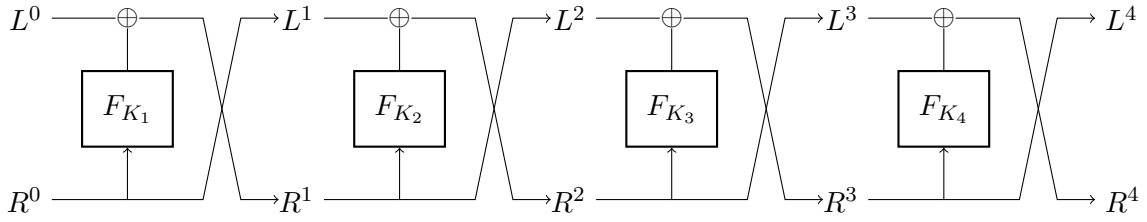
$$A^{P,P^{-1}} = 1$$

Finally, however, if we repeat the construction four times, with four independent pseudorandom functions, we get a pseudorandom permutation.

12.3 The Luby-Rackoff Construction

Let $F : \{0,1\}^k \times \{0,1\}^m \to \{0,1\}^m$ be a pseudorandom function, we define the following function $P : \{0,1\}^{4k} \times \{0,1\}^{2m} \to \{0,1\}^{2m}$: given a key $\overline{K} = \langle K_1, \ldots, K_4 \rangle$ and an input x,

$$P_{\overline{K}}(x) := D_{F_{K_4}}(D_{F_{K_3}}(D_{F_{K_2}}(D_{F_{K_1}}(x)))). \tag{12.3}$$



It is easy to construct the inverse permutation by composing their inverses backwards.

Theorem 52 (Pseudorandom Permutations from Pseudorandom Functions) *If F is a $(O(tr), \epsilon)$-secure pseudorandom function computable in time r, then P is a $(t, 4\epsilon + t^2 \cdot 2^{-m} + t^2 \cdot 2^{-2m})$ secure pseudorandom permutation.*

12.4 Analysis of the Luby-Rackoff Construction

Given four random functions $\overline{R} = \langle R_1, \ldots, R_4 \rangle$, $R_i : \{0,1\}^m \to \{0,1\}^m$, let $P_{\overline{R}}$ be the analog of Construction (12.3) using the random function R_i instead of the pseudorandom functions F_{K_i},

$$P_{\overline{R}}(x) = D_{R_4}(D_{R_3}(D_{R_2}(D_{R_1}(x)))) \tag{12.4}$$

We prove Theorem 52 by showing that

1. $P_{\overline{K}}$ is indistinguishable from $P_{\overline{R}}$ or else we can break the pseudorandom function
2. $P_{\overline{R}}$ is indistinguishable from a random permutation

The first part is given by the following lemma, which we prove via a standard hybrid argument.

Lemma 53 *If F is a $(O(tr), \epsilon)$-secure pseudorandom function computable in time r, then for every algorithm A of complexity $\leq t$ we have*

$$\left| \mathop{\mathbb{P}}_{\overline{K}}[A^{P_{\overline{K}}, P_{\overline{K}}^{-1}}() = 1] - \mathop{\mathbb{P}}_{\overline{R}}[A^{P_{\overline{R}}, P_{\overline{R}}^{-1}}() = 1] \right| \leq 4\epsilon \tag{12.5}$$

And the second part is given by the following lemma:

Lemma 54 *For every algorithm A of complexity $\leq t$ we have*

$$\left| \mathop{\mathbb{P}}_{\overline{R}}[A^{P_{\overline{R}}, P_{\overline{R}}^{-1}}() = 1] - \mathop{\mathbb{P}}_{\Pi}[A^{\Pi, \Pi^{-1}}() = 1] \right| \leq \frac{t^2}{2^{2m}} + \frac{t^2}{2^m}$$

where $\Pi : \{0,1\}^{2m} \to \{0,1\}^{2m}$ is a random permutation.

We now prove Lemma 53 using a hybrid argument.

PROOF: Consider the following five algorithms from $\{0,1\}^{2m}$ to $\{0,1\}^{2m}$:

- H_0: pick random keys K_1, K_2, K_3, K_4,
  $H_0(\cdot) := D_{F_{K_4}}(D_{F_{K_3}}(D_{F_{K_2}}(D_{F_{K_1}}(\cdot))))$;
- H_1: pick random keys K_2, K_3, K_4 and a random function $F_1 \colon \{0,1\}^m \to \{0,1\}^m$,
  $H_1(\cdot) := D_{F_{K_4}}(D_{F_{K_3}}(D_{F_{K_2}}(D_{F_1}(\cdot))))$;
- H_2: pick random keys K_3, K_4 and random functions $F_1, F_2 \colon \{0,1\}^m \to \{0,1\}^m$,
  $H_2(\cdot) := D_{F_{K_4}}(D_{F_{K_3}}(D_{F_2}(D_{F_1}(\cdot))))$;
- H_3: pick a random key K_4 and random functions $F_1, F_2, F_3 \colon \{0,1\}^m \to \{0,1\}^m$,
  $H_3(\cdot) := D_{F_{K_4}}(D_{F_3}(D_{F_2}(D_{F_1}(\cdot))))$;
- H_4: pick random functions $F_1, F_2, F_3, F_4 \colon \{0,1\}^m \to \{0,1\}^m$,
  $H_4(\cdot) := D_{F_4}(D_{F_3}(D_{F_2}(D_{F_1}(\cdot))))$.

Clearly, referring to (12.5), H_0 gives the first probability of using all pseudorandom functions in the construction, and H_4 gives the second probability of using all completely random functions. By triangle inequality, we know that

$$\exists i \quad \left| \mathbb{P}[A^{H_i, H_i^{-1}} = 1] - \mathbb{P}[A^{H_{i+1}, H_{i+1}^{-1}} = 1] \right| > \epsilon. \tag{12.6}$$

We now construct an algorithm $A'^{G(\cdot)}$ of complexity $O(tr)$ that distinguishes whether the oracle $G(\cdot)$ is $F_K(\cdot)$ or a random function $R(\cdot)$.

- The algorithm A' picks i keys $K_1, K_2, \ldots, K_i$ and initialize $4 - i - 1$ data structures $S_{i+2}, \ldots, S_4$ to $\emptyset$ to store pairs.
- The algorithm A' simulates $A^{O, O^{-1}}$. Whenever A queries O (or O^{-1}), the simulating algorithm A' uses the four compositions of Feistel permutations, where
  - On the first i layers, run the pseudorandom function F using the i keys $K_1, K_2, \ldots, K_i$;
  - On the i-th layer, run an oracle G;
  - On the remaining $4-i-1$ layers, simulate a random function: when a new value for x is needed, use fresh randomness to generate the random function at x and store the key-value pair into the appropriate data structure; otherwise, simply return the value stored in the data structure.

When G is F_K, the algorithm A'^G behaves like $A^{H_i, H_i^{-1}}$; when G is a random function R, the algorithm A'^G behaves like $A^{H_{i+1}, H_{i+1}^{-1}}$. Rewriting (12.6),

$$\left| \mathop{\mathbb{P}}_{K}[A'^{F_K(\cdot)} = 1] - \mathop{\mathbb{P}}_{R}[A'^{R(\cdot)} = 1] \right| > \epsilon,$$

and F is not $(O(tr), \epsilon)$-secure. □

We say that an algorithm A is *non-repeating* if it never makes an oracle query to which it knows the answer. (That is, if A is interacting with oracles g, g^{-1} for some permutation g, then A will not ask twice for $g(x)$ for the same x, and it will not ask twice for $g^{-1}(y)$ for the same y; also, after getting the value $y = g(x)$ in an earlier query, it will not ask for $g^{-1}(y)$ later, and after getting $w = g^{-1}(z)$ it will not ask for $g(w)$ later.)

We shall prove Lemma 54 for non-repeating algorithms. The proof can be extended to arbitrary algorithms with some small changes. Alternatively, we can argue that an arbitrary algorithm can be simulated by a non-repeating algorithm of almost the same complexity in such a way that the algorithm and the simulation have the same output given any oracle permutation.

In order to prove Lemma 54 we introduce one more probabilistic experiment: we consider the probabilistic algorithm $S(A)$ that simulates $A()$ and simulates every oracle query by providing a random answer. (Note that the simulated answers in the computation of SA may be incompatible with any permutation.)

We first prove the simple fact that $S(A)$ is close to simulating what really happen when A interacts with a truly random permutation.

Lemma 55 *Let A be a non-repeating algorithm of complexity at most t. Then*

$$\left|\mathbb{P}[S(A)=1] - \mathop{\mathbb{P}}_{\Pi}[A^{\Pi,\Pi^{-1}}()=1]\right| \leq \frac{t^2}{2\cdot 2^{2m}} \tag{12.7}$$

where $\Pi : \{0,1\}^{2m} \to \{0,1\}^{2m}$ is a random permutation.

Finally, it remains to prove:

Lemma 56 *For every non-repating algorithm A of complexity $\leq t$ we have*

$$\left|\mathop{\mathbb{P}}_{R}[A^{P_R,P_R^{-1}}()=1] - \mathbb{P}[S(A)=1]\right| \leq \frac{t^2}{2\cdot 2^{2m}} + \frac{t^2}{2^m}$$

It is clear that Lemma 54 follows from Lemma 55 and Lemma 56.

We now prove Lemma 55.

$$\begin{aligned}
&\left|\mathbb{P}[S(A)=1] - \mathop{\mathbb{P}}_{\Pi}[A^{\Pi,\Pi^{-1}}()=1]\right| \\
\leqslant\ & \mathbb{P}[\text{when simulating } S, \text{ get answers inconsistent with any permutation}] \\
\leqslant\ & \frac{1}{2^{2m}}(1+2+\cdots+t-1) \\
=\ & \binom{t}{2}\frac{1}{2^{2m}} \\
\leqslant\ & \frac{t^2}{2\cdot 2^{2m}}.
\end{aligned}$$

We shall now prove Lemma 56.

PROOF: The transcript of A's computation consists of all the oracle queries made by A. The notation $(x, y, 0)$ represents a query to the π oracle at point x while $(x, y, 1)$ is a query made to the π^{-1} oracle at y. The set T consists of all valid transcripts for computations where the output of A is 1 while $T^{'} \subset T$ consists of transcripts in T consistent with π being a permutation.

We write the difference in the probability of A outputting 1 when given oracles $(P_{\overline{R}}, P_{\overline{R}}^{-1})$ and when given a random oracle as in $S(A)$ as a sum over transcripts in T.

$$\begin{aligned} & \left|\mathbb{P}_{\overline{F}}[A^{P_{\overline{R}}, P_{\overline{R}}^{-1}}() = 1] - \mathbb{P}[S(A) = 1]\right| \\ = & \left|\sum_{\tau \in T}\left(\mathbb{P}_{\overline{F}}[A^{P_{\overline{R}}, P_{\overline{R}}^{-1}}() \leftarrow \tau] - \mathbb{P}[S(A) \leftarrow \tau]\right)\right| \end{aligned} \tag{12.8}$$

We split the sum over T into a sum over $T^{'}$ and $T \backslash T^{'}$ and bound both the terms individually. We first handle the simpler case of the sum over $T \setminus T^{'}$.

$$\begin{aligned} & \left|\sum_{\tau \in T \backslash T'}\left(\mathbb{P}_{\overline{F}}[A^{P_{\overline{R}}, P_{\overline{R}}^{-1}}() \leftarrow \tau] - \mathbb{P}[S(A) \leftarrow \tau]\right)\right| \\ = & \left|\sum_{\tau \in T \backslash T'}(\mathbb{P}[S(A) \leftarrow \tau])\right| \\ \leq & \frac{t^2}{2.2^{2m}} \end{aligned} \tag{12.9}$$

The first equality holds as a transcript obtained by running A using the oracle $(P_{\overline{R}}, P_{\overline{R}}^{-1})$ is always consistent with a permutation. The transcript generated by querying an oracle is inconsistent with a permutation iff. points x, y with $f(x) = f(y)$ are queried. $S(A)$ makes at most t queries to an oracle that answers every query with an independently chosen random string from $\{0,1\}^{2m}$. The probability of having a repetition is at most $(\sum_{i=1}^{t-1} i)/2^{2m} \leq t^2/2^{2m+1}$.

Bounding the sum over transcripts in $T^{'}$ will require looking into the workings of the construction. Fix a transcript $\tau \in T^{'}$ given by $(x_i, y_i, b_i), 1 \leq i \leq q$, with the number of queries $q \leq t$. Each x_i can be written as (L_i^0, R_i^0) for strings L_i^0, R_i^0 of length m corresponding to the left and right parts of x_i. The string x_i goes through 4 iterations of D using the function $F_k, 1 \leq k \leq 4$ for the kth iteration. The output of the construction after iteration k, $0 \leq k \leq 4$ for input x_i is denoted by (L_i^k, R_i^k).

Functions F_1, F_4 are said to be good for the transcript τ if the multisets $\{R_1^1, R_2^1, \cdots, R_q^1\}$ and $\{L_1^3, L_2^3, \cdots, L_q^3\}$ do not contain any repetitions. We bound the probability of F_1 being bad for τ by analyzing what happens when $R_i^1 = R_j^1$ for some i, j:

$$R_i^1 = L_i^0 \oplus F_1(R_i^0)$$

$$R_j^1 = L_j^0 \oplus F_1(R_j^0)$$

$$0 = L_i^0 \oplus L_j^0 \oplus F_1(R_i^0) \oplus F_1(R_j^0) \tag{12.10}$$

The algorithm A does not repeat queries so we have $(L_i^0, R_i^0) \neq (L_j^0, R_j^0)$. We observe that $R_i^0 \neq R_j^0$ as equality together with equation (12.10) above would yield $x_i = x_j$. This shows that equation (12.10) holds only if $F_1(R_j^0) = s \oplus F_1(R_i^0)$, for a fixed s and distinct strings R_i^0 and R_j^0. This happens with probability $1/2^m$ as the function F_1 takes values from $\{0,1\}^m$ independently and uniformly at random. Applying the union bound for all pairs i, j,

$$Pr_{F_1}[\exists i, j \in [q], \;\; R_i^1 = R_j^1] \leq \frac{t^2}{2^{m+1}} \tag{12.11}$$

We use a similar argument to bound the probability of F_4 being bad. If $L_i^3 = L_j^3$ for some i, j we would have:

$$\begin{aligned} L_i^3 &= R_i^4 \oplus F_4(L_i^4) \\ L_j^3 &= R_j^4 \oplus F_4(L_j^4) \end{aligned}$$

$$0 = R_i^4 \oplus R_j^4 \oplus F_4(L_i^4) \oplus F_4(L_j^4) \tag{12.12}$$

The algorithm A does not repeat queries so we have $(L_i^4, R_i^4) \neq (L_j^4, R_j^4)$. We observe that $L_i^4 \neq L_j^4$ as equality together with equation (12.12) above would yield $y_i = y_j$. This shows that equation (12.12) holds only if $F_4(L_j^4) = s^{'} \oplus F_4(L_i^4)$, for a fixed string $s^{'}$ and distinct strings L_i^4 and L_j^4. This happens with probability $1/2^m$ as the function F_4 takes values from $\{0,1\}^m$ independently and uniformly at random. Applying the union bound for all pairs i, j,

$$Pr_{F_4}[\exists i, j \in [q], \;\; L_i^3 = L_j^3] \leq \frac{t^2}{2^{m+1}} \tag{12.13}$$

Equations (12.11) and (12.13) together imply that

$$Pr_{F_1,F_4}[F_1, F_4 \text{ not good for transcript } \tau] \leq \frac{t^2}{2^m} \tag{12.14}$$

Continuing the analysis, we fix good functions F_1, F_4 and the transcript τ. We will show that the probability of obtaining τ as a transcript in this case is the same as the probability of obtaining τ for a run of $S(A)$. Let $\tau = (x_i, y_i, b_i), 1 \leq i \leq q \leq t$. We calculate the probability of obtaining y_i on query x_i over the choice of F_2 and F_3 .

The values of the input x_i are in bijection with pairs (L_i^1, R_i^1) while the values of the output y_i are in bijection with pairs (L_i^3, R_i^3), after fixing F_1 and F_4. We have the relations (from (**??**)(**??**)):

$$\begin{aligned} L_i^3 = R_i^2 &= L_i^1 \oplus F_2(R_i^1) \\ R_i^3 = L_i^2 \oplus F_3(R_i^2) &= R_i^1 \oplus F_3(L_i^3) \end{aligned}$$

These relations imply that (x_i, y_i) can be an input output pair if and only if we have $F_2(R_i^1), F_3(L_i^3) = (L_i^3 \oplus L_i^1, R_i^3 \oplus R_i^1)$. Since F_2 and F_3 are random functions with range

$\{0,1\}^m$, the pair (x_i, y_i) occurs with probability 2^{-2m}. The values R_i^1 and $L_i^3, (i \in [q])$ are distinct because the functions F_1 and F_4 are good. This makes the occurrence of (x_i, y_i) independent from the occurrence of (x_j, y_j) for $i \neq j$. We conclude that the probability of obtaining the transcript τ equals 2^{-2mq}.

The probability of obtaining transcript τ equals 2^{-2mq} in the simulation $S(A)$ as every query is answered by an independent random number from $\{0,1\}^{2m}$. Hence,

$$\begin{aligned} & \left| \sum_{\tau \in T'} \left(\mathop{\mathbb{P}}_{\overline{F}} [A^{P_{\overline{R}}, P_{\overline{R}}^{-1}}() \leftarrow \tau] - \mathbb{P}[S(A) \leftarrow \tau] \right) \right| \\ \leq & \left| \sum_{\tau \in T'} \mathop{\mathbb{P}}_{F_2, F_3} \left[A^{P_{\overline{R}}, P_{\overline{R}}^{-1}}() \leftarrow \tau | F_1, F_4 \text{ not good for } \tau \right] \right| \\ \leq & \frac{t^2}{2^m} \left| \sum_{\tau \in T'} \mathop{\mathbb{P}}_{F_2, F_3} [A^{P_{\overline{R}}, P_{\overline{R}}^{-1}}() \leftarrow \tau] \right| \\ \leq & \frac{t^2}{2^m} \end{aligned} \tag{12.15}$$

The statement of the lemma follows by adding equations (12.9) and (12.15) and using the triangle inequality. □

Lecture 13

Public-key Encryption

Summary

Today we begin to talk about public-key cryptography, starting from public-key encryption.

We define the public-key analog of the weakest form of security we studied in the private-key setting: message-indistinguishability for one encryption. Because of the public-key setting, in which everybody, including the adversary, has the ability to encrypt messages, this is already equivalent to CPA security.

We then describe the El Gamal cryptosystem, which is message-indistinguishable (and hence CPA-secure) under the plausible *Decision Diffie-Hellman* assumption.

13.1 Public-Key Cryptography

So far, we have studied the setting in which two parties, Alice and Bob, share a secret key K and use it to communicate securely over an unreliable channel. In many cases, it is not difficult for the two parties to create and share the secret key; for example, when we connect a laptop to a wireless router, we can enter the same password both into the router and into the laptop, and, before we begin to do online banking, our bank can send us a password in the physical mail, and so on.

In many other situations, however, the insecure channel is the only communication device available to the parties, so it is not possible to share a secret key in advance. A general problem of private-key cryptography is also that, in a large network, the number of required secret keys grows with the *square* of the size of the network.

In public-key cryptography, every party generates two keys: a secret key SK and a public key PK. The secret key is known only to the party who generated it, while the public key is known to everybody.

(For public-key cryptosystems to work, it is important that everybody is aware of, or has secure access to, everybody else's public key. A mechanism for the secure exchange of public

keys is called a *Public Key Infrastructure* (PKI). In a network model in which adversaries are *passive*, meaning that they only eavesdrop on communication, the parties can just send each other's public keys over the network. In a network that has *active* adversaries, who can inject their own packets and drop other users' packets, creating a public-key infrastructure is a very difficult problem, to which we may return when we talk about network protocols. For now we assume that either the adversary is passive or that a PKI is in place.)

As in the private-key setting, we will be concerned with two problems: *privacy*, that is the communication of data so that an eavesdropper can gain no information about it, and *authentication*, which guarantees to the recipient the identity of the sender. The first task is solved by public-key *encryption* and the second task is solved by *signature* schemes.

13.2 Public Key Encryption

A public-key encryption scheme is defined by three efficient algorithms (G, E, D) such that

- G takes no input and outputs a pair of keys (PK, SK)
- E, on input a public key PK and a plaintext message m outputs a ciphertext $E(PK, M)$.

  (Typically, E is a probabilistic procedure.)
- D, on input a secret key SK and ciphertext C, decodes C. We require that for every message m

$$\mathop{\mathbb{P}}_{\substack{(PK,SK)=G() \\ \text{randomness of } E}} [D(SK, E(PK, m)) = m] = 1$$

A basic definition of security is message-indistinguishability for one encryption.

Definition 57 *We say that a public-key encryption scheme (G, E, D) is (t, ϵ) message-indistinguishable if for every algorithm A of complexity $\leq t$ and for every two messages m_1, m_2,*

$$\left| \mathop{\mathbb{P}}_{\substack{(PK,SK)=G() \\ \textit{randomness of } E}} [A(PK, E(PK, m_1)) = 1] - \mathop{\mathbb{P}}_{\substack{(PK,SK)=G() \\ \textit{randomness of } E}} [A(PK, E(PK, m_2)) = 1] \right| \leq \epsilon$$

(From now on, we will not explicitly state the dependance of probabilities on the internal coin tosses of E, although it should always be assumed.)

Exercise 15 *Formalize the notion of CPA-security for public-key encryption. Show that if (G, E, D) is (t, ϵ) message indistinguishable, and $E(\cdot, \cdot)$ is computable in time $\leq r$, then (G, E, D) is also $(t/r, \epsilon)$ CPA-secure.*

13.3 Definitions of Security

There are three ways in which the definitions of security given in class differ from the way they are given in the textbook. The first one applies to all definitions, the second to definitions of encryption, and the third to CPA and CCA notions of security for encryption:

1. Our definitions usually refer to schemes of fixed key length and involve parameters t, ϵ, while the textbook definitions are asymptotic and parameter-free.

   Generally, one obtains the textbook definition by considering a family of constructions with arbitrary key length (or, more abstractly "security parameter") k, and allowing t to grow like any polynomial in k and requiring ϵ to be negligible. (Recall that a non-negative function $\nu(k)$ is negligible if for every polynomial p we have $\lim_{k\to\infty} p(k) \cdot \nu(k) = 0$.)

   The advantage of the asymptotic definitions is that they are more compact and make it easier to state the result of a security analysis. (Compare "if one-way permutations exist, then length-increasing pseudorandom generators exist" with "if $f : \{0,1\}^n \to \{0,1\}^n$ is a (t, ϵ) one-way permutation computable in time $\leq r$, then there is a generator $G : \{0,1\}^{2n} \to \{0,1\}^{2n+1}$ computable in time $r + O(n)$ that is $(t\epsilon^4/(n^2 \log n) - r\epsilon^{-4}n^3 \log n, \epsilon/3)$ pseudorandom")

   The advantage of the parametric definitions is that they make sense for fixed-key constructions, and that they make security proofs a bit shorter.

   (Every asymptotic security proof starts from "There is a polynomial time adversary A, an infinite set N of input lengths, and a polynomial $q()$, such that for every $n \in N \ldots$.)

2. Definitions of security for an encryption algorithm $E()$, after the proper quantifications, involve an adversary A (who possibly has oracles, etc.) and messages m_0, m_1; we require

$$|\,\mathbb{P}[A(E(m_0)) = 1] - \mathbb{P}[A(E(m_1)) = 1]| \leq \epsilon \tag{13.1}$$

   while the textbook usually has a condition of the form

$$\mathop{\mathbb{P}}_{b\in\{0,1\}}[A(E(m_b)) = b] \leq \frac{1}{2} + \frac{\epsilon}{2}$$

   The two conditions are equivalent since

$$\mathbb{P}[A(E(m_b)) = b] = \frac{1}{2}\,\mathbb{P}[A(E(m_0)) = 0] + \frac{1}{2}\,\mathbb{P}[A(E(m_1)) = 1$$

and the absolute value in (13.1) may be removed without loss of generality (at the cost of increasing the complextiy parameter by one).

3. Definitions of CPA and CCA security, as well as all definitions in the public-key setting, have a different structure in the book. The difference is best explained by an example. Suppose we have a public-key scheme (G, E, D) such that, for every valid public key pk, $E(pk, pk)$ has some distinctive pattern that makes it easy to distinguish it from other ciphertexts. This could be considered a security weakness because an eavesdropper is able to see if a party is sending a message that concerns the public key.

   This would not be a concern if the encryption mechanism were separate from the transport mechanism. For instance, if the application of this scheme occurred in such a way they two parties are securely communicating over an instant messaging client which exists in the application layer and encryption were occurring layers below in the transport layer. This abstraction of layers and separation of the encryption mechanism from the application abstracts away the notion that the public key could be encrypted with the public key. The messaging client is aware of the interface, but it never exposes the actual public or private key to the user, which prevents incorrectly using the cryptographic primitives.

   You can show as an exercise that if a secure public-key encryption scheme exists, then there is a public-key encryption scheme that is secure according to our definition from last lecture but that has a fault of the above kind.

   The textbook adopts a two-phase definition of security, in which the adversary is allowed to choose the two messages m_0, m_1 that it is going to try and distinguish, and the choice is done *after* having seen the public key. A random bit b is chosen and then the ciphertext of m_b is computed and given to the adversary. The adversary continues to have access to the Encryption function with the given public key. When the adversary is done, it outputs a guess called b' The output of this procedure is 1 when $b = b'$. A cryptosystem in which $E(pk, pk)$ can be distinguished from other ciphertexts violates this definition of security.

13.4 The Decision Diffie-Hellman Assumption

Fix a prime p and consider the group $\mathbb{Z}_p^*$, which consists of the elements in the set $\{1, \ldots, p-1\}$, along with the operator of multiplication mod p. This is a group because it includes 1, the operation is associative and commutative, and every element in $\{1, \ldots, p-1\}$ has a multiplicative inverse mod$p$ if p is prime.

It is a theorem which we will not prove that $\mathbb{Z}_p^*$ is a *cyclic group*, that is, there exists a $g \in \{1, \ldots, p-1\}$ such that $\{g^1, g^2, \ldots, g^{p-1}\}$ is the set of all elements in the group. That is,

each power of g generates a different element in the group, and all elements are generated. We call g a *generator*.

Now, pick a prime p, and assume we have a generator g of $\mathbb{Z}_p^*$. Consider the function that maps an element x to $g^x \mod p$. This function is a bijection, and its inverse is called the discrete log. That is, given $y \in \{1, \ldots, p-1\}$, there is a unique x such that $g^x \mod p = y$. x is the discrete logarithm of $y \mod p$.

It is believed that the discrete log problem is hard to solve. No one knows how to compute it efficiently (without a quantum algorithm). $\mathbb{Z}_p^*$ is but one family of groups for which the above discussion applies. In fact, our discussion generalizes to any cyclic group. Since the points on the elliptic curve (along with the addition operator) form a cyclic group, this generalization also captures elliptic curve cryptography.

While computing the discrete log is believed to be hard, modular exponentiation is efficient in $\mathbb{Z}_p^*$ using binary exponentiation, even when p is very large. In fact, in any group for which multiplication is efficient, exponentiation is also efficient, because binary exponentiation uses only $O(\log n)$ multiplications.

For the construction of the public-key cryptosystem that we will present, we will actually need an assumption slightly stronger than the assumption of the hardness of the discrete log problem. (As we shall see in the next section, this assumption is false for $\mathbb{Z}_p^*$, but it is believed to be true in other related groups.)

Definition 58 (Decision Diffie-Hellman Assumption) *A distribution $\mathcal{D}$ over triples $(\mathbb{G}, g, q)$, where $\mathbb{G}$ is a cyclic group of q elements and g is a generator, satisfies the (t, ϵ) Decision Diffie-Hellman Assumption if for every algorithm A of complexity $\leq t$ we have*

$$\left| \mathop{\mathbb{P}}_{\substack{(\mathbb{G},g,q) \sim \mathcal{D} \\ x,y,z \sim \{0,\ldots,q-1\}}} [A(\mathbb{G}, g, q, g^x, g^y, g^z) = 1] - \mathop{\mathbb{P}}_{\substack{(\mathbb{G},g,q) \sim \mathcal{D} \\ x,y \sim \{0,\ldots,q-1\}}} [A(\mathbb{G}, g, q, g^x, g^y, g^{xy}) = 1] \right| \leq \epsilon$$

Note that the Decision Diffie-Hellman assumption may be plausibly satisfied even by a *fixed* group $\mathbb{G}$ and a fixed generator g.

13.5 Decision Diffie Hellman and Quadratic Residues

In this section we show that the Decision Diffie Hellman assumption, however, is always false for groups of the type $\mathbb{Z}_p^*$.

To see why, we need to consider the notion of *quadratic residue* in $\mathbb{Z}_p^*$. An integer $a \in \{1, \ldots, p-1\}$ is a quadratic residue if there exists $r \in \{1, \ldots, p-1\}$ such that $a = r^2 \bmod p$. (In such a case, we say that r is a *square root* of a.)

As an example, let $p = 11$. The elements of $\mathbb{Z}_p^*$ are,

$$\mathbb{Z}_p^* = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$$

The squares of the elements, mod p:

$$1, 4, 9, 5, 3, 3, 5, 9, 4, 1$$

The quadratic residues of $\mathbb{Z}_p^*$:

$$\mathbb{Q}_p = \{1, 4, 9, 5, 3\}$$

For every odd primes p, exactly $(p-1)/2$ of the elements of $\mathbb{Z}_p^*$ are quadratic residues, and each has two square roots. Furthermore, there is an efficient algorithm (polynomial in the number of digits of a) to check whether a is a quadratic residue. This fact immediately gives an algorithm that contradicts (**??**) if we take $\mathbb{G}$ to be $\mathbb{Z}_p^*$, when $\epsilon < 1/4$. To see why this is so, first consider g, a generator in $\mathbb{Z}_p^*$. Then $g^x \bmod p$ is a quadratic residue if and only if x is even:

Since g is a generator, $\mathbb{Z}_p^*$ can be written as,

$$\mathbb{Z}_p^* = \{g^0, g^1, \ldots, g^{p-2}\}$$

Squaring every element in $\mathbb{Z}_p^*$ gives the quadratic residues:

$$\mathbb{Q}_p = \{g^0, g^2, \ldots, g^{2(p-2)}\}$$

When x is even, g^x is the square of $g^{x/2} \in \mathbb{Z}_p^*$. All the quadratic residues can be written as $g^x, x = 2i$ so x must be even.

In (**??**) g^z is a random element in $\mathbb{Z}_p^*$ so it is a quadratic residue with probability $1/2$. But $g^{x \cdot y}$ is a quadratic residue with probability $3/4$ (if $x, y,$ or x and y are even). Since there is an efficient algorithm to check if a value is a quadratic residue, the algorithm A can easily distinguish between $(\mathbb{G}, g, q, g^x, g^y, g^z)$ and $(\mathbb{G}, g, q, g^x, g^y, g^{x \cdot y}$ with probability $> 1/4$ by outputting 1 when it finds that the final input parameter (either g^z or $g^{x \cdot y}$) is a quadratic residue.

Note, however, that the set $\mathbb{Q}_p$ of quadratic residues of $\mathbb{Z}_p^*$ is itself a cyclic group, and that if g is a generator of $\mathbb{Z}_p^*$ then $g^2 \bmod p$ is a generator of $\mathbb{Q}_p$.

$\mathbb{Q}_p$ is a cyclic group:

- 1 is a quadratic residue
- if $r_1 = x_1^2$, $r_2 = x_2^2$ are quadratic residues, then $r_1 \cdot r_2 = (x_1 \cdot x_2)^2$ is a quadratic residue.
- if $r = x^2$ is a quadratic residue then $r^{-1} = (x^{-1})^2$ is a quadratic residue

If g is a generator of $\mathbb{Z}_p^*$ then $g^2 \bmod p$ is a generator of $\mathbb{Q}_p$:

$$\mathbb{Q}_p = \{g^0, g^2, \ldots g^{2(p-2)}\}$$

Which is exactly

$$\mathbb{Q}_p = \{(g^2)^0, (g^2)^1, \ldots (g^2)^{(p-2)}\}$$

It is believed that if p is a prime of the form $2q+1$, where q is again prime then taking $\mathbb{G} = \mathbb{Q}_p$ and letting g be any generator of $\mathbb{G}$ satisfies (**??**), with t and $1/\epsilon$ exponentially large in the number of digits of q.

13.6 El Gamal Encryption

The El Gamal encryption scheme works as follows. Let $\mathcal{D}$ be a distribution over $(\mathbb{G}, g, q)$ that satisfies the Decision Diffie-Hellman assumption:

- $G()$ samples $(\mathbb{G}, g, q)$, and picks a random number $x \in \{0, \ldots, q-1\}$.
  - $PK = (\mathbb{G}, g, q, g^x)$
  - $SK = (\mathbb{G}, g, q, x)$
- $E((\mathbb{G}, g, q, a), m)$:
  - pick at random $r \in \{0, \ldots, q-1\}$
  - output $(g^r, a^r \cdot m)$
- $D((\mathbb{G}, g, q, x), (c_1, c_2))$
  - Compute $b := c_1^x$
  - Find the multiplicative inverse b' of b
  - output $b' \cdot c_2$

The decryption algorithm works as follows. c_1 is g^r (as returned by E), so $b = g^{rx}$. c_2, as returned by E, is $a^r \cdot m$, where a is g^x. This means that $c_2 = g^{rx} \cdot m$. We see that $c_2 = b \cdot m$, which is why multiplying c_2 by b^{-1} correctly yields m.

Theorem 59 *Suppose $\mathcal{D}$ is a distribution that satisfies the (t, ϵ) Decision Diffie-Hellman assumption and that it is possible to perform multiplication in time $\leq r$ in the groups $\mathbb{G}$ occurring in $\mathcal{D}$.*

Then the El Gamal cryptosystem is $(t - r, 2\epsilon)$ message-indistinguishable.

PROOF: Let A be an algorithm of complexity $\leq t - r$ and fix any two messages m_1, m_2. We want to prove

$$|\,\mathbb{P}[A(\mathbb{G},g,q,g^x,g^r,g^{xr}\cdot m_1)=1]-\mathbb{P}[A(\mathbb{G},g,q,g^x,g^r,g^{xr}\cdot m_2)=1]|\le 2\epsilon$$

(From now, we shall not write the dependency on $\mathbb{G}, g, q$.)

We utilize a variant of the encryption algorithm that uses a random group element g^y (instead of g^{xr}) as the multiplier for m.[1]

$$|\,\mathbb{P}[A(g^x,g^r,g^{xr}\cdot m_1)=1]-\mathbb{P}[A(g^x,g^r,g^{xr}\cdot m_2)=1]| \tag{13.2}$$

$$\le |\,\mathbb{P}[A(\mathbb{G},g,q,g^x,g^r,g^{xr}\cdot m_1)=1]-\mathbb{P}[A(g^x,g^r,g^y\cdot m_1)=1]| \tag{13.3}$$

$$+|\,\mathbb{P}[A(g^x,g^r,g^y\cdot m_1)=1]-\mathbb{P}[A(g^x,g^r,g^y\cdot m_2)=1]| \tag{13.4}$$

$$+|\,\mathbb{P}[A(g^x,g^r,g^{xr}\cdot m_2)=1]-\mathbb{P}[A(g^x,g^r,g^y\cdot m_2)=1]| \tag{13.5}$$

Each of the expressions in (13.3) and (13.5) is $\le \epsilon$ due to the (t,ϵ) Decision Diffie-Hellman Assumption. There is an extra factor of m_1 or m_2, respectively, but the D.D.H. still holds in this case. Informally, multiplying a group element that looks random by a fixed element yields another random-looking element. We can formalize this as follows:

We claim that if G, g, q satisfies the (t,ϵ) Decision Diffie-Hellman Assumption, and r is an upper bound to the time it takes to compute products in G, then for all group elements m and for all algorithms A of complexity $\le t-r$

$$|\,\mathbb{P}[A(g^x,g^y,g^{xy}\cdot m)=1]-\mathbb{P}[A(g^x,g^y,g^z\cdot m)=1]|\le\epsilon$$

To prove this claim, suppose to the contrary that there exists an algorithm A of complexity $\le t-r$ and a group element m such that the above difference is $>\epsilon$.

Let A' be an algorithm that on input (G,g,q,a,b,c) outputs $A(G,g,q,a,b,c\cdot m)$. Then A' has complexity $\le t$ and

$$\begin{aligned} & |\,\mathbb{P}[A'(g^x,g^y,g^{xy})=1]-\mathbb{P}[A'(g^x,g^y,g^z)=1]| \\ = \;& |\,\mathbb{P}[A(g^x,g^y,g^{xy}\cdot m)=1]-\mathbb{P}[A(g^x,g^y,g^z\cdot m)=1]| \\ > \;& \epsilon \end{aligned}$$

which contradicts the (t,ϵ) Decision Diffie-Hellman Assumption.

Next, we consider (13.4). This is an instance of "perfect security," since distinguishing between m_1 and m_2 requires distinguishing two completely random elements. (Again, we use the fact that multiplying a random element by a fixed element yields a random element.) Thus, the expression in line (13.4) is equal to 0.

This means that (13.2) is at most 2ϵ. □

[1] This would not actually function as an encryption algorithm, but we can still consider it, as the construction is well-defined.

Lecture 14

CPA-secure Public-Key Encryption

Summary

Today we discuss the three ways in which definitions of security given in class differ from the way they are given in the Katz-Lindell textbook,.

Then we study the security of *hybrid* encryption schemes, in which a public-key scheme is used to encode the key for a private-key scheme, and the private-key scheme is used to encode the plaintext.

We also define RSA and note that in order to turn RSA into an encryption scheme we need a mechanism to introduce randomness.

Finally, we abstract RSA via the notion of a "family of trapdoor permutations," and show how to achieve CPA-secure encryption from any family of trapdoor permutations.

14.1 Hybrid Encryption

Let (G_1, E_1, D_1) be a public-key encryption scheme and (E_2, D_2) a private-key encryption scheme.

Consider the following *hybrid* scheme (G, E, D):

- $G()$: same as $G_1()$
- $E(pk, m)$: pick a random key K for E_2, output $(E_1(pk, K), E_2(K, m))$
- $D(sk, (C_1, C_2))$: output $D_2(D_1(sk, C_1), C_2)$

A hybrid approach to public key cryptography is often desired due to the fact that public key operations are computationally expensive (i.e modular exponentiation), while symmetric key cryptosystem are usually more efficient. The basic idea behind the hybrid approach that if we encrypt the symmetric private key with the public key and encrypt the message

with the symmetric private key, only the small symmetric private key needs to be encrypted with the public key and symmetric key encryption/decryption can take place on the actual message. This allows for efficient computation of the message encryption and decryption while only using asymmetric key cryptography for transmitting the symmetric shared secret. This construction makes encryption and decryption much more efficient while still ensuring the construction has message indistinguishability and CPA security.

Theorem 60 *Suppose* (G_1, E_1, D_1) *is* (t, ϵ_1)*-secure for one encryption and* (E_2, D_2) *is* (t, ϵ_2)*-secure for one encryption. Suppse also that* E_1, E_2 *have running time* $\leq r$*.*

Then (G, E, D) *is* $(t - 2r, 2\epsilon_1 + \epsilon_2)$*-secure for one encryption.*

We begin by assuming the conclusion of the theorem is false, that is (G, E, D) is not $(t - 2r, 2\epsilon_1 + \epsilon_2)$-secure.

Suppose there is an adversary A, that runs in time t' and there are two messages m_0 and m_1 such that:

$$|\,\mathbb{P}[A(pk, E(pk, m_0)) = 1] - \mathbb{P}[A(pk, E(pk, m_1)) = 1]| > \ 2\epsilon_1 + \epsilon_2$$

Then the definition of $E()$ is applied, so

$$|\,\mathbb{P}[A(pk, E_1(pk, K), E_2(K, m_0)) = 1] - \mathbb{P}[A(pk, E_1(pk, K), E_2(K, m_1) = 1]| > 2\epsilon_1 + \epsilon_2$$

We then apply a hybrid argument in which the hybrid distributions have $E_1(pk, \mathbf{0})$ instead of $E_1(pk, K)$. ($\mathbf{0}$ denotes a string of zeroes; any other fixed string could be used in the proof.)

Producing:

$$2\epsilon_1 + \epsilon_2 < \mathbb{P}[A(pk, E_1(pk, K), E_2(K, m_0)) = 1] - \mathbb{P}[A(pk, E_1(pk, K), E_2(K, m_1) = 1] \leq$$

$$\begin{aligned}
&\|\,\mathbb{P}[A(pk, E_1(pk, K), E_2(K, m_0)) = 1] - \mathbb{P}[A(pk, E_1(pk, \mathbf{0}), E_2(K, m_0)) = 1]\| + \\
&\|\,\mathbb{P}[A(pk, E_1(pk, \mathbf{0}), E_2(K, m_0)) = 1] - \mathbb{P}[A(pk, E_1(pk, \mathbf{0}), E2(K, m_1)) = 1]\| + \\
&\|\,\mathbb{P}[A(pk, E_1(pk, \mathbf{0}), E2(K, m_1)) = 1] - \mathbb{P}[A(pk, E_1(pk, K), E2(K, m_1)) = 1]\|
\end{aligned}$$

This means that at least one of the following cases must happen:

a) the first difference is at least ϵ_1

b) the second difference is at least ϵ_2

c) the third difference is at least ϵ_1

If (a) or (c) are true, then it means that there is a message m such that:

$$\| \mathbb{P}[A(pk, E_1(pk, K), E_2(K, m)) = 1] - \mathbb{P}[A(pk, E_1(pk, 0), E_2(K, m)) = 1]\| > \epsilon_1$$

Then there must exist one fixed K^* such that

$$\| \mathbb{P}[A(pk, E_1(pk, K^*), E_2(K, m)) = 1] - \mathbb{P}[A(pk, E_1(pk, \hat{0}), E2(K, m)) = 1]\| > \epsilon_1$$

and then we define an algorithm A' of complexity at most t such that:

$$\| \mathbb{P}[A'(pk, E_1(pk, K^*)) = 1] - \mathbb{P}[A'(pk, E_1(pk, \hat{0})) = 1]\| > \epsilon_1$$

which contradicts the security of E_1.
If (b) is true, then we define an algorithm A'' of complexity at most t such that:

$$\| \mathbb{P}[A''(E_2(K, m_0)) = 1] - \mathbb{P}[A''(E_2(K, m_1)) = 1]\| > \epsilon_2$$

which contradicts the security of E_2.

14.2 RSA

The RSA function has the same "syntax" of a public-key encryption scheme:

- Key generation: Pick two distinct prime numbers p, q, compute $N := pq$, and find integers e, d such that

  $$e \cdot d \equiv 1 \pmod{()p-1) \cdot (q-1)}$$

  Set the public key to (N, e) and the private key to (N, d)
- "Encryption:" given $x \in \mathbb{Z}_N$ and public key (N, e), output

  $$E_{RSA}(x, (N, e)) := x^e \bmod N$$
- "Decryption:" given $y \in \mathbb{Z}_N$ and secret key (N, d), output

  $$D_{RSA}(y, (N, d)) := y^d \bmod N$$

It is a standard calculation using the Chinese remainder theorem and Fermat's little theorem that $E_{RSA}(\cdot, (N, e))$ and $D_{RSA}(\cdot, (N, d))$ are permutations over $\mathbb{Z}_N$, and they are one the inverse of the other.

This is, however, not a secure encryption scheme because it is *deterministic*, and it suffers from several weaknesses that can exploited in practice.

A conjectural way to turn RSA into a CPA-secure encryption scheme is to employ it to encrypt plaintexts whose length is only about $\frac{1}{2}\log N$, and then pad the plaintext with $\frac{1}{2}\log N$ random bits before applying E_{RSA}. (Other choices for the length of the plaintext and the amount of randomness are possible, half-half is just an example.)

The assumption that this padded-RSA is CPA secure is a very strong one. In the next lecture we will see how to turn RSA into a CPA secure encryption scheme under the minimal assumption that E_{RSA} is hard to invert on a random input for an adversary that knows the public key but not the secret key.

14.3 Trapdoor Permutations and Encryption

A *family of trapdoor permutations* is a triple of algorithms (G, E, D) such that:

1. $G()$ is a randomized algorithm that takes no input and generates a pair (pk, sk), where pk is a *public key* and sk is a *secret key*;
2. E is a deterministic algorithm such that, for every fixed public key pk, the mapping $x \to E(pk, x)$ is a bijection;
3. D is a deterministic algorithm such that for every possible pair of keys (pk, sk) generated by $G()$ and for every x we have

$$D(sk, E(pk, x)) = x$$

That is, syntactically, a family of trapdoor permutations is like an encryption scheme except that the "encryption" algorithm is deterministic. A family of trapdoor permutations is secure if inverting $E()$ for a random x is hard for an adversary that knows the public key but not the secret key. Formally,

Definition 61 *A family of trapdoor permutations (G, E, D) is (t, ϵ)-secure if for every algorithm A of complexity $\leq t$*

$$\mathop{\mathbb{P}}_{(pk,sk)\leftarrow G(),x}[A(pk, (E(pk, x))) = x] \leq \epsilon$$

It is believed that RSA defines a family of trapdoor permutations with security parameters t and $1/\epsilon$ that grow exponentially with the number of digits of the key. Right now the fastest factoring algorithm is believed to run in time roughly $2^{O(k^{1/3})}$, where k is the number of digits, and so RSA with k-bit key can also be broken in that much time. In 2005, an RSA key of 663 bits was factored, with a computation that used about 2^{62} elementary

operations. RSA with keys of 2048 bits may plausibly be $(2^{60}, 2^{-30})$-secure as a family of trapdoor permutations.

In order to turn a family of trapdoor permutations into a public-key encryption scheme, we use the notion of a *trapdoor predicate*.

Definition 62 *Let (G, E, D) be a family of trapdoor permutations, where E takes plaintexts of length m. A boolean function $P : \{0,1\}^m \to \{0,1\}$ is a (t, ϵ)-secure trapdoor predicate for (G, E, D) if for every algorithm A of complexity $\leq t$ we have*

$$\mathop{\mathbb{P}}_{(pk,sk)\leftarrow G(),\ x}[A(pk, E(pk, x)) = P(x)] \leq \frac{1}{2} + \epsilon$$

Remark 63 *The standard definition is a bit different. This simplified definition will suffice for the purpose of this section, which is to show how to turn RSA into a public-key encryption scheme.*

Essentially, P is a trapdoor predicate if it is a hard-core predicate for the bijection $x \to E(pk, x)$. If (G, E, D) is a secure family of trapdoor permutations, then $x \to E(pk, x)$ is one-way, and so we can use Goldreich-Levin to show that $\langle x, r\rangle$ is a trapdoor predicate for the permutation $E'(pk, (x, r)) = E(pk, x), r$.

Suppose now that we have a family of trapdoor permutations (G, E, D) and a trapdoor predicate P. We define the following encryption scheme (G', E', D') which works with *one-bit* messages:

- $G'()$: same as $G()$
- $E'(pk, b)$: pick random x, output $E(pk, x), P(x) \oplus b$
- $D'(pk, (C, c)) := P(D(pk, C)) \oplus c$

Theorem 64 *Suppose that P is (t, ϵ)-secure trapdoor predicate for (G, E, D), then (G', E', D') as defined above is $(t - O(1), 2\epsilon)$-CPA secure public-key encryption scheme.*

Proof: In Theorem 2 of Lecture 13 we proved that if f is a permutation and P is a (t, ϵ) hard-core predicate for f, then for every algorithm A of complexity $\leq t - O(1)$ we have

$$|\,\mathbb{P}[A(f(x), P(x)) = 1] - \mathbb{P}[A(f(x)), r) = 1]| \leq \epsilon \tag{14.1}$$

where r is a random bit. Since

$$\mathbb{P}[A(f(x)), r) = 1] = \frac{1}{2}\,\mathbb{P}[A(f(x), P(x)) = 1] + \frac{1}{2}\,\mathbb{P}[A(f(x), 1 \oplus P(x)) = 1]$$

we can rewrite (14.1) as

$$|\,\mathbb{P}[A(f(x), P(x)) = 1] - \mathbb{P}[A(f(x)), P(x) \oplus 1) = 1]| \leq 2\epsilon$$

And taking f to be our trapdoor permutation,

$$|\,\mathbb{P}[A(E(pk,x),P(x)\oplus 0)=1]-\mathbb{P}[A(E(pk,x),P(x)\oplus 1)=1]|\leq 2\epsilon$$

that is,

$$|\,\mathbb{P}[A(E'(pk,0))=1]-\mathbb{P}[A(E'(pk,1))=1]|\leq 2\epsilon$$

showing that E' is $(t-O(1),2\epsilon)$ CPA secure. $\square$

The encryption scheme described above is only able to encrypt a one-bit message. Longer messages can be encrypted by encrypting each bit separately. Doing so, however, has the undesirable property that an ℓ bit message becomes an $\ell \cdot m$ bit ciphertext, if m is the input length of P and $E(pk,\cdot)$. A "cascading construction" similar to the one we saw for pseudorandom generators yields a secure encryption scheme in which an ℓ-bit message is encrypted as a cyphertext of length only $\ell + m$.

Lecture 15

Signature Schemes

Summary

Today we begin to talk about *signature schemes*.

We describe various ways in which "textbook RSA" signatures are insecure, develop the notion of *existential unforgeability under chosen message attack*, analogous to the notion of security we gave for authentication, and discuss the difference between authentication in the private-key setting and signatures in the public-key setting.

As a first construction, we see Lamport's *one-time signatures* based on one-way functions, and we develop a rather absurdly inefficient stateful scheme based on one-time signatures. The scheme will be interesting for its idea of "refreshing keys" which can be used to design a stateless, and reasonably efficient, scheme.

15.1 Signature Schemes

Signatures are the public-key equivalents of MACs. The set-up is that Alice wants to send a message M to Bob, and convince Bob of the authenticity of the message. Alice generates a public-key/ secret-key pair (pk, sk), and makes the public key known to everybody (including Bob). She then uses an algorithm $Sign()$ to compute a *signature* $\sigma := Sign(sk, M)$ of the message; she sends the message along with the signature to Bob. Upon receiving M, σ, Bob runs a verification algorithm $Verify(pk, M, \sigma)$, which checks the validity of the signature. The security property that we have in mind, and that we shall formalize below, is that while a valid signature can be efficiently generated given the secret key (via the algorithm $Sign()$), a valid signature cannot be efficiently generated without knowing the secret key. Hence, when Bob receives a message M along with a signature σ such that $Verify(pk, M, \sigma)$ outputs "valid," then Bob can be confident that M is a message that came from Alice. (Or, at least, from a party that knows Alice's secret key.)

There are two major differences between signatures in a public key setting and MAC in a private key setting:

1. **Signatures are transferrable:** Bob can forward (M, σ) to another party and the other party can be confident that Alice actually sent the message, assuming a public key infrastructure is in place (or, specifically that the other party has Alice's public key).
2. **Signature are non-repudiable:** Related to the first difference, once Alice signs the message and sends it out, she can no longer deny that it was actually her who sent the message.

Syntactically, a signature scheme is a collection of three algorithms $(Gen, Sign, Verify)$ such that

- $Gen()$ takes no input and generates a pair (pk, sk) where pk is a public key and sk is a secret key;
- Given a secret key sk and a message M, $Sign(sk, M)$ outputs a signature σ;
- Given a public key pk, a message M, and an alleged signature σ, $Verify(sk, M, \sigma)$ outputs either "valid" or "invalid", with the property that for every public key/ secret key pair (pk, sk), and every message M,

$$Verify(pk, M, Sign(sk, M)) = \text{ "valid"}$$

The notion of a signature scheme was described by Diffie and Hellman without a proposed implementation. The RSA paper suggested the following scheme:

- Key Generation: As in RSA, generate primes p, q, generate e, d such that $ed \equiv 1 \bmod (p-1) \cdot (q-1)$, define $N := p \cdot q$, and let $pk := (N, e)$ and $sk := (N, d)$.
- Sign: for a message $M \in \{0, \ldots, N-1\}$, the signature of M is $M^d \bmod N$
- Verify: for a message M and an alleged signature σ, we check that $\sigma^e \equiv M \pmod N$.

Unfortunately this proposal has several security flaws.

- Generating random messages with valid signatures: In this attack, you pick a random string σ and compute $M := \sigma^e \bmod N$ and now σ is a valid signature for M. This can be an effective attack if there are a large number of messages that are useful to forge.
- Combining signed messages to create the signature of their products: Suppose you have M_1 and M_2 with valid signatures σ_1 and σ_2 respectively. Note that $\sigma_1 := M_1^e \bmod N$ and $\sigma_2 := M_2^e \bmod N$. We can now generate a valid signature for $M := M_1 \cdot M_2 \bmod N$:

$$\sigma_M = M^e \bmod N$$

$$= (M_1 \cdot M_2)^e \bmod N$$

$$= M_1^E \cdot M_2^e \bmod N$$

$$= \sigma_1 \cdot \sigma_2 \bmod N$$

- Creating signatures for arbitrary messages: Suppose the adversary wants to forge message M. If it is able to get a valid signature for a random message m_1 and a specifically chosen message $m_2 := M/m_1 \bmod N$ then the adversary can use the second attack to calculate a valid signature for M (i.e. $m_1 \cdot m_2 \bmod N = M$ and $\sigma_1 \cdot \sigma_2 \bmod N = \sigma_M$.

Ideally, we would like the following notion of security, analogous to the one we achieved in the secret-key setting.

Definition 65 *A signature scheme (G, S, V) is (t, ϵ) existentially unforgeable under a chosen message attack if for every algorithm A of complexity at most t, there is probability $\leq \epsilon$ that A, given a public key and a signing oracle, produces a valid signature of a message not previously sent to the signing oracle.*

It was initially thought no signature scheme could meet this definition. The so called "paradox" of signature schemes was that it seemed impossible to both have a scheme in which forgery is difficult (that is, equivalent to factoring) while simultaneously having this scheme be immune to chosen message attacks. Essentially the paradox is that the proof that a scheme is difficult to forge will generally use a black-box forging algorithm to then construct a factoring algorithm. However, if this scheme were subject to a chosen message attack, a new algorithm could be constructed which would simulate the constructed factoring algorithm and totally break the signature scheme. This new algorithm would be exactly the same as the one used in the forgery proof except every query to the black-box forging algorithm instead becomes one of the messages sent to the oracle. Goldwasser et al.'s paper "A Digital Signature Scheme Secure Against Adaptive Chosen-Message Attacks" gives further details and describes breaking the paradox.

15.2 One-Time Signatures and Key Refreshing

We begin by describing a simple scheme which achieves a much weaker notion of security.

Definition 66 (One-Time Signature) *A signature scheme (G, S, V) is a (t, ϵ)-secure one-time signature scheme if for every algorithm A of complexity at most t, there is probability $\leq \epsilon$ that A, given a public key and one-time access to a signing oracle, produces a valid signature of a message different from the one sent to the signing oracle.*

We describe a scheme due to Leslie Lamport that is based on one-way function.

- KeyGen G: pick a secret key which consists of 2ℓ n-bit random strings

$$x_{0,1},\ x_{0,2}, ...,\ x_{0,\ell}$$

$$x_{1,1},\ x_{1,2}, ...,\ x_{1,\ell}$$

  we now generate a public key by applying f to each $x_{*,*}$ in our secret key:

$$f(x_{0,1}), f(x_{0,2}), ..., f(x_{0,\ell})$$
$$f(x_{1,1}), f(x_{1,2}), ..., f(x_{1,\ell})$$

- Sign S: we sign an ℓ-bit message $M := M_1||M_2||...||M_\ell$ with the signature $\sigma := x_{m_1,1},\ x_{m_2,2}, ...,\ x_{m_\ell,\ell}$

  e.g. the message M:= 0110 gets the signature $\sigma_M := x_{0,1},\ x_{1,2},\ x_{1,3},\ x_{0,4}$

- Verify V: we verify a message $M := M_1...M_\ell$'s signature $\sigma := z_1...z_\ell$ by using public key pk and checking $\forall i\ f(z_i) = pk_{M_i,i}$

Theorem 67 *Let $f : \{0,1\}^n \to \{0,1\}^n$ be a (t, ϵ) one way function computable in time r. Then there is a one-time signature scheme (G, S, V) that signs messages of length ℓ and that is $(t - O(rl), \epsilon \cdot 2\ell)$ secure.*

Suppose this scheme does not provide a (t, ϵ)-one time signature. This implies that there must be an algorithm A with complexity $\leq t$ which makes 1 oracle query and

$$\mathbb{P}[A^{S(sk,\cdot)}(pk) \text{ forges a signature}] > \epsilon$$

Intuitively, when we are given a string $y := f(x)$ we want to use A to break the security of f and determines the pre-image of y.

We now describe the operation A', the algorithm that breaks the security of f. A' begins by generating a public key pk (which requires 2ℓ evaluations of f). Once A' generates pk, it sets a random position in it to the string y. Note the distribution of values in this modified pk will look exactly to the same A because y is also an evaluation of f.

A' now runs A passing it pk, A will query the oracle with a message to sign. With probability $1/2$ this message will not require A' to invert y. If this is the case, then with probability $> \epsilon$ A generates a forged message M' and signature $\sigma_{M'}$. M' must differ from the oracle query by at least one bit, that is this forgery finds the inverse of at least one element of pk not queried. This inverse will be y^{-1} with probability $1/\ell$.

A' runs in time at most $t + O(r\ell)$ if r is the running time of f and and inverts y with probability $\epsilon/2\ell$. Thus if we take f to be (t, ϵ)-secure, then this signature scheme must be $(t - O(rl), \epsilon \cdot 2\ell)$ secure. □

A disadvantage of the scheme (besides the fact of being only a one-time signature scheme) is that the length of the signature and of the keys is much bigger than the length of the

message: a message of length ℓ results in a signature of length $\ell \cdot n$, and the public key itself is of length $2 \cdot \ell \cdot n$.

Using a collision resistant hash function, however, we can convert a one-time signature scheme that works for short messages into a one-time signature scheme that works for longer messages. (Without significantly affecting key length, and without affecting the signature length at all.)

We use the hash function to hash the message into a string of the appropriate length and then sign the hash:

- $G'()$: generates a secret key and a public key (sk, pk) as G did above. Also picks a random seed $d \in \{0,1\}^k$ which becomes part of the public key.
- $S'(sk, M)$: $\sigma := S(sk, H_d(M))$
- $V'((pk, d), M, \sigma)$: $V(pk, H_d(M), \sigma)$

Theorem 68 *Suppose (G, S, V) is a (t, ϵ) secure one-time signature scheme for messages of length ℓ, which has public key length kl and signature length sl. Suppose also that we have a (t, ϵ) secure family of collision-resistant hash functions $H : \{0,1\}^k \times \{0,1\}^m \to \{0,1\}^\ell$. Suppose, finally, that H, G, S all have running time at most r.*

Then there exists a $(t - O(r), 2\epsilon)$ secure one-time signature scheme (G', S', V') with public key length $kl + k$, signature length sl and which can sign messages of length m.

We present a sketch of the proof of this theorem.

Suppose we had an algorithm A that produced forged signatures with probability $> 2\epsilon$ after one oracle query. That is, A queries the oracle with message M and gets back $\sigma := S'(sk, H_d(M))$ and then produces with probability $> 2\epsilon$ a message signature pair, (M', σ'), such that $V(pk, H_d(M'), \sigma') = valid$ and $M \neq M'$.

One of the two following cases must occur with probability $> \epsilon$:

1. The message M' has the same hash as M, $H(M) = H(M')$. This means A was able to find a collision in H. If A does this with probability $> \epsilon$ then it contradicts our security assumptions on H.
2. If $H(M) \neq H(M')$, then A forged a signature for the original scheme (G, S, V) for a fresh message. If A can do this with probability $> \epsilon$ then it contradicts our security assumptions on (G, S, V).

Because we reach a contradiction in both cases, (G', S', V') must be a $(t - O(r), 2\epsilon)$ secure one-time signature scheme.

In particular, given a one-way function and a family of collision-resistant hash functions we can construct a one-time signature scheme in which the length of a signature plus the length of the public key is less than the length of messages that can be signed by the scheme.

If (G, S, V) is such a one-time signature scheme, then the following is a stateful scheme that is existentially unforgeable under a chosen message attack.

Initially, the signing algorithm generates a public key/ secret key pair (pk, sk). When it needs to sign the first message M_1, it creates a new key pair (pk_1, sk_1), and generates the signature $\sigma_0 := S(sk, M_1||pk_1)$. The signature of M_1 is the pair (σ_0, pk_1). When it, later, signs message M_2, the signing algorithm generates a new key pair (pk_2, sk_2), and the signature $\sigma_1 = S(sk_1, M_2||pk_2)$. The signature of M_2 is the sequence

$$M_1, pk_1, \sigma_0, pk_2, \sigma_1$$

and so on. Of course it is rather absurd to have a signature scheme in which the signature of the 100th message contains in its entirety the *previously signed* 100 messages along with their signatures, but this scheme gives an example of the important paradigm of *key refreshing*, which will be more productively employed in the next section.

15.3 From One-Time Signatures to Fully Secure Signatures

Assume we have a (t, ϵ)-secure one-time signature scheme (G, S, V) such that if m is the length of messages that can be signed by S, then the length of public keys generated by $G()$ is at most $m/2$.

(Lamport's signatures do not satisfy the second property, but in Lecture 20 we described how to use a collision-resistant hash function to turn Lamport's scheme into a scheme that can sign longer messages. We can arrange the parameters of the construction so that the hash-and-sign scheme can sign messages at least twice as long as the public key.)

We describe a scheme in which the key generation and signing have exponential complexity; later we will see how to reduce their complexity.

- Key Generation: run $G()$ $2^{m+1} - 1$ times, once for every string $a \in \{0,1\}^*$ of length at most m, and produce a public key / secret key pair (pk_a, sk_a).

  It is convenient to think of the strings a of length at most m as being arranged in a binary tree, with a being the parent of $a0$ and $a1$, and the empty string ϵ being the root.

  - Public Key: pk_ϵ (where ϵ is the empty string)
  - Secret Key: the set of all pairs (pk_a, sk_a) for all a of length $\leq m$.

- Sign: given a message M of length m, denote by $M_{|i}$ the string $M_1, \ldots, M_i$ made of the first i bits of M. Then the signature of M is composed of $m + 1$ parts:

  - $pk_M, S(sk_M, M)$: the signature of M using secret key sk_M, along with the value of the matching public key pk_M

- $pk_{M_{|m-1}}, pk_{M_{|m-1}0}||pk_{M_{|m-1}1}, S(sk_{M_{|m-1}}, pk_{M_{|m-1}0}||pk_{M_{|m-1}1})$ the signature of the public keys corresponding to M and its sibling, signed using the secret key corresponding to the parent of M, along with the public keys
- $\ldots$
- $pk_{M_{|i}}, pk_{M_{|i}0}||pk_{M_{|i}1}, S(sk_{M_{|i}}, pk_{M_{|i}0}||pk_{M_{|i}1})$
- $\ldots$
- $pk_0, pk_1, S(sk_\epsilon, pk_0||pk_1)$

- Verify. The verification algorithm receives a public key pk_ϵ, a message M, and a signature made of $m+1$ pieces: the first piece is of the form (pk_m, σ_m), the following $m-1$ pieces are of the form $(pk_j, pk'_j, pk''_j, \sigma_j)$, for $j = 1, \ldots, m-1$, and the last piece is of the form $(pk'_0, pk''_0, \sigma_0)$.

  The verification algorithm:

  1. checks $V(pk_m, M, \sigma_m)$ is valid;
  2. For $j = 1, \ldots, m$, if $M_j = 0$ it checks that $pk'_{j-1} = pk_j$, and if $M_j = 1$ it checks that $pk''_{j-1} = pk_j$;
  3. For $j = 0, \ldots, m-1$, it checks that $V(pk_j, pk'_j||pk''_j, \sigma_j)$ is valid. (For the case $j = 0$, we take $pk_0 := pk_\epsilon$.)

We visualize the m-bit messages as labels for the leaf nodes of an m level complete binary tree. Each node a of the tree represents a public-secret key pair pk_a, sk_a. The above scheme signs a message M by first using the one-time signature function to sign M using the secret key sk_M at its corresponding leaf node, and releasing the public key pk_M for that node as part of the signature. Now the sender needs to convince the receiver that public key pk_M was really generated by the sender and not a forger. So the sender signs the message consisting of pk_M and its sibling, namely

$$pk_{M_{|m-1}}0||pk_{M_{|m-1}}1 \ ,$$

using the secret key of their parent node $sk_{M_{|m-1}}$, and releases these two public keys and the public key $pk_{M_{|m-1}}$ as part of the message. The sender now has to convince the receiver that $pk_{M_{|m-1}}$ was generated by the sender, and it can apply the previous procedure again to do this. This signing procedure moves up the tree from signing the message at the leaf node to signing messages of two public keys at each level of the tree until it gets to the root node. The root public key pk_ϵ doesn't have to be signed since this is the public key that is released by the sender at the very beginning for all future communication.

Each public-secret key pair node in this tree is used to sign only one message - either the message corresponding to the leaf node if the key is at a leaf node, or the message that is the concatenation of the public keys at its two children. Note that the public key length is $m/2$ and so there are only $2^{m/2}$ distinct public keys in this tree which has $2^{m+1}-1$ nodes. There will certainly be many copies (on average $2^{m/2+1}$) of each public key at different nodes of the tree. We might be concerned that an adversary might then see many signatures for the

same public key and have a much higher chance of breaking the one-time signature scheme for some public key. But if this attack was feasible, then the adversary might as well have generated public-secret key pairs by calling $G()$ and checking if one of these matched some public key seen in the signature of some earlier message - thus, in this scheme, the adversary doesn't get any extra power from seeing multiple signatures using the same key pair.

The theorem below shows that if it is hard for an adversary to forge signatures for the one-time signature scheme (G, S, V), then it will be also be hard to forge signatures under this tree-scheme.

Theorem 69 *Suppose that the scheme described in this section is not (t, ϵ) existentially unforgeable against a chosen message attack.*

Then (G, S, V) is not a $(t \cdot O(r \cdot m), \epsilon/(2tm + 1))$-secure one time signature scheme, where r is the maximum of the running time of S and G.

PROOF: If the tree-scheme is not (t, ϵ) existentially unforgeable against a chosen message attack, then there exists an algorithm A with complexity $\leq t$ such that

$$\mathbb{P}[A^{Sign()}(pk) \text{ forges}] \geq \epsilon$$

A makes $\leq t$ queries to the signing oracle before outputting a fresh message and its forged signature. Hence, A can only see $\leq 2tm + 1$ public keys (and signatures using them) generated by the key generation algorithm G. Using A as a subroutine, we will construct an algorithm A' which given as input a public key pk' of the signature scheme (G, S, V) and one-time access to the signature function $S(sk', \cdot)$ will forge a signature for a fresh message with probability $\geq \epsilon$.

A' picks a random integer i^* in $\{1, ..., 2tm+1\}$ and using the key generation algorithm $G()$, generates $2tm$ key pairs

$$(pk^1, sk^1), ..., (pk^{i^*-1}, sk^{i^*-1}), (pk^{i^*+1}, sk^{i^*+1}), ..., (pk^{2tm+1}, sk^{2tm+1})$$

For notational convenience, set $pk^{i^*} = pk'$.

A' now simulates A on input pk^1. Whenever A makes a call to $Sign()$ with a given message, A' performs the signing algorithm of the tree-scheme by using the public-secret key pairs it randomly generated at the beginning. A' will keep track of which nodes of the tree were already assigned key pairs from it's cache of $2tm + 1$ key pairs. Since at worst $2tm + 1$ key pairs are needed for performing the t signatures requested by A, A' can satisfy all these signature queries using its generated key pairs. If A' needs to sign using $S(sk', \cdot)$, it will use its one-time access to $S(sk', \cdot)$ to perform this action. A' won't have to call $S(sk', \cdot)$ twice with different messages since a public key is never used to sign more than one message in the tree-scheme, unless coincidentally pk' is present as another $pk^j, j \neq i$ in the list of $2tm$ key-pairs generated, in which case A' would have the secret key sk' corresponding to pk' and can completely break (G, S, V). The view of A being run in the simulation by A' is exactly the same as if A had been run on a random public key as input. Hence, the probability A produces a forgery is $\geq \epsilon$.

If A produces a fresh message M and its valid signature

$$\{pk'_{M_{|i}}, pk'_{M_{|i}0}||pk'_{M_{|i}1}, \sigma'_{M_{|i}}\}_{i=0}^{m-1}, pk'_{M_{|m}}, \sigma'_{M_{|m}}$$

then let j be the largest integer such that $pk'_{M_{|j}}$ was seen by A as part of the signature of some message at position $M_{|j}$ in the virtual signature tree. Hence, $pk'_{M_{|j}}$ must be one of the $2tm+1$ keys pk^i generated by A'. Such a value of j must exist because certainly 0 is a candidate for j (since the public key $pk'_\epsilon = pk'_{M_{|0}} = pk^1$ was given as input to A).

Based on the value of j, there are two cases:

- $j \in \{0, ..., m-1\}$. Hence, $pk'_{M_{|j}} = pk^i$ for some i, and if $i = i^*$, then A' will output the message-signature pair $pk'_{M_{|j}0}||pk'_{M_{|j}1}, \sigma'_{M_{|j}}$ as a forgery.
  $V(pk'_{M_{|j}}, pk'_{M_{|j}0}||pk'_{M_{|j}1}, \sigma'_{M_{|j}}) = 1$ because this was part of the valid tree-scheme signature of message M output by A. By the definition of j, A has never seen the signature of $pk'_{M_{|j}0}||pk'_{M_{|j}1}$ before. Since the position i^* was chosen randomly, the event $i = i^*$ has probability $1/(2tm+1)$.
- $j = m$. Here, all the intermediate public keys in the forged signature of M match those seen by A (and hence match the keys generated by A'), but the signature of M at the last level of the tree itself has not been seen. Hence, $pk'_{M|m} = pk^i$ for some i and $V(pk'_{M|m}, M, \sigma'_{M_{|m}}) = 1$ because M is a valid forge produced by A. If $i = i^*$, then A' outputs the forged message-signature pair $M, \sigma'_{M_{|m}}$. Again, since the position i^* was chosen randomly, the event $i = i^*$ has probability $1/(2tm+1)$.

Conditioned on algorithm A outputting a forge to the tree scheme, in both cases algorithm A' produces a forge to the original scheme (G, S, V) with probability $1/(2tm+1)$. Hence, the probability that A' produces a forge to (G, S, V) is $\geq \epsilon/(2tm+1)$. The running time of the simulation A' is dominated by having to generate $2tm$ key pairs and performing m signatures using S for each of the t signing queries made by A, and is $t \cdot O(r \cdot m)$.

□

Lecture 16

Signature Schemes in the Random Oracle Model

Summary

In the last lecture we described a very complex signature scheme based on one-time signatures and pseudorandom functions. Unfortunately there is no known simple and efficient signature scheme which is existentially unforgeable under a chosen message attack under general assumptions.

Today we shall see a very simple scheme based on RSA which is secure in the *random oracle model*. In this model, all parties have oracle access to a random function $H : \{0,1\}^n \to \{0,1\}^m$. In implementations, this random function is replaced by a cryptographic hash function. Unfortunately, the proof of security we shall see today breaks down when the random oracle is replaced by hash function, but at least the security in the random oracle model gives some heuristic confidence in the design soundness of the construction.

16.1 The Hash-and-Sign Scheme

Our starting point is the "textbook RSA" signature scheme, in which a message M is signed as $M^d \bmod N$ and an alleged signature S for a message M is verified by checking that $S^e \bmod N = M$.

We discussed various ways in which this scheme is insecure, including the fact that

1. It is easy to generate random message/ signature pairs M, S by first picking a random S and then setting $M := S^e \bmod N$;
2. If S_1 is the signature of message M_1 and S_2 is the signature of M_2, then $S_1 \cdot S_2 \bmod N$ is the signature of $M_1 \cdot M_2 \bmod N$.

Suppose now that all parties have access to a good cryptographic hash function, which we

will model as a completely random function $H : \{0,1\}^m \to \mathbb{Z}_N$, mapping every possible message M to an random integer $H(M) \in \mathbb{Z}_N$, and define a signature scheme $(Gen, Sign, Verify)$ as follows:

- Key generation: as in RSA
- Signature: the signature of a message M with secret key N, d is $H(M)^d \bmod N$
- Verification: given an alleged signature S, a message M, and a public key N, e, check that $S^e \bmod N = H(M)$.

That is, we use the textbook RSA method to sign $H(M)$.

Now it is not clear any more how to employ the previously mentioned attacks. If we first select a random S, for example, then to find a message of which S is a signature we need to compute $h := S^e \bmod N$ and then find a message M such that $H(M) = h$. This, however, requires exponential time if H is a random functions. Similarly, if we have two messages M_1, M_2 and know their signatures S_1, S_2, the number $S_1 \cdot S_2 \bmod N$ is a signature for any document M such that $H(M) = H(M_1) \cdot H(M_2) \bmod N$. Finding such an M is, however, again very hard.

16.2 Analysis

We provide a formal analysis of the signature scheme defined in the previous section, in the random oracle model.

Theorem 70 *Suppose that $(Gen, Sign, Verify)$, as defined in the previous section, is not (t, ϵ) existentially unforgeable under a chosen message attack in the random oracle model.*

Then RSA, with the key size used in the construction, is not a $(t \cdot O(r), \epsilon \cdot \frac{1}{t})$-secure family of trapdoor permutations, where r is the time taken by RSA computation with the selected key size.

PROOF: We will prove that, if A is an algorithm of complexity at most t that breaks existential unforgeability under chosen message attack with probability $\geq \epsilon$, then there is an algorithm A' that breaks RSA (finds X given $X^e \bmod N$) with probability $\geq \frac{\epsilon}{t}$ and complexity $\leq t \cdot O(r)$.

$$Pr[A^{H, Sign(N,d,\cdot)}(N, e) = (M, S) : (H(M)) = S^e] \geq \epsilon$$

Without the loss of generality we assume that:

- A never makes the same random oracle query twice.
- A queries $H(M)$ before it requests a signature on a message M.

- If A outputs (M, S) then it had previously queried $H(M)$

We construct an algorithm A' which on input (N, e, y) where $y = X^e \bmod N$, finds X.

Algorithm A' is defined as:

- Pick $i \leftarrow \{1, ..., t\}$ randomly.
- Initialise datastructure that stores triples, initially empty.
- Simulate A:
  - When A makes its jth random oracle query $H(M_j)$
    - If $j = i$, answer the oracle query with y.
    - Otherwise, randomly pick X_j, compute ${X_j}^e \bmod N$, store $(M_j, X_j, {X_j}^e \bmod N)$ in the datastructure and answer the oracle query with $y_j = {X_j}^e \bmod N$
  - When A requests $Sign(M_k)$
    - If $k = i$ abort.
    - If $k \neq i$ look for $(M_k, X_k, {X_k}^e \bmod N)$ in the datastructure and answer the oracle query with X_k.
      (Note that we had made the assumption that A queries $H(M)$ before it requests a signature on a message M.)
- After A finishes, it outputs (M, S). If $M = M_i$ and $S^e = y \bmod N$, then output S as the required output X.

For each random oracle query, we are doing a RSA exponentiation operation of complexity r. So the complexity of A' would be at most complexity of A multiplied by $O(r)$ i.e. $t \cdot O(r)$.

The index i chosen by A' in the first step represents a guess as to which oracle query of A will correspond to the eventual forgery output by A. When the guess is correct, view of A as part of A' is distributed identically to the view of A alone. When A' guesses correctly and A outputs a forgery, then A' solves the given instance of RSA problem (because $S^e = y \bmod N$ and thus S is *inverse* of y). Since A' guesses correctly with probability $1/t$ and A outputs a forgery with probability $\geq \epsilon$. So, Probability with which A' breaks RSA $\geq \epsilon \cdot \frac{1}{t}$, which is what we intended to prove. $\square$

Lecture 17

CCA-secure Encryption in the Random Oracle Model

Summary

Today we show how to construct an efficient CCA-secure public-key encryption scheme in the *random oracle model* using RSA.

As we discussed in the previous lecture, a cryptographic scheme defined in the *random oracle model* is allowed to use a random function $H : \{0,1\}^n \to \{0,1\}^m$ which is known to all the parties. In an implementation, usually a cryptographic hash function replaces the random oracle. In general, the fact that a scheme is proved secure in the random oracle model does not imply that it is secure when the random oracle is replaced by a hash function; the proof of security in the random oracle model gives, however, at least some heuristic confidence in the soundness of the design.

17.1 Hybrid Encryption with a Random Oracle

We describe a public-key encryption scheme $(\overline{G}, \overline{E}, \overline{D})$ which is based on: (1) a family of trapdoor permutations (for concreteness, we shall refer specifically to RSA below); (2) a CCA-secure *private-key* encryption scheme (E, D); (3) a random oracle H mapping elements in the domain and range of the trapdoor permutation into keys for the private-key encryption scheme (E, D).

1. Key generation: $\overline{G}$ picks an RSA public-key / private-key pair $(N, e), (N, d)$;
2. Encryption: given a public key N, e and a plaintext M, $\overline{E}$ picks a random $R \in \mathbb{Z}_N^*$, and outputs
$$(R^e \bmod N, E(H(R), M))$$
3. Decryption: given a private key N, d and a ciphertext C_1, C_2, $\overline{D}$ decrypts the plaintext by computing $R := C_1^d \bmod N$ and $M := D(H(R), C_2)$.

This is a hybrid encryption scheme in which RSA is used to encrypt a "session key" which is then used to encrypt the plaintext via a private-key scheme. The important difference from hybrid schemes we discussed before is that the random string encrypted with RSA is "hashed" with the random oracle before being used as a session key.

17.2 Security Analysis

The intuition behind the security of this scheme is as follows. For any adversary mounting a CCA attack on $(\overline{G}, \overline{E}, \overline{D})$, we distinguish between the case that it does not query R to the random oracle H and the case that it does. In the first case, the adversary learns nothing about $H(R)$, and so we reduce the CCA security of $(\overline{G}, \overline{E}, \overline{D})$ to the CCA security of the private-key encryption scheme (E, D). In the second case, the adversary has managed to invert RSA. Under the RSA trapdoor permutation assumption, this case can only happen with negligible probability.

This intuition is formalized in the proof of the following theorem.

Theorem 71 *Suppose that, for the key size used in $(\overline{G}, \overline{E}, \overline{D})$, RSA is a (t, ϵ)-secure family of trapdoor permutations, and that exponentiation can be computed in time $\leq r$; assume also that (E, D) is a (t, ϵ) CCA-secure private-key encryption scheme and that E, D can be computed in time $\leq r$.*

Then $(\overline{G}, \overline{E}, \overline{D})$ is $(t/O(r), 2\epsilon)$ CCA-secure in the random oracle model.

PROOF: Suppose A is a CCA attack on $(\overline{G}, \overline{E}, \overline{D})$ of complexity $t' \leq t/O(r)$ such that there are two messages M_0, M_1

$$|\,\mathbb{P}[A^{\overline{E},\overline{D},H}(\overline{E}(M_0)) = 1] - \mathbb{P}[A^{\overline{E},\overline{D},H}(\overline{E}(M_1)) = 1]| > 2\epsilon.$$

We will derive from A an algorithm A' running in time $O(t' \cdot r)$ which is a CCA attack on (E, D). If A' succeeds with probability at least ϵ in distinguishing between $E(M_0)$ and $E(M_1)$, then we have violated the assumption on the security of (E, D). But if A' succeeds with probability less than ϵ, then we can devise an algorithm A'', also of running time $O(t' \cdot r)$, which inverts RSA with probability at least ϵ.

The algorithm A' is designed to attack the private-key encryption scheme (E, D) by running A as a subroutine. Its idea is to convert the ciphertext $E(M)$ into the ciphertext $\overline{E}(M)$ and then use A to distinguish it. It needs to answer A's oracle queries to $\overline{E}, \overline{D}, H$, by accessing its own oracles E, D. In particular, in order to make its answers to the random oracle queries consistent, A' uses a data structure (a table) to remember the pairs $(R', H(R'))$ that have been queried so far. The formal definition of A' is as follows:

Algorithm $A'^{E,D}$:
Input: $C = E(K, M)$ //K is unknown, $M \in \{M_0, M_1\}$

- pick RSA keys N, e, d.

- pick a random $R \in \mathbb{Z}_N^*$.
- pairs of strings $(\cdot, \cdot)$ are stored in a table which is initially empty.
- simulate $A^{\overline{E},\overline{D},H}(R^e \bmod N, C)$ as follows:
  - when A queries $H(R')$:
    - If there is an entry (R', K') in the table, return K' to A;
    - otherwise, pick a random K', store (R', K') in the table, return K' to A.
    - If $R' = R$, FAIL.
  - when A queries $\overline{E}(M')$:
    - pick a random $R' \in \mathbb{Z}_N^*$, compute $K' := H(R')$ as above, return $(R'^e \bmod N, E(K', M'))$ to A.
  - when A queries $\overline{D}(C_1, C_2)$:
    - compute $R' := C_1^d \bmod N$.
    - if $R' \neq R$, compute $K' := H(R')$ as above, return $D(K', C_2)$ to A;
    - if $R' = R$, query C_2 to the decryption oracle D and return the result to A.

A' needs time $O(r)$ to compute the answer of every oracle query from A. Given that A has complexity $t' \leq t/O(r)$, we get A' has complexity $O(t' \cdot r) \leq t$. Furthermore, A' never submits its challenge ciphertext C to its own decryption oracle D, because the only way this could happen would be if A submitted its challenge ciphertext $(R^e \bmod N, C)$ to its own decryption oracle $\overline{D}$, but this is not allowed.

Let $QUERY$ denote the event that A queries R to the random oracle H during its execution. Note that in A''s strategy of answering A's decryption queries, A' has implicitly set $H(R) = K$. If $QUERY$ happens, i.e. A queries $H(R)$, A' returns a random $\hat{K}$ to A in response to this query, but $\hat{K}$ might not be equal to the unknown K, and this will cause an inconsistency in A''s answers. However, if $QUERY$ does not happen, the answers A' gives to A in response to its oracle queries to $\overline{E}, \overline{D}, H$ are always consistent and distributed identically to the answers obtained from the corresponding real oracles. So in this case, the behavior of A remains unchanged. Thus we get

$$\mathbb{P}[A'^{E,D}(E(M)) = 1] = \mathbb{P}[A^{\overline{E},\overline{D},H}(\overline{E}(M)) = 1 \wedge \neg QUERY],$$

and hence

$$\begin{aligned} & |\,\mathbb{P}[A'^{E,D}(E(M_0)) = 1] - \mathbb{P}[A'^{E,D}(E(M_1)) = 1]| \\ = \;& |\,\mathbb{P}[A^{\overline{E},\overline{D},H}(\overline{E}(M_0)) = 1 \wedge \neg QUERY] \\ & - \mathbb{P}[A^{\overline{E},\overline{D},H}(\overline{E}(M_1)) = 1 \wedge \neg QUERY]|. \end{aligned}$$

Then by the triangle inequality we obtain

$$\begin{aligned}
2\epsilon &< |\,\mathbb{P}[A^{\overline{E},\overline{D},H}(\overline{E}(M_0))=1] - \mathbb{P}[A^{\overline{E},\overline{D},H}(\overline{E}(M_1))=1]| \\
&\leq |\,\mathbb{P}[A^{\overline{E},\overline{D},H}(\overline{E}(M_0))=1 \wedge QUERY] \\
&\quad - \mathbb{P}[A^{\overline{E},\overline{D},H}(\overline{E}(M_1))=1 \wedge QUERY]| \\
&\quad + |\,\mathbb{P}[A^{\overline{E},\overline{D},H}(\overline{E}(M_0))=1 \wedge \neg QUERY] \\
&\quad - \mathbb{P}[A^{\overline{E},\overline{D},H}(\overline{E}(M_1))=1 \wedge \neg QUERY]| \\
&\leq \max_{M\in\{M_0,M_1\}} \mathbb{P}[A^{\overline{E},\overline{D},H}(\overline{E}(M))=1 \wedge QUERY] \qquad (1) \\
&\quad + |\,\mathbb{P}[A'^{E,D}(E(M_0))=1] - \mathbb{P}[A'^{E,D}(E(M_1))=1]| \qquad (2)
\end{aligned}$$

So at least one of (1) and (2) must be greater than ϵ.

If (2) $> \epsilon$, then A' breaks the CCA-security of (E, D), contradicting our assumption. So we should have (1) $> \epsilon$. But in this case, we can devise an algorithm A'' of complexity $O(t' \cdot r)$ such that A'' solves RSA with probability $> \epsilon$.

The idea of A'' is also to simulate A by providing it with appropriate input and oracle query answers. Similarly, A'' also needs a data structure (a table) to record the answers $(R', H(R'))$ it has given so far. However, now A'' only knows the public RSA keys N, e, but does not know the private key d. Consequently, when decrypting a ciphertext (C_1, C_2), it cannot compute $R' := C_1^d \bmod N$ (without factoring N). Then how can we get $H(R')$ without knowing R'? In order to overcome this obstacle, we store triples of strings $(R', R'^e \bmod N, H(R'))$, instead of pairs $(R', H(R'))$, in the table. If R' is unknown yet, we use a special symbol $\perp$ to represent it. Now we no longer check whether the first item of each entry in the table is equal to R', but check whether the second item of each entry is equal to $R'^e \bmod N$. Because there is a one-to-one correspondence between R' and $R'^e \bmod N$ (given N, e), by checking the second item, we can get $H(R')$ without knowing R'.

The formal definition of A'' is as follows:

Algorithm A'':
Input: N,e, $C = R^e \bmod N$ // R is unknown

- pick a random K.
- triples of strings $(\cdot, \cdot, \cdot)$ are stored in a table which initially contains only $(\perp, C, K)$.
- simulate $A^{\overline{E},\overline{D},H}(C, E(K, M))$ as follows: //M is the message for which (1)$> \epsilon$
  - when A queries $H(R')$:
    - if $R'^e \bmod N == C$, output R', halt;
    - if there is an entry $(R', R'^e \bmod N, K')$ in the table, return K' to A;
    - otherwise, if there is an entry $(\perp, R'^e \bmod N, K')$ in the table, replace it with $(R', R'^e \bmod N, K')$, return K' to A;

    - otherwise, pick a random K', store $(R', R'^e \bmod N, K')$ in the table, return K' to A.
- when A queries $\overline{E}(M')$:
    - pick a random $R' \in \mathbb{Z}_N^*$, compute $K' := H(R')$ as above, return $(R'^e \bmod N, E(K', M'))$ to A.
- when A queries $\overline{D}(C_1, C_2)$:
    - if there is an entry (R', C_1, K') or $(\perp, C_1, K')$ in the table, return $D(K', C_2)$ to A;
    - otherwise, pick a random K', store $(\perp, C_1, K')$ in the table, return $D(K', C_2)$ to A.

A'' also needs time $O(r)$ to compute the answer of every oracle query from A. Given that A has complexity $t' \leq t/O(r)$, we get A'' has complexity $O(t' \cdot r) \leq t$. Moreover, the answers A'' gives to A in response to its oracle queries to $\overline{E}, \overline{D}, H$ are always consistent and distributed identically to the answers obtained from the corresponding real oracles. Thus, the behavior of A remains unchanged. Especially, the event $QUERY$ remains unchanged. Furthermore, A'' correctly solves the given RSA instance whenever $QUERY$ occurs. So we have

$$\mathbb{P}[A''(N, e, R^e \bmod N) = R] = \mathbb{P}[QUERY] \geq (1) > \epsilon,$$

which contradicts with the assumption that RSA is (t, ϵ)-secure. This concludes the proof of the theorem. $\square$

Lecture 18

Zero Knowledge Proofs

Summary

Today we introduce the notion of *zero knowledge proof* and design a zero knowledge protocol for the *graph isomorphism* problem.

18.1 Intuition

A *zero knowledge proof* is an interactive protocol between two parties, a *prover* and a *verifier*. Both parties have in input a *statement* that may or may not be true, for example, the description of a graph G and the statement that G is 3-colorable, or integers N, r and the statement that there is an integer x such that $x^2 \bmod N = r$. The goal of the prover is to *convince* the verifier that the statement is true, and, at the same time, make sure that *no information other than the truth of the statement* is leaked through the protocol.

A related concept, from the computational viewpoint, is that of a *zero knowledge proof of knowledge*, in which the two parties share an input to an NP-type problem, and the prover wants to convince the verifier that he, the prover, *knows a valid solution for the problem on that input*, while again making sure that no information leaks. For example, the common input may be a graph G, and the prover may want to prove that he knows a valid 3-coloring of G, or the common input may be N, r and the prover may want to prove that he knows an x such that $x^2 \bmod N = r$.

If a prover "proves knowledge" of a 3-coloring of a graph G, then he also proves the statement that G is 3-coloring; in general, a proof of knowledge is also a proof of the statement that the given instance admits a witness. In some cases, however, proving that an NP statement is true, and hence proving *existence* of a witness, does not imply a proof of *knowledge* of the witness. Consider, for example, the case in which common input is an integer N, and the prover wants to prove that he knows a non-trivial factor N. (Here the corresponding "statement" would be that N is composite, but this can easily be checked by the verifier offline, without the need for an interaction.)

Identification schemes are a natural application of zero knowledge. Suppose that a user wants to log in into a server. In a typical Unix setup, the user has a password x, and the server keeps a hash $f(x)$ of the user's password. In order to log in, the user sends the password x to the server, which is insecure because an eavesdropper can learn x and later impersonate the user.

In a secure identification scheme, instead, the user generates a public-key/ secret key pair (pk, sk), the server knows only the public key pk, and the user "convinces" the server of his identity without revealing the secret key. (In SSH, for example, (pk, sk) are the public key/ secret key pair of a signature scheme, and the user signs a message containing a random session identifier in order to "convince" the server.)

If f is a one-way function, then a secure identification scheme could work as follows: the user picks a random secret key x and lets its public key be $f(x)$. To prove its identity, the user engages in a *zero knowledge proof of knowledge* with the server, in which the user plays the prover, the server plays the verifier, and the protocol establishes that the user knows an inverse of $f(x)$. Hence, the server would be convinced that only the actual person would be able to log in, and moreover from the point of view of the user he/she will not be giving away any information the server might maliciously utilize after the authentication.

This example is important to keep in mind as every feature in the definition of the protocol has something desirable in the protocol of this model.

The main application of zero knowledge proofs is in the theory of multi party protocols in which multiple parties want to compute a function that satisfies certain security and privacy property. One such example would be a protocol that allow several players to play online poker with no trusted server. By such a protocol, players exchange messages to get the local view of the game and also at the end of the game to be able to know what is the final view of the game. We would like that this protocol stays secure even in the presence of malicious players. One approach to construct such a secure protocol is to first come up with a protocol that is secure against "honest but curious" players. According to this relaxed notion of security, nobody gains extra information provided that everybody follows the protocol. Then one provides a generic transformation from security against "honest but curious" to security against malicious user. This is achieved by each user providing a ZKP at each round that in the previous round he/she followed the protocol. This would on one side convince the other players that no one is cheating and on the other side the player presenting the protocol would provide no information about his own cards. This forces apparent malicious players to act honestly, as only they can do is to analyze their own data. At at the same time this is also not a problem for the honest players.

18.2 The Graph Non-Isomorphism Protocol

We say that two graphs $G_1 = (V, E_1)$ and $G_2 = (V, E_2)$ are *isomorphic* if there is a bijective relabeling $\pi : V \to V$ of the vertices such that the relabeling of G_1 is the same graph as G_2, that is, if

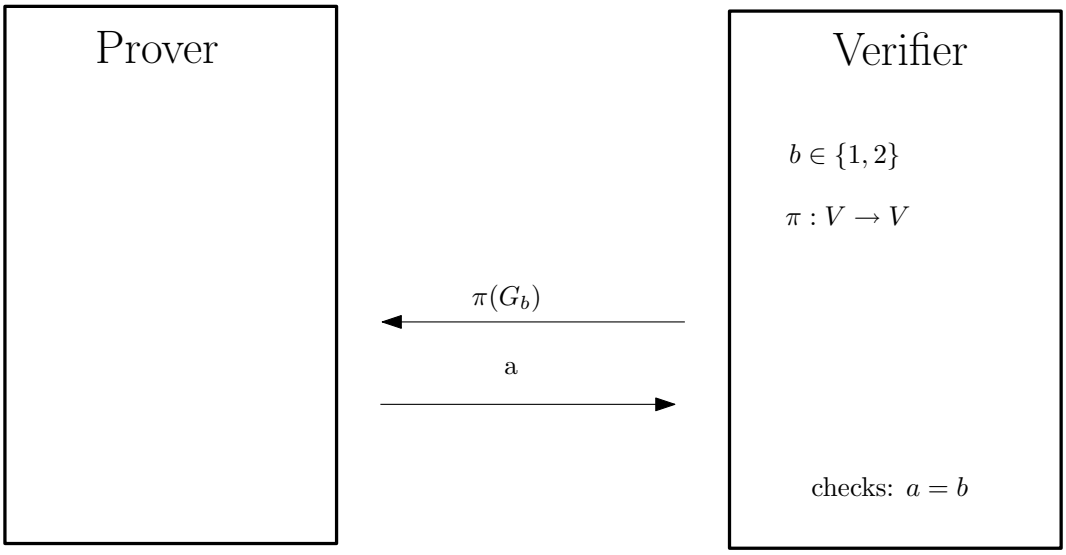
$$(u, v) \in E_1 \Leftrightarrow (\pi(u), \pi(v)) \in E_2$$

We call $\pi(G_1)$ the graph that as an edge $(\pi(u), \pi(v))$ for every edge (u, v) of E_1.

The graph isomorphism problem is, given two graphs, to check if they are isomorphic.

It is believed that this problem is not NP-complete however algorithm that would run faster than $\mathcal{O}\left(2^{\sqrt{N}}\right)$ is not known.

Here we describe an interactive protocol in which a prover can "convince" a verifier that two given graphs are not isomorphic, and in which the verifier only makes questions for which he already knows an answer, so that, intuitively, he gains no new knowledge from the interaction. (We will give a precise definition later, but we will not prove anything formal about this protocol, which is presented only for intuition.) For the prover, unfortunately, we only know how to provide an exponential time implementation. However the verifier algorithm, is very efficient.



- Common input: two graphs $G_1 = (V, E_1)$, $G_2 = (V, E_2)$; the prover wants to convince the verifier that they are *not* isomorphic
- The verifier picks a random $b \in \{1, 2\}$ and a permutation $\pi : V \to V$ and sends $G = \pi(G_b)$ to the prover
- The prover finds the bit $a \in \{1, 2\}$ such that G_a and G are isomorphic sends a to the verifier
- The verifier checks that $a = b$, and, if so, accepts

Theorem 72 *Let P be the prover algorithm and V be the verifier algorithm in the above protocol. Then*

1. *If G_1, G_2 are not isomorphic, then the interaction $P(x) \leftrightarrow V(x)$ ends with the verifier accepting with probability 1*

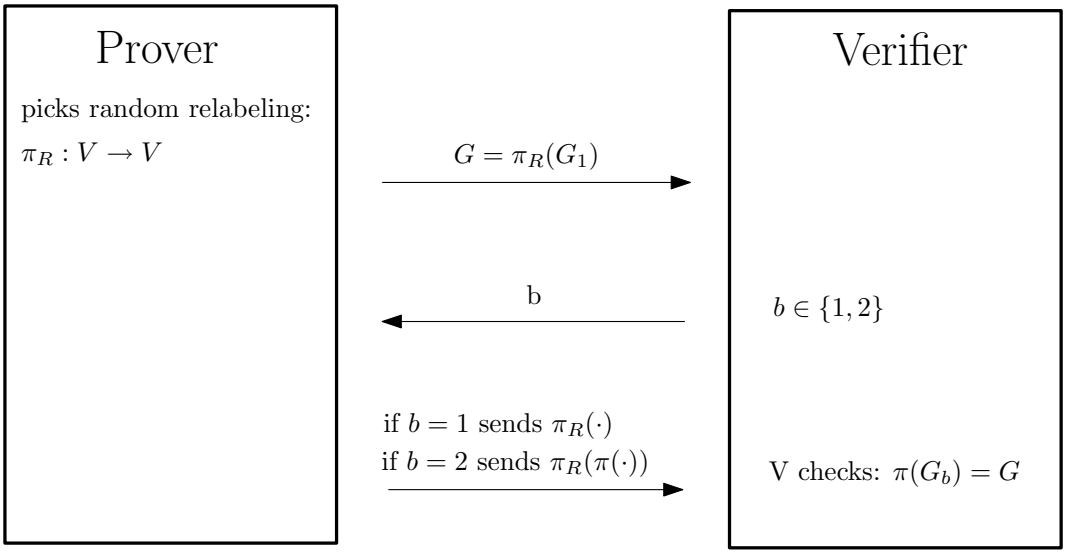
2. *If G_1, G_2 are isomorphic, then for every alternative prover strategy P^*, of arbitrary complexity, the interaction $P^*(x) \leftrightarrow V(x)$ ends with the verifier accepting with probability $1/2$*

The first part of the theorem is true as for every permutation $\pi(G_1)$ is not isomorphic to G_2 and similarly for every permutation $\pi(G_2)$ is not isomorphic to G_2, therefore if G_1 and G_2 are not isomorphic no relabeling of G_1 can make it isomorphic to G_2. Since the prover runs in exponential time he can always find out which graph the verifier has started from and therefore the prover always gives the right answer.

The second part of the theorem is true as there exist permutation π^* such that $\pi^*(G_2) = G_1$. Then if verifier picks a random permutation π_R then the distribution we obtain by $\pi_R(\pi^*(G_2))$ and the distribution $\pi_R(G_1)$ are exactly the same as both are just random relabelling of, say, G_1. This fact is analogous to the fact that if we add a random element from the group to some other group element we get again the random element of the group. Therefore here the answer of the prover is independent on b and the prover succeeds with probability half. This probability of $\frac{1}{2}$ can be reduced to 2^{-k} by repeating the protocol k times. It is important to notice that this protocol is ZK since the verifier already knows the answer so he learns nothing at the end of interaction. The reason why the verifier is convinced is because the prover would need to do something that is information theoretically impossible if the graphs are isomorphic. Therefore, it is not the answers themselves that convince the prover but the fact that prover can give those answers without knowing the isomorphism.

18.3 The Graph Isomorphism Protocol

Suppose now that the prover wants to prove that two given graphs G_1, G_2 are isomorphic, and that he, in fact, knows an isomorphism. We shall present a protocol for this problem in which both the prover and the verifier are efficient.



- Verifier's input: two graphs $G_1 = (V, E_1)$, $G_2 = (V, E_2)$;
- Prover's input: G_1, G_2 and permutation π^* such that $\pi^*(G_1) = G_2$; the prover wants to convince the verifier that the graphs are isomorphic
- The prover picks a random permutation $\pi_R : V \to V$ and sends the graph $G := \pi_R(G_2)$
- The verifier picks at random $b \in \{1, 2\}$ and sends b to the prover
- The prover sends back π_R if $b = 1$, and $\pi_R(\pi^*(\cdot))$ otherwise
- The verifier checks that the permutation π received at the previous round is such that $\pi(G_b) = G$, and accepts if so

Theorem 73 *Let P be the prover algorithm and V be the verifier algorithm in the above protocol. Then*

1. *If G_1, G_2 are isomorphic, then the interaction $P(x) \leftrightarrow V(x)$ ends with the verifier accepting with probability 1*
2. *If G_1, G_2 are not isomorphic, then for every alternative prover strategy P^*, of arbitrary complexity, the interaction $P^*(x) \leftrightarrow V(x)$ ends with the verifier accepting with probability $1/2$*

The first part is clear from the construction.

What happens if G_1 and G_2 are not isomorphic and the prover is not following the protocol and is trying to cheat a verifier? Since in the first round the prover sends a graph G, and G_1 and G_2 are not isomorphic, then G can not be isomorphic to both G_1 and G_2. So in second round with probability at least half the verifier is going to pick G_b that is not isomorphic to G. When this happens there is nothing that the prover can send in the third round to

make the verifier accept, since the verifier accepts only if what prover sends in the third round is the isomorphism between G and G_b. Hence the prover will fail with probability a half at each round and if we do the same protocol for several rounds the prover will be able to cheat only with exponentially small probability.

Definition 74 *A protocol defined by two algorithms P and V is an interactive proof with efficient prover, for a decision problem if:*

- ***(Completeness)*** *for every input x for which the correct answer is YES, there is a witness w such that $P(x,w) \leftrightarrows V(x)$ interaction ends with V accepting with probability one.*
- ***(Soundness)*** *for every input x for which answer is NO, for algorithm P^* of arbitrary complexity $P^*(x,w) \leftrightarrows V(x)$ interaction ends with V rejecting with probability at least half (or at least $1 - \frac{1}{2^k}$ if protocol repeated k times)*

So the graph isomorphism protocol described above is an *interactive proof with efficient prover* for the graph isomorphism protocol.

We now formalize what we mean by the verifier *gaining zero knowledge* by participating in the protocol. The interaction is ZK if the verifier could simulate the whole interaction by himself without talking to the prover.

Definition 75 (Honest Verifier Perfect Zero Knowledge) *A protocol (P,V) is Honest Verifier Perfect Zero Knowledge with simulation complexity s for a decision problem if there is an algorithm $S(\cdot)$ that has complexity at most s, such that $\forall x$ for which the answer is YES, $S(x)$ samples the distribution of $P(x,w) \leftrightarrows V(x)$ interactions for every valid w.*

Therefore the simulator does not know the witness but it is able to replicate the interaction between the prover and the verifier. One consequence of this is, the protocol is able to simulate all possible interactions regardless of what particular witness the prover is using. Hence the protocol does the same regardless of witness. This *witness indistinguishability* property is useful on its own.

Now consider an application in which the user is the prover and the server is the verifier. For this application of ZKP it is not sufficient that honest V does not learn anything following the protocol but also that if the verifier is not honest and does not follow the protocol he will still not be able learn anything from the prover.

Therefore the full definition of zero knowledge is the following.

Definition 76 (Perfect Zero Knowledge) *A prover algorithm P is (general) Perfect Zero Knowledge with simulation overhead $so(\cdot)$ for a decision problem if*

- *for every algorithm V' of complexity at most t there is a simulator algorithm S' of complexity at most $so(t)$,*

- *such that for every x for which the answer is YES, and every valid witness w, $S'(x)$ samples $P(x,w) \leftrightarrows V'(x)$.*

(In an asymptotic setting, we would want $so(t)$ to be polynomial in t. Typically, we have $so(t) \leq O(t) + n^{O(1)}$.)

So from the prover's viewpoint the protocol is always safe, since even if the verifier does not follow the protocol he would be able to gain only the information that he (the verifier) would gain otherwise anyway, by running the simulator himself.

The zero knowledge property is purely the property of the prover algorithm since it is quantified of over all witnesses, all inputs, and all verifier algorithms. Symmetrically the soundness property was the property of the verifier algorithm since the verifier would get convinced with high probability only if the property is really true, regardless whether the prover is malicious or not.

18.4 A Simulator for the Graph Isomorphism Protocol

In order to prove that the protocol of Section 18.3 is zero knowledge, we have to show the existence of an efficient simulator.

Theorem 77 (Honest-Verifier Zero Knowledge) *There exists an efficient simulator algorithm S^* such that, for every two isomorphic graphs G_1, G_2, and for every isomorphism π between them, the distributions of transcripts*

$$P(\pi, G_1, G_2) \leftrightarrow Ver(G_1, G_2) \tag{18.1}$$

and

$$S(G_1, G_2) \tag{18.2}$$

are identical, where P is the prover algorithm and Ver is the verifier algorithm in the above protocol.

Proof:

Algorithm S on input G_1, G_2 is described as follows:

- Input: graphs G_1, G_2
- pick uniformly at random $b \in \{1,2\}$, $\pi_R : V \to V$
- output the transcript:
  1. prover sends $G = \pi_R(G_b)$
  2. verifier sends b
  3. prover sends π_R

At the first step, in the original protocol we have a random permutation of G_2, while in the simulation we have either a random permutation of G_1 or a random permutation of G_2; a random permutation of G_1, however, is distributed as $\pi_R(\pi^*(G_2))$, where π_R is uniformly distributed and π^* is fixed. This is the same as a random permutation of G_2, because composing a fixed permutation with a random permutation produces a random permutation.

The second step, both in the simulation and in the original protocol, is a random bit b, selected independently of the graph G sent in the first round. This is true in the simulation too, because the distribution of $G := \pi_R(G_b)$ conditioned on $b = 1$ is, by the above reasoning, identical to the distribution of G conditioned on $b = 0$.

Finally, the third step is, both in the protocol and in the simulation, a distribution uniformly distributed among those establishing an isomorphism between G and G_b. □

To establish that the protocol satisfies the general zero knowledge protocol, we need to be able to simulate cheating verifiers as well.

Theorem 78 (General Zero Knowledge) *For every verifier algorithm V^* of complexity t there is a simulator algorithm S^* of expected complexity $\leq 2t + O(n^2)$ such that, for every two isomorphic graphs G_1, G_2, and for every isomorphism π between them, the distributions of transcripts*

$$P(\pi, G_1, G_2) \leftrightarrow V^*(G_1, G_2) \tag{18.3}$$

and

$$S^*(G_1, G_2) \tag{18.4}$$

are identical.

Proof:

Algorithm S^* on input G_1, G_2 is described as follows:

Input G_1, G_2

1. pick uniformly at random $b \in \{1, 2\}$, $\pi_R : V \to V$

- $G := \pi_R(G_b)$
- let b' be the second-round message of V^* given input G_1, G_2, first message G
- if $b \neq b'$, abort the simulation and go to 1.
- else output the transcript
  - prover sends G
  - verifier sends b
  - prover sends π_R

As in the proof of Theorem 77, G has the same distribution in the protocol and in the simulation.

The important observation is that b' depends only on G and on the input graphs, and hence is statistically independent of b. Hence, $\mathbb{P}[b = b'] = \frac{1}{2}$ and so, on average, we only need two attempts to generate a transcript (taking overall average time at most $2t + O(n^2)$). Finally, conditioned on outputting a transcript, G is distributed equally in the protocol and in the simulation, b is the answer of V^*, and π_R at the last round is uniformly distributed among permutations establishing an isomorphism between G and G_b. $\square$

Lecture 19

Zero Knowledge Proofs of Quadratic Residuosity

Summary

Today we discuss the *quadratic residuosity problem* modulo a composite, and define a protocol for proving quadratic residuosity.

19.1 The Quadratic Residuosity Problem

We review some basic facts about quadratic residuosity modulo a composite.

If $N = p \cdot q$ is the product of two distinct odd primes, and $\mathbb{Z}_N^*$ is the set of all numbers in $\{1, \ldots, N-1\}$ having no common factor with N, then we have the following easy consequences of the Chinese remainder theorem:

- $\mathbb{Z}_N^*$ has $(p-1) \cdot (q-1)$ elements, and is a group with respect to multiplication;

  PROOF:

  Consider the mapping $x \to (x \bmod p, x \bmod q)$; it is a bijection because of the Chinese remainder theorem. (We will abuse notation and write $x = (x \bmod p, x \bmod q)$.) The elements of $\mathbb{Z}_N^*$ are precisely those which are mapped into pairs (a, b) such that $a \neq 0$ and $b \neq 0$, so there are precisely $(p-1) \cdot (q-1)$ elements in $\mathbb{Z}_N^*$.

  If $x = (x_p, x_q)$, $y = (y_p, y_q)$, and $z = (x_p \times y_p \bmod p, x_q \times y_q \bmod q)$, then $z = x \times y \bmod N$; note that if $x, y \in \mathbb{Z}_N^*$ then x_p, y_p, x_q, y_q are all non-zero, and so $z \bmod p$ and $z \bmod q$ are both non-zero and $z \in \mathbb{Z}_N^*$.

  If we consider any $x \in \mathbb{Z}_N^*$ and we denote $x' = (x_p^{-1} \bmod p, x_q^{-1} \bmod q)$, then $x \cdot x' \bmod N = (x_p x_p^{-1}, x_q x_q^{-1}) = (1, 1) = 1$.

  Therefore, $\mathbb{Z}_N^*$ is a group with respect to multiplication. $\square$

- If $r = x^2 \bmod N$ is a quadratic residue, and is an element of $\mathbb{Z}_N^*$, then it has exactly 4 square roots in $\mathbb{Z}_N^*$

  PROOF:

  If $r = x^2 \bmod N$ is a quadratic residue, and is an element of $\mathbb{Z}_N^*$, then:

  $r \equiv x^2 \bmod p$

  $r \equiv x^2 \bmod q$.

  Define $x_p = x \bmod p$ and $x_q = x \bmod q$ and consider the following four numbers:

  $x = x_1 = (x_p, x_q)$

  $x_2 = (-x_p, x_q)$

  $x_3 = (x_p, -x_q)$

  $x_4 = (-x_p, -x_q)$.

  $x^2 \equiv x_1^2 \equiv x_2^2 \equiv x_3^2 \equiv x_4^2 \equiv r \bmod N$.

  Therefore, x_1, x_2, x_3, x_4 are distinct square roots of r, so r has 4 square roots.

  □

- Precisely $(p-1)\cdot(q-1)/4$ elements of $\mathbb{Z}_N^*$ are quadratic residues

  PROOF:

  According to the previous results, $\mathbb{Z}_N^*$ has $(p-1)\cdot(q-1)$ elements, and each quadratic residue in $\mathbb{Z}_N^*$ has exactly 4 square roots. Therefore, $(p-1)\cdot(q-1)/4$ elements of $\mathbb{Z}_N^*$ are quadratic residues. □

- Knowing the factorization of N, there is an efficient algorithm to check if a given $y \in \mathbb{Z}_N^*$ is a quadratic residue and, if so, to find a square root.

It is, however, believed to be hard to find square roots and to check residuosity modulo N if the factorization of N is not known.

Indeed, we can show that from any algorithm that is able to find square roots efficiently mod N we can derive an algorithm that factors N efficiently.

Theorem 79 *If there exists an algorithm A of running time t that finds quadratic residues modulo $N = p \cdot q$ with probability $\geq \epsilon$, then there exists an algorithm A^* of running time $t + O(\log N)^{O(1)}$ that factors N with probability $\geq \frac{\epsilon}{2}$.*

PROOF: Suppose that, for a quadratic residue $r \in \mathbb{Z}_N^*$, we can find two square roots x, y such that $x \neq \pm y \pmod N$. Then $x^2 \equiv y^2 \equiv r \bmod N$, then $x^2 - y^2 \equiv 0 \bmod N$. Therefore, $(x-y)(x+y) \equiv 0 \bmod N$. So either $(x-y)$ or $(x+y)$ contains p as a factor, the other contains q as a factor.

The algorithm A^* is described as follows:

Given $N = p \times q$

- pick $x \in \{0 \ldots N-1\}$
- if x has common factors with N, return $\gcd(N, x)$
- if $x \in \mathbb{Z}_N^*$
  - $r := x^2 \bmod N$
  - $y := A(N, r)$
  - if $y \neq \pm x \bmod N$ return $gcd(N, x+y)$

WIth probability ϵ over the choice of r, the algorithm finds a square root of r. Now the behavior of the algorithm is independent of how we selected r, that is which of the four square roots of r we selected as our x. Hence, there is probability $1/2$ that, conditioned on the algorithm finding a square root of r, the square root y satisfies $x \neq \pm y \pmod N$, where x is the element we selected to generate r. □

19.2 The Quadratic Residuosity Protocol

We consider the following protocol for proving quadratic residuosity.

- Verifier's input: an integer N (product of two unknown odd primes) and a integer $r \in \mathbb{Z}_N^*$;
- Prover's input: N, r and a square root $x \in Z_N^*$ such that $x^2 \bmod N = r$.
- The prover picks a random $y \in Z_N^*$ and sends $a := y^2 \bmod N$ to the verifier
- The verifier picks at random $b \in \{0, 1\}$ and sends b to the prover
- The prover sends back $c := y$ if $b = 0$ or $c := y \cdot x \bmod N$ if $b = 1$
- The verifier cheks that $c^2 \bmod N = a$ if $b = 0$ or that $c^2 \equiv a \cdot r \pmod N$ if $b = 1$, and accepts if so.

We show that:

- If r is a quadratic residue, the prover is given a square root x, and the parties follow the protocol, then the verifier accepts with probability 1;
- If r is not a quadratic residue, then for every cheating prover strategy P^*, the verifier rejects with probability $\geq 1/2$.

Proof:

Suppose r is not a quadratic residue. Then it is not possible that both a and $a \times r$ are quadratic residues. If $a = y^2 \bmod N$ and $a \times r = w^2 \bmod N$, then $r = w^2(y^{-1})^2 \bmod N$, meaning that r is also a perfect square.

With probability 1/2, the verifier rejects no matter what the Prover's strategy is. □

We now show that the above protocol is also zero knowledge. More precisely, we show the following.

Theorem 80 *For every verifier algorithm V^* of complexity $\leq t$ there is a simulator algorithm of average complexity $\leq 2t + (\log N)^{O(1)}$ such that for every odd composite N, every r which is a quadratic residue $\pmod N$ and every square root of x of r, the distributions*

$$S^*(N, r) \tag{19.1}$$

and

$$P(N, r, x) \leftrightarrow V^*(N, r) \tag{19.2}$$

are identical.

PROOF: The simulator S^* is defined as follows. It first picks $b_1 \in \{0, 1\}$ uniformly at random. It also picks $y \in Z_n^*$ uniformly at random. If $b_1 = 0$, set $a = y^2$ and if $b_1 = 1$, set $a = y^2 r^{-1}$. Note that irrespective of the value of b_1, a is a uniformly random element of Z_n^*. With this S^* simulates the interaction as follows. First, it simulates the prover by sending a. If the second round reply from V^* (call it b) is not the same as b_1, then it aborts the simulation and starts again. If not, then clearly $c = y$ is the reply the prover will send for both $b = 0$ and $b = 1$. Hence whenever the simulation is completed, the distribution of the simulated interaction is same as the actual interaction. Also observe that b_1 is a random bit statistically independent of a while b is totally dependent on a (and probably some other random coin tosses). Hence in expectation, in two trials of the simulation, one will be able to simulate one round of the actual interaction.Hence the expected time required for simulation is the time to simulate V^* twice and the time to do couple of multiplications in Z_n^*. So, in total it is at most $2t + (\log N)^{O(1)}$. □

Lecture 20

Proofs of Knowledge and Commitment Schemes

Summary

In this lecture we provide the formal definition of *proof of knowledge*, and we show that the quadratic residuosity protocol is also a proof of knowledge. We also start discussing the primitives required to prove that any language in NP admits a zero-knowledge proof.

20.1 Proofs of Knowledge

Suppose that L is a language in NP; then there is an NP relation $R_L(\cdot,\cdot)$ computable in polynomial time and polynomial $p(\cdot)$ such that $x \in L$ if and only if there exists a witness w such that $|w| \leq p(|x|)$ (where we use $|z|$ to denote the length of a bit-string z) and $R(x,w) = 1$.

Recall the definition of *soundness* of a proof system (P,V) for L: we say that the proof system has soundness error at most ϵ if for every $x \notin L$ and for every cheating prover strategy P^* the probability that $P^*(x) \leftrightarrow V(x)$ accepts is at most ϵ. Equivalently, if there is a prover strategy P^* such that the probability that $P^*(x) \leftrightarrow V(x)$ accepts is bigger than ϵ, then it must be the case that $x \in L$. This captures the fact that if the verifier accepts then it has high confidence that indeed $x \in L$.

In a proof-of-knowledge, the prover is trying to do more than convince the verifier that a witness exists proving $x \in L$; he wants to convince the verifier that he (the prover) *knows* a witness w such that $R(x,w) = 1$. How can we capture the notion that an algorithm "knows" something?

Definition 81 (Proof of Knowledge) *A proof system (P,V) for an NP relation R_L is a proof of knowledge with knowledge error at most ϵ and extractor slowdown es if there is an algorithm K (called a knowledge extractor) such that, for every prover strategy P^* of*

complexity $\leq t$ *and every input* x*, if*

$$\mathbb{P}[P^*(x) \leftrightarrow V(x) \text{ accepts}] \geq \epsilon + \delta$$

then $K(P^*, x)$ *outputs a* w *such that* $R(x, w) = 1$ *in average time at most*

$$es \cdot (n^{O(1)} + t) \cdot \delta^{-1}$$

In the definition, giving P^* as an input to K means to give the code of P^* to K. A stronger definition, which is satisfied by all the proof systems we shall see, is to let K be an oracle algorithm of complexity $\delta^{-1} \cdot es \cdot poly(n)$, and allow K to have oracle access to P^*. In such a case, "oracle access to a prover strategy" means that K is allowed to select the randomness used by P^*, to fix an initial part of the interaction, and then obtain as an answer what the next response from P^* would be given the randomness and the initial interaction.

Theorem 82 *The protocol for quadratic residuosity of the previous section is a proof of knowledge with knowledge error* $1/2$ *and extractor slowdown 2.*

PROOF: Consider an a such that the prover returns the correct answer both when $b = 0$ and $b = 1$. More precisely, when $b = 0$, prover returns a in the third round of the interaction and if $b = 1$, prover returns $a.r$ in the third round of interaction. If we can find such an a, then upon dividing the answers (for the cases when $b = 1$ and $b = 0$) returned by the prover strategy in third round, we can get the value of r. Note that if verifier V accepts with probability $\frac{1}{2} + \delta$, then by a Markov argument, we get that with probability δ, a randomly chosen $a \in Z_n^*$ is such that for both $b = 0$ and $b = 1$, the prover returns the correct answer. Clearly, the knowledge error of the protocol is $\frac{1}{2}$ and for one particular a, the prover strategy is executed twice. So, the extractor slowdown is 2. Note that in expectation, we will be sampling about $\frac{1}{\delta}$ times before we get an a with the aforementioned property. Hence, the total expected time for running K is $2 \cdot ((\log N)^{O(1)} + t) \cdot \delta^{-1}$ $\square$

20.2 Uses of Zero Knowledge proofs

In the coming lectures, we shall consider general multi-party protocols, an example of which might be playing "poker over the phone/internet". In this, one needs to devise a protocol such that n mutually distrusting players can play poker or any other game over the internet.

Our approach will be to first devise a protocol for the "honest but curious" case, in which all the players follow the protocol but they do try to gain information about others by simply tracking all the moves in the game. To go to the general case, we will require that every player gives a zero knowledge proof that it played honestly in every round. As one can see, this statement is much more general than say that two graphs are isomorphic. However, it is a statement which has a short certificate and hence is in NP. This gives motivation for our next topic which is developing zero knowledge protocols for every language in NP. We shall describe an important primitive for this purpose called *commitment schemes*.

20.3 Commitment Scheme

A commitment scheme is a two-phase protocol between a *Sender* and a *Receiver*. The Sender holds a message m and, in the first phase, it picks a random key K and then "encodes" the message using the key and sends the encoding (a *commitment* to m) to the Receiver. In the second phase, the Sender sends the key K to the Receiver can *open* the commitment and find out the content of the message m.

A commitment scheme should satisfy two security properties:

- **Hiding.** Receiving a commitment to a message m should give no information to the Receiver about m;
- **Binding.** The Sender cannot "cheat" in the second phase and send a different key K' that causes the commitment to open to a different message m'.

It is impossible to satisfy both properties against computationally unbounded adversaries. It is possible, however, to have schemes in which the Hiding property holds against computationally unbounded Receivers and the Binding property holds (under appropriate assumptions on the primitive used in the construction) for bounded-complexity Senders; and it is possible to have schemes in which the Hiding property holds (under assumptions) for bounded-complexity Receivers while the Binding property holds against any Sender. We shall describe a protocol of the second type, based on one-way permutations. The following definition applies to one-round implementations of each phase, although a more general definition could be given in which each phase is allowed to involve multiple interactions.

Definition 83 (Computationally Hiding, Perfectly Binding, Commitment Scheme) *A Perfectly Binding and (t, ϵ)-Hiding Commitment Scheme for messages of length ℓ is a pair of algorithms (C, O) such that*

- **Correctness.** *For every message m and key K,*

$$O(K, C(K, m)) = m$$

- **(t, ϵ)-Hiding.** *For every two messages $m, m' \in \{0,1\}^\ell$, the distributions $C(K, m)$ and $C(K, m')$ are (t, ϵ)-indistinguishable, where K is a random key, that is, for every algorithm A of complexity $\leq t$,*

$$|\,\mathbb{P}[A(C(K, m)) = 1] - \mathbb{P}[A(C(K, m')) = 1]| \leq \epsilon$$

- **Perfectly Binding.** *For every message m and every two keys K, K',*

$$O(K', C(K, m)) \in \{m, FAIL\}$$

In the following we shall refer to such a scheme (C, O) as simply a (t, ϵ)-secure commitment scheme.

Given a one-way permutation $f : \{0,1\}^n \to \{0,1\}^n$ and a hard-core predicate P, we consider the following construction of a one-bit commitment scheme:

- $C(K, m) := f(K), m \oplus P(K)$
- $O(K, (c_1, c_2))$ equals $FAIL$ if $f(K) \neq c_1$, and $P(K) \oplus c_2$ otherwise.

Theorem 84 *If P is a (t, ϵ)-secure hard core predicate for f, then the above construction is a $(t - O(1), 2\epsilon)$-secure commitment scheme.*

PROOF: The binding property of the commitment scheme is easy to argue as the commitment is a permutation of the key and the message. In particular, given $C(K, m) = (x, y)$, we can find the unique K and m that generate it as

$$K = f^{-1}(x) \quad \text{and} \quad m = y \oplus P(K) = y \oplus P(f^{-1}(x))$$

To prove the hiding property in the contrapositive, we want to take an algorithm which distinguishes the commitments of two messages and convert it to an algorithm which computes the predicate P with probability better than $1/2 + \epsilon$. Let A be such an algorithm which distinguishes two different messages (one of which must be 0 and the other must be 1). Then, we have that for A

$$\begin{aligned} & |\mathbb{P}[A(C(K, m)) = 1] - \mathbb{P}[A(C(K, m) = 1]| > 2\epsilon \\ \implies \quad & |\mathbb{P}[A(f(K), P(K) \oplus 0) = 1] - \mathbb{P}[A(f(K), P(K) \oplus 1) = 1]| > 2\epsilon \end{aligned}$$

Assume without loss of generality that the quantity in the absolute value is positive i.e.

$$\mathbb{P}[A(f(K), P(K)) = 1] - \mathbb{P}[A(f(K), P(K) \oplus 1) = 1] > 2\epsilon$$

Hence, A outputs 1 significantly more often when given the correct value of $P(K)$. As seen in previous lectures, we can convert this into an algorithm A' that predicts the value of $P(K)$. Algorithm A' takes $f(K)$ as input and generates a random bit b as a guess for $P(K)$. It then runs $A(f(K), b)$. Since A is correct more often on the correct value of $P(K)$, A' outputs b if $A(f(K), b) = 1$ and outputs $b \oplus 1$ otherwise. We can analyze its success

probability as below

$$\begin{aligned}
& \mathbb{P}[A'(f(K)) = P(K)] = \mathbb{P}[b = P(K)] \cdot \mathbb{P}[A(f(K), P(K)) = 1] \\
& + \mathbb{P}[b \neq P(K)] \cdot \mathbb{P}[A(f(K), P(K) \oplus 1) = 0] \\
= \; & \frac{1}{2} \cdot \mathbb{P}[A(f(K), P(K)) = 1] \\
& + \frac{1}{2} \cdot (1 - \mathbb{P}[A(f(K), P(K) \oplus 1) = 1]) \\
= \; & \frac{1}{2} + \frac{1}{2} \cdot (\mathbb{P}[A(f(K), P(K)) = 1] - \mathbb{P}[A(f(K), P(K) \oplus 1) = 1]) \\
\geq \; & \frac{1}{2} + \epsilon
\end{aligned}$$

Thus, A' predicts P with probability $1/2 + \epsilon$ and has complexity only $O(1)$ more than A (for generating the random bit) which contradicts the fact that P is (t, ϵ)-secure. □

There is a generic way to turn a one-bit commitment scheme into a commitment scheme for messages of length ℓ (just concatenate the commitments of each bit of the message, using independent keys).

Theorem 85 *Let (O, C) be a (t, ϵ)-secure commitment scheme for messages of length k such that $O(\cdot, \cdot)$ is computable in time r. Then the following scheme $(\overline{C}, \overline{O})$ is a $t - O(r \cdot \ell), \epsilon \cdot l)$-secure commitment scheme for message of length $k \cdot \ell$:*

- $\overline{C}(K_1, \ldots, K_\ell, m) := C(K_1, m_1), \ldots, C(K_\ell, m_\ell)$
- $\overline{O}(K_1, \ldots, K_\ell, c_1, \ldots, c_\ell)$ *equals $FAIL$ if at least one of $O(K_i, c_i)$ outputs $FAIL$; otherwise it equals $O(K_1, c_1), \ldots, O(K_\ell, c_\ell)$.*

PROOF: The commitment to m is easily seen to be binding since the commitments to each bit of m are binding. The soundness can be proven by a hybrid argument.

Suppose there is an A algorithm distinguishing $\overline{C}(K_1, \ldots, K_\ell, m)$ and $C(K_1, \ldots, K_\ell, m)$ with probability more than $\epsilon \cdot \ell$. We then consider "hybrid messages" $m^{(0)}, \ldots, m^{(\ell)}$, where $m^{(i)} = m'_1 \ldots m'_i m_{i+1}, \ldots, m_\ell$. By a hybrid argument, there is some i such that

$$\left| \mathbb{P}[A(K_1, \ldots, K_\ell, m^{(i)}) = 1] - \mathbb{P}[A(K_1, \ldots, K_\ell, m^{(i+1)}) = 1] \right| > \epsilon$$

But since $m^{(i)}$ and $m^{(i+1)}$ differ in only one bit, we can get an algorithm A' that breaks the hiding property of the one bit commitment scheme $C(\cdot, \cdot)$. A', given a commitment c, outputs

$$A'(c) \; = \; A(C(K_1, m_1), \ldots, C(K_i, m_i), c, C(K_{i+2}, m'_{i+2}), \ldots, C(K_\ell, m'_\ell))$$

Hence, A' has complexity at most $t + O(r \cdot l)$ and distinguishes $C(K_{i+1}, m_{i+1})$ from $C(K_{i+1}, m'_{i+1})$. □

There is also a construction based on one-way permutations that is better in terms of key length.

Lecture 21

Zero Knowledge Proofs of 3-Colorability

Summary

In this lecture we give the construction and analysis of a zero-knowledge protocol for the 3-coloring problem. Via reductions, this extends to a protocol for any problem in NP. We will only be able to establish a weak form of zero knowledge, called "computational zero knowledge" in which the output of the simulator and the interaction in the protocol are computationally indistinguishable (instead of identical). It is considered unlikely that NP-complete problem can have zero-knowledge protocols of the strong type we defined in the previous lectures.

21.1 A Protocol for 3-Coloring

We assume we have a (t, ϵ)-secure commitment scheme (C, O) for messages in the set $\{1, 2, 3\}$.

The prover P takes in input a 3-coloring graph $G = ([n], E)$ (we assume that the set of vertices is the set $\{1, \ldots, n\}$ and use the notation $[n] := \{1, \ldots, n\}$) and a proper 3-coloring $\alpha : [n] \to \{1, 2, 3\}$ of G (that is, α is such that for every edge $(u, v) \in E$ we have $\alpha(u) \neq \alpha(v)$). The verifier V takes in input G. The protocol, in which the prover attempts to convince the verifier that the graph is 3-colorable, proceeds as follows:

- The prover picks a random permutation $\pi : \{1, 2, 3\} \to \{1, 2, 3\}$ of the set of colors, and defines the 3-coloring $\beta(v) := \pi(\alpha(v))$. The prover picks n keys $K_1, \ldots, K_n$ for (C, O), constructs the commitments $c_v := C(K_v, \beta(v))$ and sends $(c_1, \ldots, c_n)$ to the verifier;
- The verifier picks an edge $(u, v) \in E$ uniformly at random, and sends (u, v) to the prover;

- The prover sends back the keys K_u, K_v;
- If $O(K_u, c_u)$ and $O(K_v, c_v)$ are the same color, or if at least one of them is equal to $FAIL$, then the verifier rejects, otherwise it accepts

Theorem 86 *The protocol is complete and it has soundness error at most* $(1 - 1/|E|)$.

PROOF: The protocol is easily seen to be complete, since if the prover sends a valid 3-coloring, the colors on endpoints of every edge will be different.

To prove the soundness, we first note that if any commitment sent by the prover opens to an invalid color, then the protocol will fail with probability at least $1/|E|$ when querying an edge adjacent to the corresponding vertex (assuming the graph has no isolated vertices - which can be rivially removed). If all commitments open to valid colos, then the commitments define a 3-coloring of the graph. If the graph is not 3-colorable, then there must be at least one edge e both of whose end points receive the same color. Then the probability of the verifier rejecting is at least the probability of choosing e, which is $1/|E|$. □

Repeating the protocol k times sequentially reduces the soundness error to $(1 - 1/|E|)^k$; after about $27 \cdot |E|$ repetitions the error is at most about 2^{-40}.

21.2 Simulability

We now describe, for every verifier algorithm V^*, a simulator S^* of the interaction between V^* and the prover algorithm.

The basic simulator is as follows:

Algorithm S^*_{1round}

- Input: graph $G = ([n], E)$
- Pick random coloring $\gamma : [n] \to \{1, 2, 3\}$.
- Pick n random keys $K_1, \ldots, K_n$
- Define the commitments $c_i := C(K_i, \gamma(i))$
- Let (u, v) be the 2nd-round output of V^* given G as input and $c_1, \ldots, c_n$ as first-round message
- If $\gamma(u) = \gamma(v)$, then output FAIL
- Else output $((c_1, \ldots, c_n), (u, v), (K_u, K_v))$

And the procedure $S^*(G)$ simply repeats $S^*_{1round}(G)$ until it provides an output different from $FAIL$.

It is easy to see that the output distribution of $S^*(G)$ is always *different* from the actual distribution of interactions between P and V^*: in the former, the first round is almost always a commitment to an invalid 3-coloring, in the latter, the first round is always a valid 3-coloring.

We shall prove, however, that the output of $S^*(G)$ and the actual interaction of P and V^* have *computationally indistinguishable* distributions provided that the running time of V^* is bounded and that the security of (C, O) is strong enough.

For now, we prove that $S^*(G)$ has efficiency comparable to V^* provided that security of (C, O) is strong enough.

Theorem 87 *Suppose that (C, O) is $(t + O(nr), \epsilon/(n \cdot |E|))$-secure and C is computable in time $\leq r$ and that V^* is a verifier algorithm of complexity $\leq t$.*

Then the algorithm S^*_{1round} as defined above has probability at most $\frac{1}{3} + \epsilon$ of outputting $FAIL$.

The proof of Theorem 87 relies on the following result.

Lemma 88 *Fix a graph G and a verifier algorithm V^* of complexity $\leq t$.*

Define $p(u, v, \alpha)$ to be the probability that V^* asks the edge (u, v) at the second round in an interaction in which the input graph is G and the first round is a commitment to the coloring α.

Suppose that (C, O) is $(t + O(nr), \epsilon/n)$-secure, and C is computable in time $\leq r$.

Then for every two colorings α, β and every edge (u, v) we have

$$|p(u, v, \alpha) - p(u, v, \beta)| \leq \epsilon$$

PROOF: If $p(u, v, \alpha)$ and $p(u, v, \beta)$ differ by more than ϵ for any edge (u, v), then we can define an algorithm which distinguishes the n commitments corresponding to α from the n commitments corresponding to β. A simply runs the verifier given commitments for n colors and outputs 1 if the verifier selects the edge (u, v) in the second round.

Then, by assumption, A ϵ-distinguishes the n commitments corresponding to α from the n commitments corresponding to β in time $t + O(nr)$. However, by Theorem 85, this means that (C, O) is not $(t + O(nr), \epsilon/n)$-secure which is a contradiction. □

Given the lemma, we can now easily prove the theorem.

PROOF: (of Theorem 87) The probability that S^*_{1round} outputs $FAIL$ is given by

$$\mathbb{P}\left[S^*_{1round} = FAIL\right] = \frac{1}{3^n} \cdot \sum_{c \in \{1,2,3\}^n} \sum_{\substack{(u,v) \in E \\ c(u) \neq c(v)}} p(u, v, c)$$

Let $\mathbf{1}$ denote the coloring which assigns the color 1 to every vertex. Then using Lemma 88 we bound the above as

$$\begin{aligned}
\mathbb{P}\left[S^*_{1round} = FAIL\right] &\leq \frac{1}{3^n} \cdot \sum_{c \in \{1,2,3\}^n} \sum_{\substack{(u,v)\in E \\ c(u)\neq c(v)}} (p(u,v,\mathbf{1}) + \epsilon) \\
&= \sum_{(u,v)\in E} p(u,v,\mathbf{1}) \left(\sum_{c: c(u)\neq c(v)} \frac{1}{3^n} \right) + \epsilon \\
&= \frac{1}{3} \sum_{(u,v)\in E} p(u,v,\mathbf{1}) + \epsilon \\
&= \frac{1}{3} + \epsilon
\end{aligned}$$

where in the second step we used the fact that $c(u) \neq c(v)$ for a 1/3 fraction of all the colorings and the last step used that the probability of V^* selecting some edge given the coloring $\mathbf{1}$ is 1. □

21.3 Computational Zero Knowledge

We want to show that this simulator construction establishes the *computational zero knowledge* property of the protocol, assuming that (C, O) is secure. We give the definition of computational zero knowledge below.

Definition 89 (Computational Zero Knowledge) *We say that a protocol (P, V) for 3-coloring is (t, ϵ) computational zero knowledge with simulator overhead $so(\cdot)$ if for every verifier algorithm V^* of complexity $\leq t$ there is a simulator S^* of complexity $\leq so(t)$ on average such that for every algorithm D of complexity $\leq t$, every graph G and every valid 3-coloring α we have*

$$|\,\mathbb{P}[D(P(G,\alpha) \leftrightarrow V^*(G)) = 1] - \mathbb{P}[D(S^*(G)) = 1]| \leq \epsilon$$

Theorem 90 *Suppose that (C, O) is $(2t + O(nr), \epsilon/(4 \cdot |E| \cdot n))$-secure and that C is computable in time $\leq r$.*

Then the protocol defined above is (t, ϵ) computational zero knowledge with simulator overhead at most $1.6 \cdot t + O(nr)$.

21.4 Proving that the Simulation is Indistinguishable

In this section we prove Theorem 90.

Suppose that the Theorem is false. Then there is a graph G, a 3-coloring α, a verifier algorithm V^* of complexity $\leq t$, and a distinguishing algorithm D also of complexity $\leq t$ such that

$$|\,\mathbb{P}[D(P(G,\alpha) \leftrightarrow V^*(G)) = 1 - \mathbb{P}[D(S^*(G)) = 1]| \geq \epsilon$$

Let $2R_{u,v}$ be the event that the edge (u,v) is selected in the second round; then

$$\begin{aligned}
\epsilon &\leq |\,\mathbb{P}[D(P(G,\alpha) \leftrightarrow V^*(G)) = 1] - \mathbb{P}[D(S^*(G)) = 1]| \\
&= \Bigg| \sum_{(u,v)\in E} \mathbb{P}[D(P(G,\alpha) \leftrightarrow V^*(G)) = 1 \wedge 2R_{u,v}] \\
&\quad - \sum_{(u,v)\in E} \mathbb{P}[D(S^*(G)) = 1 \wedge 2R_{u,v}] \Bigg| \\
&\leq \sum_{(u,v)\in E} |\,\mathbb{P}[D(P(G,\alpha) \leftrightarrow V^*(G)) = 1 \wedge 2R_{u,v}] \\
&\quad - \mathbb{P}[D(S^*(G)) = 1 \wedge 2R_{u,v}]|
\end{aligned}$$

So there must exist an edge $(u^*, v^*) \in E$ such that

$$|\,\mathbb{P}[D(P \leftrightarrow V^*) = 1 \wedge 2R_{u^*,v^*}] - \mathbb{P}[D(S^*) = 1 \wedge 2R_{u^*,v^*}]| \geq \frac{\epsilon}{|E|} \tag{21.1}$$

(We have omitted references to G, α, which are fixed for the rest of this section.)

Now we show that there is an algorithm A of complexity $2t+O(nr)$ that is able to distinguish between the following two distributions over commitments to $3n$ colors:

- **Distribution (1)** commitments to the $3n$ colors $1, 2, 3, 1, 2, 3, \ldots, 1, 2, 3$;
- **Distribution (2)** commitments to $3n$ random colors

Algorithm A:

- Input: 3n commitments $d_{a,i}$ where $a \in \{1,2,3\}$ and $i \in \{1,\ldots,n\}$;
- Pick a random permutation $\pi : \{1,2,3\} \to \{1,2,3\}$
- Pick random keys K_{u^*}, K_{v^*}
- Construct the sequence of commitments $c_1, \ldots, c_n$ by setting:
  - $c_{u^*} := C(K_{u^*}, \pi(\alpha(u^*))$
  - $c_{v^*} := C(K_{v^*}, \pi(\alpha(v^*))$
  - for every $w \in [n] - \{u^*, v^*\}$, $c_w := d_{\pi(\alpha(w)),w}$
- If the 2nd round output of V^* given G and $c_1, \ldots, c_n$ is different from (u^*, v^*) output 0

- Else output $D((c_1, \ldots, c_n), (u^*, v^*), (K_{u^*}, K_{v^*}))$

First, we claim that

$$\mathbb{P}[A(\text{Distribution 1}) = 1] = \mathbb{P}[D(P \leftrightarrow V^*) = 1 \wedge 2R_{u^*,v^*}] \tag{21.2}$$

This follows by observing that A on input Distribution (1) behaves exactly like the prover given the coloring α, and that A accepts if and only if the event $2R_{u^*,v^*}$ happens and D accepts the resulting transcript.

Next, we claim that

$$|\,\mathbb{P}[A(\text{Distribution 2}) = 1] - \mathbb{P}[D(S^*) = 1 \wedge 2R_{u^*,v^*}]| \leq \frac{\epsilon}{2|E|} \tag{21.3}$$

To prove this second claim, we introduce, for a coloring γ, the quantity $DA(\gamma)$, defined as the probability that the following probabilistic process outputs 1:

- Pick random keys $K_1, \ldots, K_n$
- Define commitments $c_u := C(K_u, \gamma(u))$
- Let (u, v) be the 2nd round output of V^* given the input graph G and first round message $c_1, \ldots, c_n$
- Output 1 iff $(u, v) = (u^*, v^*)$, $\gamma(u^*) \neq \gamma(v^*)$, and

$$D((c_1, \ldots, c_n), (u^*, v^*), (K_{u^*}, K_{v^*})) = 1$$

Then we have

$$\mathbb{P}[A(\text{Distribution 2}) = 1] = \sum_{\gamma:\gamma(u^*)\neq\gamma(v^*)} \frac{3}{2} \cdot \frac{1}{3^n} \cdot DA(\gamma) \tag{21.4}$$

Because A, on input Distribution 2, first prepares commitments to a coloring chosen uniformly at random among all $1/(6 \cdot 3^{n-2})$ colorings such that $\gamma(u^*) \neq \gamma(v^*)$ and then outputs 1 if and only if, given such commitments as first message, V^* replies with (u^*, v^*) and the resulting transcript is accepted by D.

We also have

$$\mathbb{P}[D(S^*) = 1 \wedge 2R_{u^*,v^*}] = \frac{1}{\mathbb{P}[S^*_{1Round} \neq FAIL]} \cdot \sum_{\gamma:\gamma(u^*)\neq\gamma(v^*)} \frac{1}{3^n} \cdot DA(\gamma) \tag{21.5}$$

To see why Equation (21.5) is true, consider that the probability that S^* outputs a particular transcript is exactly $1/\,\mathbb{P}[S^*_{1Round} \neq FAIL]$ times the probability that S^*_{1Round} outputs that transcript. Also, the probability that S^*_{1Round} outputs a transcript which involves (u^*, v^*)

at the second round and which is accepted by $D()$ conditioned on γ being the coloring selected at the beginning is $DA(\gamma)$ if γ is a coloring such that $\gamma(u^*) \neq \gamma(v^*)$, and it is zero otherwise. Finally, S^*_{1Round} selects the initial coloring uniformly at random among all possible 3^n coloring.

From our security assumption on (C, O) and from Lemma 6 in Lecture 27 we have

$$\left| \mathbb{P}[S^*_{1Round} \neq FAIL] - \frac{2}{3} \right| \leq \frac{\epsilon}{4|E|} \tag{21.6}$$

and so the claim we made in Equation (21.3) follows from Equation (21.4), Equation (21.5), Equation (21.6) and the fact that if p, q are quantities such that $\frac{3}{2}p \leq 1$, $\frac{1}{q} \cdot p \leq 1$, and $\left|q - \frac{2}{3}\right| \leq \delta \leq \frac{1}{6}$ (so that $q \geq 1/2$), then

$$\left| \frac{3}{2}p - \frac{1}{q}p \right| = \frac{3}{2} \cdot p \cdot \frac{1}{q} \cdot \left| q - \frac{2}{3} \right| \leq 2\delta$$

(We use the above inequality with $q = \mathbb{P}[S^*_{1Round} \neq FAIL]$, $\delta = \epsilon/4|E|$, and $p = \sum_{\gamma:\gamma(u^*)\neq\gamma(v^*)} \frac{1}{3^n} DA(\gamma)$.)
Having proved that Equation (21.3) holds, we get

$$|\, \mathbb{P}[A(\text{Distribution 1}) = 1] - \mathbb{P}[A(\text{Distribution 2}) = 1]| \geq \frac{\epsilon}{2|E|}$$

where A is an algorithm of complexity at most $2t + O(nr)$. Now by a proof similar to that of Theorem 3 in Lecture 27, we have that (C, O) is not $(2t + O(nr), \epsilon/(2|E|n))$ secure.

Bibliography

[BLR93] Manuel Blum, Michael Luby, and Ronitt Rubinfeld. Self-testing/correcting with applications to numerical problems. *Journal of Computer and System Sciences*, 47(3):549–595, 1993. Preliminary version in *Proc. of STOC'90*. 62

[GL89] O. Goldreich and L. Levin. A hard-core predicate for all one-way functions. In *Proceedings of the 21st ACM Symposium on Theory of Computing*, pages 25–32, 1989. 62