

Welcome to CS 127!

This handout contains:

- Course overview and syllabus
- Lecture notes for lecture 1: Tue 1/26.
- Lecture notes for lecture 2: Thu 1/28.
- Homework 1 – The first problem set, due **Friday Feb 5, 5pm**.

All information about the course (including links to the canvas and Piazza websites there) can be found on its home page:

<http://www.boazbarak.org/cs127> (google “CS127 boaz”)

If you take this course, you need to:

- Read the lecture notes for lectures 1 and 2; preferably ***before*** the lecture on Thursday and in any case no later than the weekend. (See also mathematical background document on the web page.)
- Sign up to **Piazza** website today (so you can get announcements) and post intro message about yourself on discussion board sometime this week (see Exercise 1 in the homework).
- From week 2 on, you will need to read lecture notes for each lecture ***before*** it's given, and do a short reading comprehension quiz on the canvas website before the Thursday lecture. First quiz will be on lectures 1-4 due on Thu 2/4 11:30am.
- Problem sets 2 and onwards will be due on **Tuesday 11:30am** the week after they are given (submit via canvas).
- The ideal way to prepare the homework is to type them using **Markdown**. Source files for the P-sets are on the web and you can use <https://www.madoko.net/> to edit them and create a PDF.

Foreword and Syllabus

“Human ingenuity cannot concoct a cipher which human ingenuity cannot resolve.” Edgar Allan Poe, 1841

Cryptography - the art or science of “secret writing” - has been around for several millenia, and for almost all of that time Edgar Allan Poe’s quote above held true. Indeed, the history of cryptography is littered with the figurative corpses of cryptosystems believed secure and then broken, and sometimes with the actual corpses of those who have mistakenly placed their faith in these cryptosystems. Yet, something changed in the last few decades. New cryptosystems have been found that have not been broken despite being subjected to immense efforts involving both human ingenuity and computational power on a scale that completely dwarves the “crypto breakers” of Poe’s time. Even more amazingly, these cryptosystems are not only seemingly unbreakable, but they also achieve this under much harsher conditions. Not only do today’s attackers have more computational power but they also have more data to work with. In Poe’s age, an attacker would be lucky if they got access to more than a few ciphertexts with known plaintexts. These days attackers might have massive amounts of data- terabytes or more - at their disposal. In fact, with *public key* encryption, an attacker can generate as many ciphertexts as they wish.

These new types of cryptosystems, both more secure and more versatile, have enabled many applications that in the past were not only impossible but in fact *unimaginable*. These include secure communication without sharing a secret, electronic voting without a trusted authority, anonymous digital cash, and many more. Cryptography now supplies crucial infrastructure without which much of the modern “communication economy” could not function.

This course is about the story of this cryptographic revolution. However, beyond the cool applications and the crucial importance of cryptography to our society, it contains also intellectual and mathematical beauty. To understand these often paradoxical notions of cryptography, you need to think differently, adapting the point of view of an attacker, and (as we will see) sometimes adapting the points of view of other hypothetical entities. More than anything, this course is about this cryptographic way of thinking. It may not be immediately applicable to protecting your credit card information or to building a secure system, but learning a new way of thinking is its own reward.

Syllabus

In this fast-paced course, I plan to start from the very basic notions of cryptography and by the end of the term reach some of the exciting advances that happened in the last few years such as the construction of *fully homomorphic encryption*, a notion that Brian Hayes called “one of the most amazing magic tricks in all of computer science”, and *indistinguishability obfuscators* which are even more amazing. To achieve this, our focus will be on *ideas* rather than *implementations* and so we will present cryptographic notions in their pedagogically simplest form– the one that best illustrates the underlying concepts– rather than the one that is most efficient, widely deployed, or conforms to Internet standards. We will discuss some examples of practical systems and attacks, but only when these serve to illustrate a conceptual point.

Depending on time, I plan to cover the following notions:

- Part I: Introduction
 1. **How do we define security for encryption?** Arguably the most important step in breaking out of the “build-break-tweak” cycle that Poe’s quote described has been the idea that we can have a *mathematically precise definition* of security, rather than relying on fuzzy notions, that allow us only to determine with certainty that a system is *broken* but never have a chance of *proving* that a system is *secure*.
 2. **Perfect security and its limitations:** Showing the possibility (and the limitations) of encryptions that are perfectly secure regardless the attacker’s computational resources.
 3. **Computational security:** Bypassing the above limitations by restricting to computationally efficient attackers. Proofs of security by reductions.
- Part II: Private Key Cryptography
 1. **Pseudorandom generators:** The basic building block of cryptography, which also provided a new twist on the age-old philosophical and scientific question of the nature of randomness.
 2. **Pseudorandom functions, permutations, block ciphers:** Block ciphers are the working horse of crypto.
 3. **Authentication and active attacks:** *Authentication* turns out to be as crucial, if not more, to security than *secrecy* and often a precondition to the latter. We’ll talk about notions such as Message Authentication Codes and Chosen-Ciphertext-Attack secure encryption, as well as real-world examples why these notions are necessary.

4. **Hash functions and the “Random Oracle Model”:** Hash functions are used all over in crypto, including for verifying integrity, entropy distillation, and many other cases.
 5. **Building pseudorandom generators from one-way permutations (optional):** Justifying our “axiom” of pseudo-random generators by deriving it from a weaker assumption.
- Part III: Public key encryption
 1. **Public key cryptography and the obfuscation paradigm:** How did Diffie, Hellman, Merkle, Ellis even dare to *imagine* the possibility of public key encryption?
 2. **Constructing public key encryption: Factoring, discrete log, and lattice based systems:** We’ll discuss several variants for constructing public key systems, including those that are widely deployed such as RSA, Diffie-Hellman, and the elliptic curve variants, as well as some variants of *lattice based cryptosystems* that have the advantage of not being broken by quantum computers, as well as being more versatile. The former is the reason why the NSA has advised people to transition to lattice-based cryptosystems in the not too far future.
 3. **Signature schemes:** These are the public key versions of authentication though interestingly are easier to construct in some sense than the latter.
 4. **Active attacks for encryption:** Chosen ciphertext attacks for public key encryption.
 - Part IV: Advanced notions
 1. **Fully homomorphic encryption:** Computing on encrypted data.
 2. **Multiparty secure computation:** An amazing construction that enables applications such as playing poker over the net without trusting the server, privacy preserving data mining, electronic auctions without a trusted auctioneer, electronic elections without a trusted central authority.
 3. **Zero knowledge proofs:** Prove a statement without revealing the reason to *why* its true.
 4. **Quantum computing and cryptography:** Shor’s algorithm to break RSA and friends. Quantum key distribution. On “quantum resistant” cryptography.
 5. **Indistinguishability obfuscation:** Construction of indistinguishability obfuscators, the potential “master tool” for crypto.
 6. **Practical protocols:** Techniques for constructing practical protocols for particular tasks as opposed to general (and often inefficient)

feasibility proofs.

Prerequisites

The main prerequisite is the ability to read, write (and even enjoy!) mathematical proofs. In addition, familiarity with algorithms, basic probability theory and basic linear algebra will be helpful. We'll only use fairly basic concepts from all these areas: e.g. Oh-notation- e.g. $O(n)$ running time- from algorithms, notions such as events, random variables, expectation, from probability theory, and notions such as matrices, vectors, and eigenvectors. Mathematically mature students should be able to pick up the needed notions on their own. See the "mathematical background" handout for more details.

No programming knowledge is needed. If you're interested in the course but are not sure if you have sufficient background, or you have any other questions, please don't hesitate to contact me.

Why is cryptography hard?

Cryptography is a hard topic. Over the course of history, many brilliant people have stumbled in it, and did not realize subtle attacks on their ciphers. Even today it is frustratingly easy to get crypto wrong, and often system security is compromised because developers used crypto schemes in the wrong, or at least suboptimal, way. Why is this topic (and this course) so hard? Some of the reasons include:

- To argue about the security of a cryptographic scheme, you have to think like an attacker. This requires a very different way of thinking than what we are used to when developing algorithms or systems, and arguing that they perform well.
- To get robust assurances of security you need to argue about *all possible attacks*. The only way I know to analyze this infinite set is via *mathematical proofs*. Moreover, these types of mathematical proofs tend to be rather different than the ones most mathematicians typically work with. Because the proof itself needs to take the viewpoint of the attacker, these often tend to be proofs by contradiction and involve several twists of logic that take some getting used to.
- As we'll see in this course, even *defining* security is a highly non trivial task. Security definitions often get subtle and require quite a lot of creativity. For example, the way we model in general a statement such as "An attacker Eve does not get more information from observing a system above what she knew a-priori" is that we posit a "hypothetical alter ego" of Eve called Lilith who knows everything Eve knew a-priori but does not get to observe the actual interaction in the system. We then want to prove that anything

that Eve learned could also have been learned by Lilith. If this sounds confusing, it is. But it is also fascinating, and leads to ways to argue mathematically about *knowledge* as well as beautiful generalizations of the notion of encryption and protecting communication into schemes for protecting *computation* .

If cryptography is so hard, is it really worth studying? After all, given this subtlety, a single course in cryptography is no guarantee of using (let alone inventing) crypto correctly. In my view, regardless of its immense and growing practical importance, cryptography is worth studying for its *intellectual* content. There are many areas of science where we achieve goals once considered to be science fiction. But cryptography is an area where current achievements are so fantastic that in the thousands of years of secret writing people did not even dare *imagine* them. Moreover, cryptography may be hard because it forces you to think differently, but it is also rewarding because it teaches you to think differently. And once you pass this initial hurdle, and develop a “cryptographer’s mind”, you might find that this point of view is useful in areas that seem to have nothing to do with crypto.

Lecture 1 - Introduction

Optional additional reading: Chapters 1 and 2 of Katz-Lindell book.¹

Ever since people started to communicate, there were some messages that they wanted kept secret. Thus cryptography has an old though arguably *undistinguished* history. For a long time cryptography shared similar features with Alchemy as a domain in which many otherwise smart people would be drawn into making fatal mistakes. The definitive text on the history of cryptography is David Kahn's "The Codebreakers", whose title already hints at the ultimate fate of most cryptosystems.² (See also "The Code Book" by Simon Singh.) We now recount just a few stories to get a feel for this field. But, before we do so, we should introduce the cast of characters. The basic setting of "encryption" or "secret writing" is the following: one person, whom we will call **Alice**, wishes to send another person, whom we will call **Bob**, a **secret** message. Since Alice and Bob are not in the same room (perhaps because Alice is imprisoned in a castle by her cousin the queen of England), they cannot communicate directly and need to send their message in writing. Alas, there is a third person, whom we will call **Eve**, that can see their message. Therefore Alice needs to find a way to *encode* or *encrypt* the message so that only Bob (and not Eve) will be able to understand it.

In 1587, Mary the queen of Scots, and the heir to the throne of England, wanted to arrange the assassination of her cousin, queen Elisabeth I of England, so that she could ascend to the throne and finally escape the house arrest under which she has been for the last 18 years. As part of this complicated plot, she sent a coded letter to Sir Anthony Babington. It is what's known as a *substitution cipher* where each letter is transformed into a different symbol, and so the resulting letter looks something like the following:

At a first look, such a letter might seem rather inscrutable- a meaningless sequence of strange symbols. However, after some thought, one might recognize that these symbols *repeat* several times and moreover that different symbols repeat with different frequencies. Now it doesn't take a large leap of faith to assume that perhaps each symbol corresponds to a different letter and the more frequent symbols correspond to letters that occur in the alphabet with higher

¹Because this is an undergraduate course, I omit almost all references and credits from these lecture notes unless the name has become standard in the literature, or I believe that the story of some discovery can serve a pedagogical point. See the Katz-Lindell book for historical notes and references.

²Traditionally, *cryptography* was the name for the activity of *making* codes, while *cryptanalysis* is the name for the activity of *breaking* them, and *cryptology* is the name for the union of the two. These days *cryptography* is often used as the name for the broad science of constructing and analyzing the security of not just encryptions but many schemes and protocols for protecting the confidentiality and integrity of communication and computation.

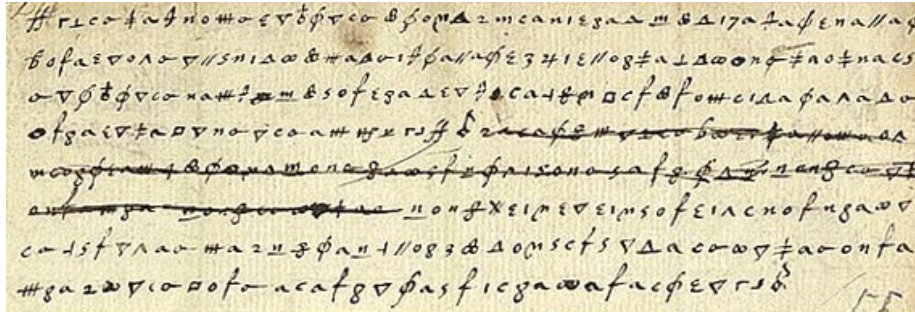


Figure 1: Snippet from encrypted communication between queen Mary and Sir Babington

frequency. From this observation, there is a short gap to completely breaking the cipher, which was in fact done by queen Elisabeth's spies who used the decoded letters to learn of all the co-conspirators and to convict queen Mary of treason, a crime for which she was executed.

Trusting in superficial security measures (such as using “indecipherable” symbols) is a trap that users of cryptography have been falling into again and again over the years. As in many things, this is the subject of a great XKCD cartoon:



Figure 2: On the added security of using uncommon symbols

The *Vigenère cipher* is named after Blaise de Vigenère who described it in a book in 1586 (though it was invented earlier by Bellaso). The idea is to use a collection of substitution ciphers - if there are n different ciphers then the first letter of the plaintext is encoded with the first cipher, the second with the second cipher, the n^{th} with the n^{th} cipher, and then the $n + 1^{\text{st}}$ letter is again encoded with the first cipher. The key is usually a word or a phrase of n letters, and the i^{th} substitution cipher is obtained by shifting each letter k_i positions in the alphabet. This “flattens” the frequencies and makes it much harder to do frequency analysis, which is why this cipher was considered “unbreakable” for 300+ years and got the nickname “le chiffre indéchiffrable” (“the unbreakable cipher”). Charles Babbage cracked the Vigenère cipher in 1854 but did not publish it. In 1863 Friedrich Kasiski broke the cipher and published the result. The idea is that once you guess the length of the cipher, you can reduce the task to breaking a simple substitution cipher which can be done via frequency analysis (can you see why?). Confederate generals used Vigenère regularly during the civil war, and their messages were routinely cryptanalyzed by Union officers.



Figure 3: Confederate Cipher Disk for implementing the Vigenère cipher



Figure 4: Confederate encryption of the message “Gen’l Pemberton: You can expect no help from this side of the river. Let Gen’l Johnston know, if possible, when you can attack the same point on the enemy’s lines. Inform me also and I will endeavor to make a diversion. I have sent some caps. I subjoin a despatch from General Johnston.”

The story of the *Enigma* cipher had been told many times, and you can get some information on it from Kahn’s book as well as Andrew Hodges’ biography of Alan Turing. This was a mechanical cipher (looking like a typewriter) where each letter typed would get mapped into a different letter depending on the (rather complicated) key and current state of the machine which had several rotors that rotated at different paces. An identically wired machine at the other end could be used to decrypt. Just as many ciphers in history, this has also been believed by the Germans to be “impossible to break” and even quite late in the war they refused to believe it was broken despite mounting evidence to that effect. (In fact, some German generals refused to believe it was broken even *after* the war.) Breaking Enigma was an heroic effort which was initiated by the Poles and then completed by the British at Bletchley Park; as part of this effort they built arguably the world’s first large scale mechanical computation devices (though they looked more similar to washing machines than to iPhones). They were also helped along the way by some quirks and errors of the German operators. For example, the fact that their messages ended with “Heil Hitler” turned out to be quite useful. Here is one entertaining anecdote: the Enigma machine would never map a letter to itself. In March 1941, Mavis Batey, a cryptanalyst at Bletchley Park received a very long message that she tried to decrypt. She then noticed a curious property— the message did *not* contain the letter “L”.³ She realized that it must be the case that the operator, perhaps to test the machine, have simply sent out a message where he repeatedly pressed the letter “L”. This observation helped her decode the next message, which helped inform of a planned Italian attack and secure a resounding British victory in what became known as “the Battle of Cape Matapan”. Mavis also helped break another Enigma machine which helped in the effort of feeding the Germans with the false information that the main allied invasion would take place in Pas de Calais rather than on Normandy. See this interview with Sir Harry Hinsley for more on the effect of breaking the Enigma on the war. General Eisenhower said that the intelligence from Bletchley park was of “priceless value” and made a “very decisive contribution to the Allied war effort”.

Defining encryptions

We now turn to actually defining what is an encryption scheme. Clearly we can encode every message as a string of bits, i.e., an element of $\{0,1\}^\ell$ for some ℓ . Similarly, we can encode the *key* as a string of bits as well, i.e., an element of $\{0,1\}^n$ for some n . Thus, we can think of an encryption scheme as composed of two functions. The *encryption function* E maps a secret key $k \in \{0,1\}^n$ and a message (known also as *plaintext*) $m \in \{0,1\}^\ell$ into a *ciphertext* $c \in \{0,1\}^o$ for some o . We write this as $c = E_k(m)$. The *decryption function* D does the reverse operation, mapping the secret key k and the ciphertext c back into the

³Here is a nice exercise: compute (up to an order of magnitude) the probability that a 50-letter long message composed of random letters will end up not containing the letter “L”.

plaintext message m , which we write as $m = D_k(c)$. The basic equation is that if we use the same key for encryption and decryption, then we should get the same message back. That is, for every $k \in \{0, 1\}^n$ and $m \in \mathcal{Z}^{\ell}$,

$$m = D_k(E_k(m)) .$$

A note on notation: We will always use i, j, ℓ, n, o to denote natural numbers. n will often denote the length of our secret key, and ℓ the length of the message, sometimes also known as “block length” since longer messages are simply chopped into “blocks” of length ℓ and also appropriately padded. We will use k to denote the secret key, m to denote the secret plaintext message, and c to denote the encrypted ciphertext. Note that c, m and k are bit strings of lengths o, ℓ and n respectively. The length of the secret key is often known as the “security parameter” and in other texts it is often denoted by k or κ . We use n to correspond with the standard algorithmic notation for input length (as in $O(n)$ time algorithms).

Note that this definition so far says nothing about security and does not rule out trivial “encryption” schemes such as the scheme $E_k(m) = m$ that simply outputs the plaintext as is. Defining security is tricky, and we’ll take it one step at a time, but let’s start by pondering what is secret and what is not. A priori we are thinking of an attacker Eve that simply sees the ciphertext C and does not know anything on how it was generated. So, it does not know the details of E and D , and certainly does not know the secret key k . However, many of the troubles past cryptosystems went through was caused by them relying on “security through obscurity”—trusting that the fact their *methods* are not known to their enemy will protect them from being broken. This is a faulty assumption - if you reuse a method again and again (even with a different key each time) then eventually your adversaries will figure out what you are doing. And if Alice and Bob meet frequently in a secure location to decide on a new method, they might as well take the opportunity to exchange their secrets.. These considerations led Kerchoffs to state the following principle:

A cryptosystem should be secure even if everything about the system, except the key, is public knowledge. (Auguste Kerckhoffs, 1883)

(The actual quote is “Il faut qu’il n’exige pas le secret, et qu’il puisse sans inconvénient tomber entre les mains de l’ennemi” loosely translated as “The system must not require secrecy and can be stolen by the enemy without causing trouble”. According to Steve Bellovin the NSA version is “assume that the first copy of any device we make is shipped to the Kremlin”.)

Why is it OK to assume the key is secret and not the algorithm? Because we can always choose a fresh key. But of course if we choose our key to be “1234” or “passw0rd!” then that is not exactly secure. In fact, if you use any deterministic algorithm to choose the key then eventually your adversary will figure out. Therefore for security we must choose the key at *random*. Thus following can be thought of as a restatement of Kerchkoﬀs’s principle:

There is no secrecy without randomness

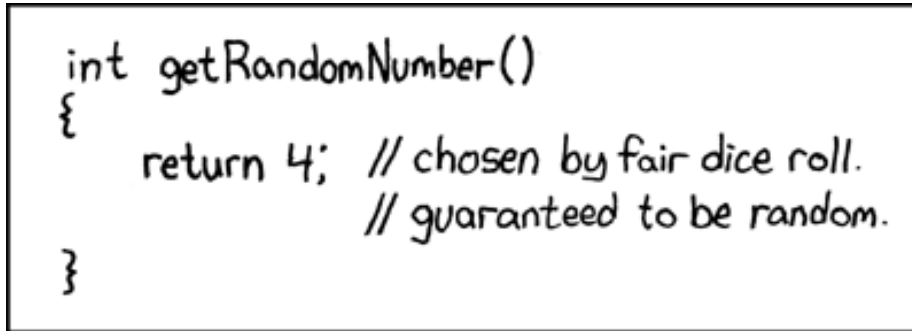
This is such a crucial point that is worth repeating:

There is no secrecy without randomness

At the heart of every cryptographic scheme there is a secret key, and the secret key is always chosen at random. A corollary of that is that to understand cryptography, you need to know some probability theory. Fortunately, we don't need much of probability- only probability over finite spaces, and basic notions such as expectation, variance, concentration and the union bound suffice for most of we need. In fact, understanding the following two statements will already get you much of what you need for cryptography:

- For every fixed string $x \in \{0, 1\}^n$, if you toss a coin n times, the probability that the heads/tails pattern will be exactly x is 2^{-n} .
- A probability of 2^{-128} is really really small.

The handout on mathematical background contains some of the probability and discrete mathematics that we'll need, and this will also be reviewed in the sections.



```
int getRandomNumber()
{
    return 4; // chosen by fair dice roll.
             // guaranteed to be random.
}
```

Figure 5: XKCD Cartoon: Random number generator

Note: Generating randomness in actual cryptographic systems

How do we actually get random bits in actual systems? The main idea is to use a two stage approach. First we need to get some data that is *unpredictable* from the point of view of an attacker on our system. Some sources for this could be measuring latency on the network or hard drives (getting harder with solid state disk), user keyboard and mouse movement patterns (problematic when you need fresh randomness at boot time), clock drift and more, there are some other sources including audio, video, and network. All of these can be problematic, especially for servers or virtual machines, and so hardware based random number generators based on phenomena such as thermal noise or nuclear decay are

becoming more popular. Once we have some data X that is unpredictable, we need to estimate the *entropy* in it. You can roughly imagine that X has k bits of entropy if the probability that an attacker can guess X is at most 2^{-k} . People then use a *hash function* (an object we'll talk about more later) to map X into a string of length k which is then hopefully distributed (close to) uniformly at random. All of this process, and especially understanding the amount of information an attacker may have on the entropy sources, is a bit of a dark art and indeed a number of attacks on cryptographic systems were actually enabled by weak generation of randomness. Here are a few examples.

One of the first attacks was on the SSL implementation of Netscape (*the* browser at the time). Netscape used the following “unpredictable” information—the time of day and a process ID both of which turned out to be quite predictable (who knew attackers have clocks too?). Netscape tried to protect its security through “security through obscurity” by not releasing the source code for its pseudorandom generator, but it was reverse engineered by Ian Goldberg and David Wagner (Ph.D students at the time) who demonstrated this attack.

In 2006 a programmer removed a line of code from the procedure to generate entropy in OpenSSL package distributed by Debian since it caused a warning in some automatic verification code. As a result for two years (until this was discovered) all the randomness generated by this procedure used only the process ID as an “unpredictable” source. This means that all communication done by users in that period is fairly easily breakable (and in particular, if some entities recorded that communication they could break it also retroactively). This caused a huge headache and a worldwide regeneration of keys, though it is believed that many of the weak keys are still used. See XKCD's take on that incidence.

In 2012 two separate teams of researchers scanned a large number of RSA keys on the web and found out that about 4% of them are easy to break. The main issue were devices such as routers, internet-connected printers and such. These devices sometimes run variants of Linux— a desktop operating system— but without a harddrive, mouse or keyboard, they don't have access to many of the entropy sources that desktop have. Coupled with some good old fashioned ignorance of cryptography and software bugs, this led to many keys that are downright trivial to break, see this blog post and this web page for more details.

After the entropy is collected and then “purified” or “extracted” to a uniformly random string that is, say, a few hundred bits long, we often need to “expand” it into a longer string that is also uniform (or at least looks like that for all practical purposes). We will discuss how to go about that in the next lecture. This step has its weaknesses too and in particular the Snowden documents, combined with observations of Shumow and Ferguson, strongly suggest that the NSA has deliberately inserted a *trapdoor* in one of the pseudorandom generators published by the National Institute of Standards and Technologies (NIST). Fortunately, this generator wasn't widely adapted but apparently the NSA did pay \$10M to RSA security so the latter would make this generator their default option in their products.

Defining the secrecy requirement.

Defining the secrecy requirement for an encryption is not simple. Over the course of history, many smart people got it wrong and convinced themselves that ciphers were impossible to break. The first person to truly ask the question in a rigorous way was Claude Shannon in 1949. Simply by asking this question, he made an enormous contribution to the science of cryptography and practical security. We now will try to examine how one might answer it. Let me warn you ahead of time that we are going to insist on a *mathematically precise definition* of security. That means that the definition must capture security in all cases, and the existence of a single counterexample, no matter how “silly”, would make us rule out a candidate definition. This exercise of coming up with “silly” counterexamples might seem, well, silly. But in fact it is this method that has led Shannon to formulate his theory of secrecy, which (after much followup work) eventually revolutionized cryptography, and brought this science to a new age where Poe’s maxim no longer holds, and we are able to design ciphers which human (or even nonhuman) ingenuity cannot break.

The most natural way to attack an encryption is for Eve to guess all possible keys. In many encryption schemes this number is enormous and this attack is completely infeasible. For example, the theoretical number of possibilities in the Enigma cipher was about 10^{113} which roughly means that even if we built a filled the milky way galaxy with computers operating at light speed, the sun would still die out before it finished examining all the possibilities.⁴ One can understand why the Germans thought it was impossible to break. (Note that despite the number of possibilities being so enormous, such a key can still be easily specified and shared between Alice and Bob by writing down 113 digits on a piece of paper.) Ray Miller from the NSA had calculated that, in the way the Germans used the machine, the number of possibilities was “only” 10^{23} which still would mean that it would take about a year to exhaust using the fastest supercomputer of 2015, at a time digital computers were not yet invented. Clearly, it is sometimes possible to break an encryption without trying all possibilities, and so having a huge number of key combinations does not guarantee security, as an attacker might find a shortcut (as the allies did) and recover the key without trying all options.

But perhaps we can simply define security as requiring the key to be unrecoverable except with tiny probability, no matter what method? Here is an attempt at such a definition:

Security Definition (First Attempt): An encryption scheme (E, D) is *n-secure* if no matter what method Eve employs, the probability that she can

⁴There are about 10^{68} atoms in the galaxy, so even if we assumed that each one of those atoms was a computer that can process say 10^{21} decryption attempts per second (as the speed of light is 10^9 meters per second and the diameter of an atom is about 10^{-12} meters), then it would still take $10^{113-89} = 10^{24}$ seconds, which is about 10^{17} years to exhaust all possibilities, while the sun is estimated to burn out in about 5 billion years.

recover the true key k from the ciphertext c is at most 2^{-n} .

You might wonder if this definition is not too strong to make sense, after all how are we going ever to prove that Eve cannot recover the secret key no matter what she does? Edgar Allan Poe would say that there can always be a method that we overlooked. However, in fact this definition is too *weak*! Consider the following encryption: the secret key k is chosen at random in $\{0, 1\}^n$ but our encryption scheme simply ignores it and lets $E_k(m) = m$ and $D_k(c) = c$. This is a valid encryption, but of course completely insecure as we are simply outputting the plaintext in the clear. Yet, no matter what Eve does, if she only sees c and not k , there is no way she can guess the true value of k with probability better than 2^{-n} , since it was chosen completely at random and she gets no information about it. Formally, one can prove the following result:

Theorem: Let (E, D) be the encryption scheme above. For every function $Eve : \{0, 1\}^\ell \rightarrow \{0, 1\}^n$ and for every $m \in \{0, 1\}^\ell$, the probability that $Eve(E_k(m)) = s$ is exactly 2^{-n} .

Proof: This follows because $E_k(m) = m$ and hence $Eve(E_k(m)) = Eve(m)$ which is some fixed value $k' \in \{0, 1\}^n$ that is independent of k . Hence the probability that $k = s'$ is 2^{-n} . QED

The math behind the above argument is very simple, yet I urge you to read and re-read the last two paragraphs until you are sure that you completely understand why this encryption is in fact secure according to the above definition. This is a “toy example” of the kind of reasoning that we will be employing constantly throughout this course, and you want to make sure that you follow it.

So, the above “Theorem” is true, but one might question its meaning. Clearly this silly example was not what we meant when stating this definition. However, as mentioned above, we are not willing to ignore even silly examples and must amend the definition to rule them out. One obvious objection is that we don’t care about hiding the key- it is the *message* that we are trying to keep secret. This suggests the next attempt:

Security Definition (Second Attempt): An encryption scheme (E, D) is *n-secure* if for every message m no matter what method Eve employs, the probability that she can recover m from the ciphertext $c = E_k(m)$ is at most 2^{-n} .

Now this seems like it captures our intended meaning. But remember that we are being anal, and truly insist that the definition holds as stated, namely that for every plaintext message m and every function $Eve : \{0, 1\}^\ell \rightarrow \{0, 1\}^n$, the probability over the choice of k that $Eve(E_k(m)) = m$ is at most 2^{-n} . But now we see that this is clearly impossible. After all, this is supposed to work for *every* message m and *every* function Eve , but clearly if m is that all-zeroes message 0^ℓ and Eve is the function that ignores its input and simply outputs 0^ℓ , then it will hold that $Eve(E_k(m)) = m$ with probability one.

So, if before the definition was too weak, the new definition is too strong and

is impossible to achieve. The problem is that of course we could guess a fixed message with probability one, so perhaps we could try to consider a definition with a *random* message. That is:

Security Definition (Third Attempt): An encryption scheme (E, D) is *n-secure* if no matter what method Eve employs, if m is chosen at random from $\{0, 1\}^\ell$, the probability that she can recover m from the ciphertext $c = E_k(m)$ is at most 2^{-n} .

This weakened definition can in fact be achieved, but we have again weakened it too much. Consider an encryption that hides the last $\ell/2$ bits of the message, but completely reveals the first $\ell/2$ bits. The probability of guessing a random message is $2^{-\ell/2}$, but this is still a scheme that would be completely insecure in practice. The point being that in practice we don't encrypt random messages—our messages might be in English, might have common headers, and might have even more structures based on the context. In fact, it may be that the message is either “Yes” or “No” (or perhaps either “Attack today” or “Attack tomorrow”) but we want to make sure Eve doesn't learn which one it is.

Perfect Secrecy

So far all of our attempts at definitions oscillated between being too strong (and hence impossible) or too weak (and hence not guaranteeing actual security). The key insight of Shannon was that in a secure encryption scheme the ciphertext should not reveal *any additional information* about the plaintext. So, if for example it was a priori possible for Eve to guess the plaintext with some probability $1/k$ (e.g., because there were only k possibilities for it) then she should not be able to guess it with higher probability after seeing the ciphertext. This is formalized as follows:

Security Definition (Perfect Secrecy): An encryption scheme (E, D) is *perfectly secret* if there for every set $M \subseteq \{0, 1\}^\ell$ of plaintexts, and for every strategy used by Eve, if we choose at random $m \in M$ and a random key $k \in \{0, 1\}^n$, then the probability that Eve guesses m after seeing $E_k(m)$ is at most $1/|M|$.

In particular, if we encrypt either “Yes” or “No” with probability $1/2$, then Eve won't be able to guess which one it is with probability better than half. In fact, that turns out to be the heart of the matter:

Two to Many Theorem: An encryption scheme (E, D) is perfectly secret if and only if for every two distinct plaintexts $\{m_0, m_1\} \subseteq \{0, 1\}^\ell$ and every strategy used by Eve, if we choose at random $b \in \{0, 1\}$ and a random key $k \in \{0, 1\}^n$, then the probability that Eve guesses m_b after seeing $E_k(m_b)$ is at most $1/2$.

Proof: The “only if” direction is obvious— this condition is a special case of the perfect secrecy condition for a set m of size 2.

The “if” direction is trickier. We need to show that if there is some set M (of size possibly much larger than 2) and some strategy for Eve to guess (based on the ciphertext) a plaintext chosen from M with probability larger than $1/|M|$, then there is also some set M' of size two and a strategy Eve' for Eve to guess a plaintext chosen from M' with probability larger than $1/2$.

Let's fix the message m_0 for example to be the all zeroes message. Since $Eve(E_k(m_0))$ is a fixed string, if we pick a random m_1 from M then it holds that

$$\Pr[Eve(E_k(m_0)) = m_1] \leq 1/|M|$$

while under our assumption, on average it holds that

$$\Pr[Eve(E_k(m_1)) = m_1] > 1/|M|$$

Thus in particular, due to linearity of expectation, there *exists* some m_1 satisfying

$$\Pr[Eve(E_k(m_1)) = m_1] > \Pr[Eve(E_k(m_0)) = m_1] .$$

But this can be turned into an attacker Eve' such that the probability that $Eve'(E_k(m_b)) = m_b$ is larger than $1/2$. Indeed, we can define $Eve'(c)$ to output m_1 if $Eve(c) = m_1$ and otherwise output a random message in $\{m_0, m_1\}$. The probability that $Eve'(c)$ equals m_1 is higher when $c = E_k(m_1)$ than when $c = E_k(m_0)$, and since Eve' outputs either m_0 or m_1 , this means that the probability that $Eve'(E_k(m_b)) = m_b$ is larger than $1/2$ (Can you see why?) QED.

Exercise: Another equivalent condition for perfect secrecy is the following: (E, D) is perfectly secret if for every plaintexts $m, m' \in \{0, 1\}^\ell$, the two random variables $\{E_k(m)\}$ and $\{E_{k'}(m')\}$ (for randomly chosen keys k and k') have precisely the same distribution.

So, perfect secrecy is a natural condition, and does not seem to be too weak for applications, but can it actually be achieved? After all, the condition that two different plaintexts are mapped to the same distribution seems somewhat at odds with the condition that Bob would succeed in decrypting the ciphertexts and find out if the plaintext was in fact m or m' . It turns out the answer is yes! For example, the table below details a perfectly secret encryption for two bits.

Table 1: A perfectly secret encryption scheme with 2 bit keys, plaintexts, and ciphertexts; the rows are indexed by possible ciphertexts, the columns indexed by possible plaintexts, and the (c, m) location of the matrix corresponds to the key that maps m to c .

Plain:	00	01	10	11
Cipher:				
00	00	01	10	11

01	01	00	11	10
10	10	11	00	01
11	11	10	01	00

In fact, this can be generalized to any number of bits:

Theorem (One time pad, Vernam 1917): For every n , there is a perfectly secret encryption (E, D) with plaintexts of n bits, where the key size and the ciphertext size is also n .

Proof: The encryption scheme is actually very simple - to encrypt a message $m \in \{0, 1\}^n$ with key $k \in \{0, 1\}^n$, we output $E_k(m) = m \oplus k$ where \oplus is the exclusive or (XOR) operation. That is, $m \oplus k$ is a vector in $\{0, 1\}^n$ such that $(m \oplus k)_i = k_i + m_i \pmod{2}$. Decryption works identically - $D_k(c) = c \oplus k$. It is not hard to use the associativity of addition (and in particular XOR) to verify that $D_k(E_k(m)) = (m \oplus k) \oplus k = m \oplus (k \oplus k) = m$ where the last equality follows from $k \oplus k = 0^n$ (can you see why?). Now we claim that for every message $m \in \{0, 1\}^n$, the distribution $E_k(m)$ for a random k is the uniform distribution U_n on $\{0, 1\}^n$. By the exercise above, this implies that the scheme is perfectly secret, since for every two messages m, m' the distributions $\{E_k(m)\}$ and $\{E_k(m')\}$ will both be equal to the uniform distribution. To prove the claim we need to show that for every $y \in \{0, 1\}^n$, $\Pr[E_k(m) = y] = 2^{-n}$ where this probability is taken over the choice of a random $k \in \{0, 1\}^n$. Now note that $E_k(m) = y$ if and only if $m \oplus k = y$ or, equivalently, $k = m \oplus y$. Since k is chosen uniformly at random in $\{0, 1\}^n$, the probability that it equals $m \oplus y$ is exactly 2^{-n} QED.

Note: Importance of using the one time pad only once:

The “one time pad” is a name analogous to the “point away from yourself gun”- the name suggests the fatal mistake people often end up doing. Perhaps the most dramatic example of the dangers of “key reuse” is the *Venona Project*. The Soviets have used the one-time pad for their confidential communication since before the 1940’s (WHEN?), and in fact even before Shannon apparently the U.S. intelligence already knew that it is in principle “unbreakable” in 1941 (see page 32 in the Venona document)). However, it turned out that the hassles of manufacturing so many keys for all the communication took its toll on the Soviets and they ended up reusing the same

keys for more than one message, though they tried to use them for completely different receivers in the (false) hope that this wouldn't be detected. The Venona project of the U.S. Army was founded in February 1943 by Gene Grabeel- a former highschool teacher from Madison Heights, Virginia and Lt. Leonard Zukbo. In October 1943, they had their breakthrough when it was discovered that the Russians are reusing their keys (credit to this discovery is shared by Lt. Richard Hallock, Carrie Berry, Frank Lewis, and Lt. Karl Elmquist, see page 27 in the document). In the 37 years of its existence, the project has resulted in a treasure chest of intelligence, exposing hundreds of KGB agents and Russian spies in the U.S. and other countries, including Julius Rosenberg, Harry Gold, Klaus Fuchs, Albert Hiss, Harry Dexter White and many others.

Necessity of long keys

The one time pad requires a key the size of the message, which means that if you plan to communicate with x people, you are going to have to maintain (securely!) x huge files that are each as long as the length of the maximum total communication you expect with that person. Imagine that every time you opened an account with Amazon, Google, or any other service, they would need to send you in the mail a DVD full of random numbers, and every time you suspected a virus, you'll need to ask all these services for a fresh DVD. This doesn't sound so appealing. Ideally, one could think that Alice and Bob only share a key that is long enough to be unguessable, e.g., 128 bits, and use that for all their communication. Unfortunately this is impossible to achieve with perfect secrecy:

Theorem: If E is a perfectly secret system with key of length n and messages of length ℓ then $\ell \leq n$.

Proof: Suppose, towards the sake of contradiction that there was a perfectly secret system (E, D) with a key of length n but messages of length $\ell > n$. Then consider the following adversary strategy for Eve: given a ciphertext c , guess a random key $k \in \{0, 1\}^n$ and output $m = D_k(c)$. The probability that Eve is successful is at least 2^{-n} , since with this probability she guesses the key correctly. But by perfect secrecy, if the message is chosen at random, she should have been successful with probability at most $2^{-\ell} < 2^{-n}$.

This proof might not be fully convincing - after all, an attack that succeeds with probability 2^{-n} is not very worrying. But this violation of the security definition can be significantly boosted:

Theorem: If E is an encryption with key of length n and messages of length $\geq n + 10$ then there exist two messages m_0, m_1 and a strategy for Eve so that given an encryption $c = E_k(m_b)$ for random k and $b \in \{0, 1\}$, Eve can output m_b with probability at least 0.99.

Proof: Suppose that we choose two messages m_0, m_1 at random, encrypt m_1 to obtain a ciphertext c_1 and ask what is the probability that there exists some key k such that $c_1 = E_k(m_0)$. Now, let's fix the choice of m_0 and so consider the set $C_0 = \{E_k(m_0) : k \in \{0, 1\}^n\}$. The size of this set is at most 2^n . Now for every choice of the key k , the map $m_1 \mapsto E_k(m_1)$ is one to one and so the image of this map is some set D_k of size 2^{n+10} (i.e., there are exactly 2^{n+10} ciphertexts that are the encryption under k of some $m_1 \in \{0, 1\}^{n+10}$). If we pick m_1 at random then $c_1 = E_k(m_1)$ is chosen at random from the set D_k and hence the probability that c_1 falls into C_0 is at most $|C_0|/|D_k| \leq 2^{-10} < 0.01$. Hence in particular, there must be *some* choice of m_0, m_1 such that Eve decides given c to output m_0 if $c \in C_0$ and output m_1 otherwise, then she will be successful with probability at least 0.99. QED

Note: The above proof is short but subtle. I suggest you try to read it *very* carefully and make sure you understand it, since it is a prototype for future probabilistic arguments that we will be making regularly. It might help for you to consider a “baby case” when there are, say, 10 possible messages and 4 possible keys, and try to prove in this case that you can always find a pair of messages m_0, m_1 such that you can tell with probability at least 60% whether an encryption was of m_0 or of m_1 .

Advanced comment: Adding probability into the picture

There is a sense in which both our secrecy and our impossibility results might not be fully convincing, and that is that we did not explicitly consider algorithms that use *randomness*. For example, maybe Eve can break a perfectly secret encryption if she is not modeled as a deterministic function $Eve : \{0, 1\}^o \rightarrow \{0, 1\}^\ell$ but rather a *probabilistic* process. Similarly, maybe the encryption and decryption functions as well could be probabilistic processes as well. It turns out that none of those matter. For the former, note that a probabilistic process can be thought of as a *distribution* over functions, in the sense that we have a collection of functions f_1, \dots, f_N mapping $\{0, 1\}^o$ to $\{0, 1\}^\ell$, and some probabilities p_1, \dots, p_N (non-negative numbers summing to 1), so we now think of Eve as selecting the function f_i with probability p_i . But if none of those functions can give advantage better than $1/2$, then neither can this collection. A similar (though more involved) argument shows that the impossibility result showing that the key must be at least as long as the message still holds even if the encryption and decryption algorithms are allowed to be probabilistic processes as well (working this out is a great exercise).

~ MathDefs ~

CS 127: Cryptography / Boaz Barak

Additional reading: Chapter 3 up to and including Section 3.3 in Katz Lindell book.

Recall our cast of characters- Alice and Bob want to communicate securely over a channel that is monitored by the nosy Eve. In the last lecture, we have seen the definition of *perfect secrecy* that guarantees that Eve cannot learn *anything* about their communication beyond what she already knew. However, this security came at a price. For every bit of communication, Alice and Bob have to exchange in advance a bit of a secret key. In fact, the proof of this result gives rise to the following simple Python program that can break every encryption scheme that uses, say, a 128 bit key, with a 129 bit message:

```
# Gets ciphertext as input and two potential plaintexts
# Positive return value means first is more likely,
# negative means second is more likely,
# 0 means both have same likelihood.
#
# We assume we have access to the function Decrypt(key,ciphertext)
def Distinguish(ciphertext,plaintext1,plaintext2):
    bias = 0
    key = [0,0,...,0] #128 0's
    while(sum(key)<128):
        p = Decrypt(key,ciphertext)
        if p==plaintext1: bias++
        if p==plaintext2: bias--
        increment(key)
    return bias

# increment key when thought of as a number sorted from least significant
# to most significant bit. Assume not all bits are 1.
def increment(key):
    i = key.index(0);
    for j in range(i-1): key[j]=0
    key[i]=1
```

Now, generating, distributing, and protecting huge keys causes immense logistical problems, which is why almost all encryption schemes used in practice do in fact utilize short keys (e.g., 128 bits long) with messages that can much longer (sometimes even terabytes or more of data).

So, why can't we use the above Python program to break all encryptions in the Internet and win infamy and fortune? We can in fact, but we'll have to wait a *really* long time, since the loop in `Distinguish` will run 2^{128} times, which will take much more than the lifetime of the universe to complete, even if we used all the computers on the planet.

However, the fact that *this* particular program is not a feasible attack, does not mean there does not exist a different attack. But this still suggests a tantalizing possibility: if we consider a relaxed version of perfect secrecy that restricts Eve to performing computations that can be done in this universe (e.g., less than 2^{256} steps should be safe not just for human but for all potential alien civilizations) then can we bypass the impossibility result and allow the key to be much shorter than the message?

This in fact does seem to be the case, but as we've seen, defining security is a subtle task, and will take some care. As before, the way we avoid (at least some of) the pitfalls of so many cryptosystems in history is that we insist on very precisely *defining* what it means for a scheme to be secure.

Let us defer the discussion how one defines a function being computable in “less than T operations” and just say that there is a way to formally do so. Given the perfect secrecy definition we saw last time, a natural attempt for defining computational security would be the following:

Security Definition (First Attempt): An encryption scheme (E, D) is *computationally n -secure* if for every two distinct plaintexts $\{m_0, m_1\} \subseteq \{0, 1\}^\ell$ and every strategy of Eve using at most 2^n computational steps, if we choose at random $b \in \{0, 1\}$ and a random key $k \in \{0, 1\}^n$, then the probability that Eve guesses m_b after seeing $E_k(m_b)$ is at most $1/2$.

This seems a natural definition, but is in fact impossible to achieve if the key is shorter than the message. The reason is that if the message is even one bit longer we can always have a very efficient procedure that achieves success probability of about $1/2 + 2^{-n-1}$ by guessing the key. (I.e., replace the loop in **Distinguish** by choosing the key at random.)

However, such tiny advantage does not seem very useful, and hence our actual definition will be the following:

Security Definition (Computational Security): An encryption scheme (E, D) is *computationally n -secure* if for every two distinct plaintexts $\{m_0, m_1\} \subseteq \{0, 1\}^\ell$ and every strategy of Eve using at most 2^n computational steps, if we choose at random $b \in \{0, 1\}$ and a random key $k \in \{0, 1\}^n$, then the probability that Eve guesses m_b after seeing $E_k(m_b)$ is at most $1/2 + 2^{-n}$.

Having learned our lesson, let's try to see that this strategy does give us the kind of conditions we desired. In particular, let's verify that this definition implies the analogous condition to perfect secrecy.

Theorem: If (E, D) is computationally n -secure then every subset $M \subseteq \{0, 1\}^\ell$ and every strategy of Eve using at most $2^n - (100\ell + 100)$ computational steps, if we choose at random $m \in M$ and a random key $k \in \{0, 1\}^n$, then the probability that Eve guesses m after seeing $E_k(m)$ is at most $1/|M| + 2^{-n+1}$.

Before proving this theorem note that it gives us a pretty strong guarantee. In the exercises we will strengthen it even further showing that no matter what prior

information Eve had on the message before, she will never get any non-negligible new information on it. One way to phrase it is that if your attacker used a 256-bit secure encryption to encrypt a message, then your chances of getting to learn any additional information about it before the universe collapses are more or less the same as the chances that a fairy will materialize and whisper it in your ear.

Proof: The proof is rather similar to the equivalence of guessing one of two messages vs. one of many messages for perfect secrecy. However, in the computational context we need to be careful keeping track of Eve’s running time. In that proof we showed that if there exists:

- A subset $M \subseteq \{0, 1\}^\ell$ of messages

and

- An adversary $Eve : \{0, 1\}^o \rightarrow \{0, 1\}^\ell$ such that

$$\Pr_{m \leftarrow_R M, k \leftarrow_R \{0, 1\}^n} [Eve(E_k(m)) = m] > 1/|M|$$

Then there exist two messages m_0, m_1 and an adversary $Eve' : \{0, 1\}^o \rightarrow \{0, 1\}^\ell$ such that $\Pr_{b \leftarrow_R \{0, 1\}, k \leftarrow_R \{0, 1\}^n} [Eve'(E_k(m_b)) = m_b] > 1/2$.

To adapt this proof to the computational setting and complete the proof of the current theorem we need to:

- Show that if the probability of Eve succeeding was $\frac{1}{|M|} + \epsilon$ then the probability of Eve' succeeding is at least $\frac{1}{2} + \epsilon/2$.
- Show that if Eve can be computed in T operations, then Eve' can be computed in $T + 100\ell + 100$ operations.

The first point can be shown by simply doing the same proof more carefully, keeping track how the advantage over $\frac{1}{|M|}$ for Eve translates into an advantage over $\frac{1}{2}$ for Eve' . The second point is obtained by looking at the definition of Eve' from that proof. On input c , Eve' computed $m = Eve(c)$ (which costs T operations) and then checked if $m = m_0$ (which costs, say, at most 5ℓ operations), and then output either 1 or a random bit (which is a constant, say at most 100 operations). QED

Note: The proof of this theorem is a model to how a great many of the results in this course will look like. Generally we will have many theorems of the form:

“If there is a scheme S' satisfying security definition X' then there is a scheme S satisfying security definition X ”

In this case X' was “computational n -security” (hardness of distinguishing between encryptions of two ciphertexts) and X was the more general notion of hardness of getting a non-trivial advantage over guessing for an encryption of

a random $m \in M$. Also here the scheme S was the same as S' , but generally that need not always be the case. All of these proofs will have the same global structure— we will assume towards a contradiction, that there is an efficient adversary strategy Eve demonstrating that S violates X , and build from Eve a strategy Eve' demonstrating that S' violates X . This is such an important point that it deserves repeating:

The way you show that if S' is secure then S is secure is by giving a transformation from an adversary that breaks S into an adversary that breaks S'

For computational security, we will always want that Eve' will be efficient if Eve is, and that will usually be the case because Eve' will simply use Eve as a black box, which it will not invoke too many times, and addition will use some polynomial time preprocessing and postprocessing. The more challenging parts of such proofs are typically:

- Coming up with the strategy Eve' .
- Analyzing the probability of success and in particular showing that if Eve had non-negligible advantage then so will Eve' .

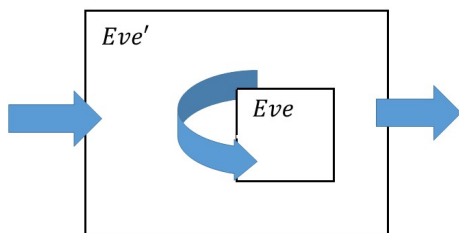


Figure 1: We show that the security of S' implies the security of S by transforming an adversary Eve breaking S into an adversary Eve' breaking S'

The asymptotic approach

For practical security, often every bit of security matters. We want our keys to be as short as possible and our schemes to be as fast as possible while satisfying a particular level of security. However, for understanding the *principles* behind encryption, keeping track of those bits can be a distraction, and so just like we do for algorithms, we will use *asymptotic analysis* (also known as *big Oh notation*) to sweep many of those details under the carpet.

To a first approximation, there will be only two types of running times we will encounter in this course:

- *Polynomial* running time of the form $d \cdot n^c$ for some constants $d, c > 0$ (or $\text{poly}(n) = n^{O(1)}$ for short) , which we will consider as *efficient*
- *Exponential* running time of the form $2^{d \cdot n^\epsilon}$ for some constants $d, \epsilon > 0$ (or $2^{n^{\Omega(1)}}$ for short) which we will consider as *infeasible*.¹

Similarly, we will consider probabilities of the form $1/\text{poly}(n)$ as *noticeable* while probabilities of the form $2^{-n^{\Omega(1)}}$ will be called *negligible* .

These are not all the theoretically possible running times. One can have intermediate functions such as $n^{\log n}$ though we will generally not encounter those. To make things clean (and to correspond to standard terminology), we will say that an algorithm A is *efficient* if it runs in time $\text{poly}(n)$ when n is its input length (which will always be the same, up to polynomial factors, as the key length). If $\mu(n)$ is some probability that depends on the input/key length parameter n , then we say that $\mu(n)$ is *negligible* if it's smaller than every polynomial. That is, for every c, d there is some N , such that if $n > N$ then $\mu(n) < 1/(cn)^d$. Note that for every non-constant polynomials p, q , $\mu(n)$ is negligible if and only if the function $\mu'(n) = p(\mu(q(n)))$ is negligible.

From now on, we will require all of our encryption schemes to be *efficient* which means that the encryption and decryption algorithms should run in polynomial time. Security will mean that any efficient adversary can make at most a negligible gain in the probability of guessing the message over its a priori probability. That is, we make the following definition:

Security Definition (Computational Security): An encryption scheme (E, D) is *computationally secure* if for every two distinct plaintexts $\{m_0, m_1\} \subseteq \{0, 1\}^\ell$ and every efficient strategy of Eve, if we choose at random $b \in \{0, 1\}$ and a random key $k \in \{0, 1\}^n$, then the probability that Eve guesses m_b after seeing $E_k(m_b)$ is at most $1/2 + \mu(n)$ for some negligible function $\mu(\cdot)$.

Counting number of operations.

One more detail that we've so far ignored is what does it mean exactly for a function to be computable using at most T operations. Fortunately, when we don't really care about the difference between T and, say, T^2 , then essentially every reasonable definition gives the same answer. Formally, we can use the notions of Turing machines or Boolean circuits to define complexity. For concreteness, let's define that a function $F : \{0, 1\}^n \rightarrow \{0, 1\}^m$ has complexity there exists a Boolean circuit (that uses the AND, OR and NOT gates) with at most T gates that computes F . We will often also consider *probabilistic* functions in which case we allow the circuit a RAND gate that outputs a single random

¹Some texts reserve the term *exponential* to running times of the form $2^{\epsilon n}$ for some $\epsilon > 0$ and call running time of , say, $2^{\sqrt{n}}$ *subexponential* . However, we will generally not make this distinction in this course.

bits. For more on circuit complexity you can take a look at Chapter 6 of my computational complexity textbook with Arora (see also draft on the web).

The fact that we only care about asymptotics means you don't really need to think of gates, etc.. when arguing in cryptography. However, it is comforting to know that this notion has a precise mathematical formulation. See the appendix below for a more precise formulation of this and some discussion.

Our first conjecture

We are now ready to make our first conjecture:

The Cipher Conjecture:² There exists a computationally secure encryption scheme (E, D) (where E, D are efficient) with a key of size n for messages of size $n + 1$.

A *conjecture* is a well defined mathematical statement which (1) we believe is true but (2) don't know yet how to prove. Proving the cipher conjecture will be a great achievement and would in particular settle the P vs NP question, which is arguably *the* fundamental question of computer science. That is, the following is known to be a theorem (feel free to ignore it if you don't know the definition of P and NP, though if it piques your curiosity, you can find more about it by reading the first two chapters of my book with Arora):

Theorem: If $P = NP$ then there does not exist a computationally secure encryption with efficient E and D and where the message is longer than the key.

Proof idea: If $P = NP$ then whenever we have a loop that searches through some domain to find some string that satisfies a particular property (like the loop in the **Distinguish** subroutine above that searches over all keys) then this loop can be sped up *exponentially* .

While it is very widely believed that $P \neq NP$, at the moment we do not know how to *prove* this, and so have to settle for accepting the cipher conjecture as essentially an axiom, though we will see later in this course that we can show it follows from some seemingly weaker conjectures.

There are several reasons to believe the cipher conjecture. We now briefly mention some of them:

- *Intuition:* If the cipher conjecture is false then it means that for *every* possible cipher we can make the exponential time attack described above become efficient. It seems “too good to be true” in a similar way that the assumption that $P=NP$ seems too good to be true.

²As will be the case for other conjectures we talk about, the name “The Cipher Conjecture” is not a standard name, but rather one we'll use in this course. In the literature this conjecture is mostly referred to as the conjecture of existence of *one way functions*, a notion we will learn about later. These two conjectures a priori seem quite different but have been shown to be equivalent.

- *Concrete candidates:* As we will see in the next lecture, there are several concrete candidate ciphers using keys shorter than messages for which despite *tons* of effort, no one knows how to break them. Some of them are widely used and hence governments and other benign or not so benign organizations have every reason to invest huge resources in trying to break them. Despite that as far as we know (and we know a little more after Snowden) there is no significant break known for the most popular ciphers. Moreover, there are other ciphers that can be based on canonical mathematical problems such as factoring large integers or decoding random linear codes that are immensely interesting in their own right independently of their cryptographic applications.
- *Minimalism:* Clearly if the cipher conjecture is false then we also don't have a secure encryption with a key, say, twice as long as the message. But it turns out the cipher conjecture is in fact *necessary* for essentially every cryptographic primitive, including not just private key and public key encryptions but also digital signatures, hash functions, pseudorandom generators, and more. That is, if the cipher conjecture is false then to a large extent cryptography does not exist, and so we essentially have to assume it if we want to do any kind of cryptography.

Why care about the cipher conjecture?

“Give me a place to stand, and I shall move the world” Archimedes,
circa 250 BC

Every perfectly secure encryption scheme is clearly also computationally secure, and so if required a message of size n instead $n + 1$ then the conjecture would have been trivially satisfied by the one-time pad. However, having a message longer than the key by just a single bit does not seem that impressive. Sure, if we used such a scheme with 128-bit long keys, our communication will be smaller by a factor of 128/129 (or a saving of about 0.8%) over the one-time pad, but this doesn't seem worth the risk of using an unproven conjecture. However, it turns out that if we assume this rather weak condition, we can actually get a computationally secure encryption scheme with a message of size $p(n)$ for *every* polynomial $p(\cdot)$. In essence, we can fix a single n -bit long key and communicate securely as many bits as we want!

Moreover, this is just the beginning. There is a huge range of other useful cryptographic tools that we can obtain from this seemingly innocent conjecture: (We will see what all these names and some of these reductions mean later in the course.)

We will soon see the first of the many reductions we'll learn in this course. Together this “web of reductions” forms the scientific core of cryptography, connecting many of the core concepts and enabling us to construct increasingly

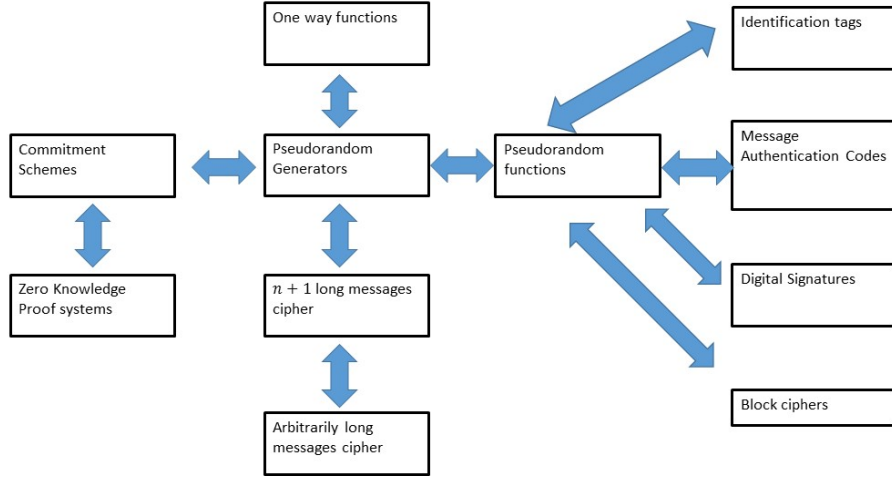


Figure 2: Web of reductions between notions equivalent to ciphers with larger than key messages

sophisticated tools based on relatively simple “axioms” such as the cipher conjecture.

Prelude: Computational Indistinguishability

The task of Eve in breaking an encryption scheme is to *distinguish* between an encryption of m_0 and an encryption of m_1 . It turns out to be useful to consider this question of when two distributions are *computationally indistinguishable* more broadly:

Definition (Computational Indistinguishability): Let X and Y be two distributions over $\{0,1\}^o$. We say that X and Y are (T, ϵ) -computationally indistinguishable, denoted by $X \approx_{T, \epsilon} Y$, if for every function Eve computable with at most T operations,

$$|\Pr[Eve(X) = 1] - \Pr[Eve(Y) = 1]| \leq \epsilon .$$

We say that X and Y are simply *computationally indistinguishable*, denoted by $X \approx Y$, if they are (T, ϵ) indistinguishable for every polynomial $T(o)$ and inverse polynomial $\epsilon(o)$.³

³This definition implicitly assumes that X and Y are actually *parameterized* by some number n (that is polynomially related to o) so for every polynomial $T(o)$ and inverse polynomial $\epsilon(o)$ we can take n to be large enough so that X and Y will be (T, ϵ) indistinguishable. In all the cases we will consider, the choice of the parameter n (which is usually the length of the key) will be clear from the context.

Note: The expression $\Pr[Eve(X) = 1]$ can also be written as $\mathbb{E}[Eve(X)]$ (since we can assume that whenever $Eve(x)$ does not output 1 it outputs zero). This notation will be useful for us sometimes.

We can use computational indistinguishability to phrase the definition of computational security more succinctly:

Theorem (C.I. phrasing of computational security): Let (E, D) be a valid encryption scheme. Then (E, D) is computationally secure if and only if for every two messages $m_0, m_1 \in \{0, 1\}^\ell$,

$$\{E_k(m_0)\} \approx \{E_k(m_1)\}$$

where each of these two distributions is obtained by sampling a random $k \leftarrow_R \{0, 1\}^n$.

The proof is left as an Exercise in Homework 1.

One intuition for computational indistinguishability that it is related to some notion of *distance*. If two distributions are computationally indistinguishable, then we can think of them as “very close” to one another, at least as far as efficient observers are concerned. Intuitively, if X is close to Y and Y is close to Z then X should be close to Z . Similarly if four distributions X, X', Y, Y' satisfy that X is close to Y and X' is close to Y' , then you might expect that the distribution (X, X') where we take two independent samples from X and X' respectively, is close to the distribution (Y, Y') where we take two independent samples from Y and Y' respectively. We will now verify that these intuitions are in fact correct:

Lemma (Triangle Inequality for Computational Indistinguishability):⁴ Suppose $\{X_1\} \approx_{T, \epsilon} \{X_2\} \approx_{T, \epsilon} \dots \approx_{T, \epsilon} \{X_m\}$. Then $\{X_1\} \approx_{T, (m-1)\epsilon} \{X_m\}$, where o is the length of the X_i 's.

Proof: Suppose that there exists a T time Eve such that

$$|\Pr[Eve(X_1) = 1] - \Pr[Eve(X_m) = 1]| > (m - 1)\epsilon .$$

Write

$$\Pr[Eve(X_1) = 1] - \Pr[Eve(X_m) = 1] = \sum_{i=1}^{m-1} (\Pr[Eve(X_i) = 1] - \Pr[Eve(X_{i+1}) = 1]) .$$

⁴Results of this form are known as “triangle inequalities” since they can be viewed as generalizations of the statement that for every three points on the plane x, y, z , the distance from x to z is not larger than the distance from x to y plus the distance from y to z . In other words, the edge $\overline{x, z}$ of the triangle (x, y, z) is not longer than the sum of the lengths of the other two edges $\overline{x, y}$ and $\overline{y, z}$.

Thus,

$$\sum_{i=1}^{m-1} |\Pr[Eve(X_i) = 1] - \Pr[Eve(X_{i+1}) = 1]| > (m-1)\epsilon$$

and hence in particular there must exist some $i \in \{1, \dots, m-1\}$ such that

$$|\Pr[Eve(X_i) = 1] - \Pr[Eve(X_{i+1}) = 1]| > \epsilon$$

contradicting the assumption that $\{X_i\} \approx_{T,\epsilon} \{X_{i+1}\}$ for all $i \in \{1, \dots, m-1\}$.
QED

Lemma (Computational Indistinguishability is preserved under repetition): Suppose that $X_1, \dots, X_\ell, Y_1, \dots, Y_\ell$ are distributions over $\{0, 1\}^n$ such that $X_i \approx_{T,\epsilon} Y_i$. Then $(X_1, \dots, X_\ell) \approx_{T-10\ell n, \ell\epsilon} (Y_1, \dots, Y_\ell)$.

Proof: For every $i \in \{0, \dots, \ell\}$ we define H_i to be the distribution $(X_1, \dots, X_i, Y_{i+1}, \dots, Y_\ell)$. Clearly $H_0 = (X_1, \dots, X_\ell)$ and $H_\ell = (Y_1, \dots, Y_\ell)$. We will prove that for every i , $H_i \approx_{T-10\ell n, \epsilon} H_{i+1}$, and the proof will then follow from the triangle inequality (can you see why?). Indeed, suppose towards the sake of contradiction that there was some $i \in \{0, \dots, \ell\}$ and some $T-10\ell n$ -time Eve' 's : $\{0, 1\}^{n\ell} \rightarrow \{0, 1\}$ such that

$$|\mathbb{E}[Eve'(H_i)] - \mathbb{E}[Eve(H_{i+1})]| > \epsilon.$$

In other words

$$|\mathbb{E}_{X_1, \dots, X_{i-1}, Y_i, \dots, Y_\ell}[Eve'(X_1, \dots, X_{i-1}, Y_i, \dots, Y_\ell)] - \mathbb{E}_{X_1, \dots, X_i, Y_{i+1}, \dots, Y_\ell}[Eve'(X_1, \dots, X_i, Y_{i+1}, \dots, Y_\ell)]| > \epsilon.$$

By linearity of expectation we can write the difference of these two expectations as

$$\mathbb{E}_{X_1, \dots, X_{i-1}, X_i, Y_i, Y_{i+1}, \dots, Y_\ell}[Eve'(X_1, \dots, X_{i-1}, Y_i, Y_{i+1}, \dots, Y_\ell) - Eve'(X_1, \dots, X_{i-1}, X_i, Y_{i+1}, \dots, Y_\ell)]$$

. By the *averaging principle*⁵ this means that there exist some values $x_1, \dots, x_{i-1}, y_{i+1}, \dots, y_\ell$ such that

$$|\mathbb{E}_{X_i, Y_i}[Eve'(x_1, \dots, x_{i-1}, Y_i, y_{i+1}, \dots, y_\ell) - Eve'(x_1, \dots, x_{i-1}, X_i, y_{i+1}, \dots, y_\ell)]| > \epsilon$$

⁵This is the principle that if the average grade in an exam was at least α then *someone* must have gotten at least α , or in other words that if a real-valued random variable Z satisfies $\mathbb{E}Z \geq \alpha$ then $\Pr[Z \geq \alpha] > 0$.

Now X_i and Y_i are simply independent draws from the distributions X and Y respectively, and so if we define $Eve(z) = Eve'(x_1, \dots, x_{i-1}, z, y_{i+1}, \dots, y_\ell)$ then Eve runs in time at most the running time of Eve' plus $2\ell n$ and it satisfies

$$|\mathbb{E}_{X_i}[Eve(X_i)] - \mathbb{E}_{Y_i}[Eve(Y_i)]| > \epsilon$$

contradicting the assumption that $X_i \approx_{T, \epsilon} Y_i$. QED

Note: The above proof illustrates a powerful technique known as the *hybrid argument* whereby we show that two distributions C^0 and C^1 are close to each other by coming up with a sequence of distributions H_0, \dots, H_t such that $H_t = C^1$, $H_0 = C^0$, and we can argue that H_i is close to H_{i+1} for all i . This type of argument repeats itself time and again in cryptography, and so it is important to get comfortable with it.

The Length Extension Theorem

Extension via repetition

We now turn to showing the length extension theorem. For a warm-up, let's show that we can actually repeat encryptions to get an $n/(n+1)$ saving.

Theorem (security of repetition): Suppose that (E', D') is a computationally secure encryption scheme with n bit keys and $n+1$ bit messages. Then the scheme (E, D) where $E_{k_1, \dots, k_t}(m_1, \dots, m_t) = (E'_{k_1}(m_1), \dots, E'_{k_t}(m_t))$ and $D_{k_1, \dots, k_t}(c_1, \dots, c_t) = (D'_{k_1}(c_1), \dots, D'_{k_t}(c_t))$ is a computationally secure scheme with tn bit keys and $t(n+1)$ bit messages.

Proof: This might seem “obvious” but in cryptography, even obvious facts are sometimes wrong, so it's important to prove this formally. Luckily, this is a fairly straightforward implication of the fact that computational indistinguishability is preserved under many samples. That is, by the security of (E', D') we know that for every two messages $m, m' \in \{0, 1\}^{n+1}$, $E_k(m) \approx E_k(m')$ where k is chosen from the distribution U_n . Therefore by the indistinguishability of many samples lemma, for every two tuples $m_1, \dots, m_t \in \{0, 1\}^{n+1}$ and $m'_1, \dots, m'_t \in \{0, 1\}^{n+1}$,

$$(E'_{k_1}(m_1), \dots, E'_{k_t}(m_t)) \approx (E'_{k_1}(m'_1), \dots, E'_{k_t}(m'_t))$$

for random k_1, \dots, k_t chosen independently from U_n which is exactly the condition that (E, D) is computationally secure. QED

Theorem (Length Extension of ciphers): Suppose that there exists a computationally secure encryption scheme (E', D') with key length n and message length $n+1$. Then for every polynomial $t(n)$ there exists a computationally secure encryption scheme (E, D) with key length n and message length $t(n)$.

Proof: Let $t = t(n)$. We are given a cipher E' which can encrypt $n + 1$ -bit long messages with an n -bit long key and we need to encrypt a t -bit long message $m = (m_1, \dots, m_t) \in \{0, 1\}^t$. Our idea is simple (at least in hindsight). Let $k_0 \leftarrow_R \{0, 1\}^n$ be our key (which is chosen at random). To encrypt m using k_0 , the encryption function will choose t random strings $k_1, \dots, k_t \leftarrow_R \{0, 1\}^n$.⁶ We will then encrypt the $n + 1$ -bit long message (k_1, m_1) with the key k_0 to obtain the ciphertext c_1 , then encrypt the $n + 1$ -bit long message (k_2, m_2) with the key k_1 to obtain the ciphertext c_2 , and so on and so forth until we encrypt the message (k_t, m_t) with the key k_{t-1} . We output (c_1, \dots, c_t) as the final ciphertext.⁷

To decrypt (c_1, \dots, c_t) using the key k_0 , first decrypt c_1 to learn (k_1, m_1) , then use k_1 to decrypt c_2 to learn (k_2, m_2) , and so on until we use k_{t-1} to decrypt c_t and learn (k_t, m_t) . Finally we can simply output (m_1, \dots, m_t) .

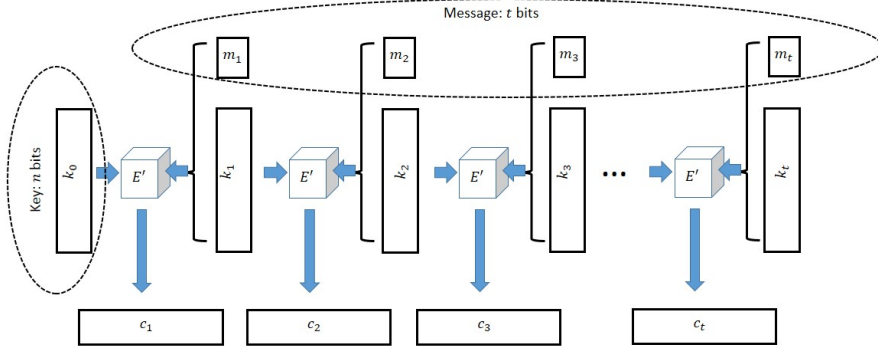


Figure 3: Constructing a cipher with t bit long messages from one with $n + 1$ long messages

The above are clearly valid encryption and decryption algorithms, and hence the real question becomes *is it secure??*. The intuition is that c_1 hides all information about (k_1, m_1) and so in particular the first bit of the message is encrypted securely, and k_1 still can be treated as an unknown random string even to an adversary that saw c_1 . Thus, we can think of k_1 as a random secret key for the encryption c_2 , and hence the second bit of the message is encrypted securely, and so on and so forth.

The above looks like a reasonable intuitive argument, but to make sure it's true we need to give an actual proof. Let $m, m' \in \{0, 1\}^t$ be two messages. We need to show that $E_{U_n}(m) \approx E_{U_n}(m')$. The heart of the proof will be the following claim:

⁶Note that this makes the encryption function *probabilistic* but it does not increase the size of the key; while we didn't explicitly say that the encryption can be probabilistic before, allowing this is absolutely fine and in fact will be *necessary* for some future security requirements.

⁷The astute reader might note that the key k_t is actually not used anywhere in the encryption nor decryption and hence we could encrypt n more bits of the message instead in this final round. We used the current description for the sake of symmetry and simplicity of exposition.

Claim: Let \hat{E} be the algorithm that on input a message m and key k_0 works like E except that its the i^{th} block contains $E'_{k_{i-1}}(k'_i, m_i)$ where k'_i is a *random* string in $\{0, 1\}^n$, that is chosen *independently* of everything else including the key k_i . Then, for every message $m \in \{0, 1\}^t$

$$E_{U_n}(m) \approx \hat{E}_{U_n}(m) .$$

Note that \hat{E} is not a valid encryption scheme since it's not at all clear there is a decryption algorithm for it. It is just an hypothetical tool we use for the proof. Once we prove the claim then we are done since we know that for every pair of message m, m' , $E_{U_n}(m) \approx \hat{E}_{U_n}(m)$ and $E_{U_n}(m') \approx \hat{E}_{U_n}(m')$ but $\hat{E}_{U_n}(m) \approx \hat{E}_{U_n}(m')$ since \hat{E} is essentially the same as the t -times repetition scheme we analyzed above. Thus by the triangle inequality we can conclude that $E_{U_n}(m) \approx E_{U_n}(m')$ as we desired.

Proof of claim: We prove the claim by the hybrid method. For $j \in \{0, \dots, \ell\}$, let H_j be the distribution of ciphertexts where in the first j blocks we act like \hat{E} and in the last $t-j$ blocks we act like E . That is, we choose $k_0, \dots, k_t, k'_1, \dots, k'_t$ independently at random from U_n and the i^{th} block of H_j is equal to $E'_{k_{i-1}}(k'_i, m_i)$ if $i > j$ and is equal to $E'_{k_{i-1}}(k'_i, m_i)$ if $i \leq j$. Clearly, $H_t = \hat{E}_{U_n}(m)$ and $H_0 = E_{U_n}(m)$ and so it suffices to prove that for every j , $H_j \approx H_{j+1}$. Indeed, let $j \in \{0, \dots, \ell\}$ and suppose towards the sake of contradiction that there exists an efficient Eve' such that

$$|\mathbb{E}[Eve'(H_j)] - \mathbb{E}[Eve'(H_{j+1})]| \geq \epsilon \quad (*)$$

where $\epsilon = \epsilon(n)$ is noticeable. By the averaging principle, there exists some fixed choice for $k'_1, \dots, k'_t, k_0, \dots, k_{j-2}, k_j, \dots, k_t$ such that $(*)$ still holds. Note that in this case the only randomness is the choice of $k_j \leftarrow_R U_n$ and moreover the first $j-1$ blocks and the last $t-j$ blocks of H_j and H_{j+1} would be identical and we can denote them by α and β respectively and hence write $(*)$ as

$$|\mathbb{E}_{k_{j-1}}[Eve'(\alpha, E_{k_{j-1}}(k_j, m_j), \beta) - Eve'(\alpha, E_{k_{j-1}}(k'_j, m_j), \beta)]| \geq \epsilon \quad (**)$$

But now consider the adversary Eve that is defined as $Eve(c) = Eve'(\alpha, c, \beta)$. Then Eve is also efficient and by $(**)$ it can distinguish between $E'_{U_n}(k_{j+1}, m_j)$ and $E'_{U_n}(k'_{j+1}, m_j)$ thus contradicting the security of (E', D') . QED

Appendix: The computational model

For concreteness sake let us give a precise definition of what it means for a function or probabilistic process f mapping $\{0, 1\}^n$ to $\{0, 1\}^m$ to be computable using T operations:

Definition: A *probabilistic straightline program* consists of a sequence of lines, each one of them one of the following forms:

- $a = b \text{ NAND } c$ where a is a variable identifier and b, c are either variables that have been assigned a value before, or the constants 0 or 1.
- $a = \text{RAND}$ where a is a variable identifier.
- $a = \text{INPUT}$ where a is a variable identifier.
- $\text{OUTPUT } b$ where b is a variable that has been assigned a value before.

Given a program π , we say that its *size* is the number of lines it contains. If the program has n **INPUT** commands and m **OUTPUT** commands, we identify it with the probabilistic process that maps $\{0, 1\}^n$ to $\{0, 1\}^m$ in the natural way. (That is, the variables all correspond to a single bit in $\{0, 1\}$, every time **INPUT** is called we take a new bit from the input, and every time **OUTPUT** is called we output a new bit.)

If F is a (probabilistic or deterministic) map of $\{0, 1\}^n$ to $\{0, 1\}^m$, the *complexity* of F is the size of the smallest program π that computes it.

If you haven't taken a class such as CS121 before, you might wonder how such a simple model captures complicated programs that use loops, conditionals, and more complex data types than simply a bit in $\{0, 1\}$, not to mention some special purpose crypto-breaking devices that might involve tailor-made hardware. It turns out that it does (for the same reason we can compile complicated programming languages to run on silicon chips with a very limited instruction set). In fact, as far as we know, this model can capture even computations that happen in nature, whether it's in a bee colony or the human brain (which contains about 10^{10} neurons, so should in principle be simulatable by a program that has up to a few order of magnitudes the same number of lines). Crucially, for cryptography, we care about such programs not because we want to actually run them, but because we want to argue about their *non existence*.⁸ If we have a process that cannot be computed by a straightline program of length shorter than $2^{128} > 10^{38}$ then it seems safe to say that a computer the size of the human brain (or even all the human and nonhuman brains on this planet) will not be able to perform it either.

⁸An interesting potential exception to this principle that every natural process should be simulatable by a straightline program of comparable complexity are processes where the quantum mechanical notions of *interference* and *entanglement* play a significant role. We will talk about this notion of *quantum computing* towards the end of the course, though note that much of what we say does not really change when we add quantum into the picture. We can still capture these processes by straightline programs (that now have somewhat more complex form), and so most of what we'll do just carries over in the same way to the quantum realm as long as we are fine with conjecturing the strong form of the Cipher conjecture and similar ones, namely that these are infeasible to break even for quantum computers. (All current evidence points toward these strong forms being true as well.)

Homework 1

Total of 141 points. (Note that while this exercise is long, 100 points are a perfect score, so you don't *have* to solve all questions if you don't have the time for it.)

0. (10 points + 5 points bonus) Log in to canvas and: **(a)** Post on the canvas discussion board a short message introducing yourself to the rest of the class- what's your background and why you are interested in cryptography. Feel free to also add something about your non academic interests and hobbies. For a bonus of 5 points include a photo of yourself. **(b)** Answer on the "Week 0" module in canvas the "background questionnaire" quiz. This is not graded and there are no wrong answers- it's just a way for me to get a better sense of people's backgrounds.

Exercises from the "mathematical background" handout.

1. (16 points) In the following exercise X, Y denote random variables over some sample space S . You can assume that the probability on S is the uniform distribution— every point s is output with probability $1/|S|$. Thus $\mathbb{E}[X] = (1/|S|) \sum_{s \in S} X(s)$. We define the variance and standard deviation of X and Y as above (e.g., $\text{Var}[X] = \mathbb{E}[(X - \mathbb{E}[X])^2]$ and the standard deviation is the square root of the variance).
 - a. (2 points) Prove that $\text{Var}[X]$ is always non-negative.
 - b. (2 points) Prove that $\text{Var}[X] = \mathbb{E}[X^2] - \mathbb{E}[X]^2$.
 - c. (2 points) Prove that always $\mathbb{E}[X^2] \geq \mathbb{E}[X]^2$.
 - d. (2 points) Give an example for a random variable X such that $\mathbb{E}[X^2] \neq \mathbb{E}[X]^2$.
 - e. (2 points) Give an example for a random variable X such that its standard deviation is *not equal* to $\mathbb{E}[|X - \mathbb{E}[X]|]$.
 - f. (2 points) Give an example for two random variables X, Y such that $\mathbb{E}[XY] = \mathbb{E}[X]\mathbb{E}[Y]$.
 - g. (2 points) Give an example for two random variables X, Y such that $\mathbb{E}[XY] \neq \mathbb{E}[X]\mathbb{E}[Y]$.

- h. (2 points) Prove that if X and Y are independent random variables (i.e., for every x, y , $\Pr[X = x \wedge Y = y] = \Pr[X = x] \Pr[Y = y]$) then $\mathbb{E}[XY] = \mathbb{E}[X]\mathbb{E}[Y]$ and $\text{Var}[X + Y] = \text{Var}[X] + \text{Var}[Y]$.
2. (15 points) Suppose that H is chosen to be a random function mapping the numbers $\{1, \dots, n\}$ to the numbers $\{1, \dots, m\}$. That is, for every $i \in \{1, \dots, n\}$, $H(i)$ is chosen to be a random number in $\{1, \dots, m\}$ and that choice is done independently for every i . For every $i \leq j \in \{1, \dots, n\}$, define the random variable $X_{i,j}$ to equal 1 if there was a *collision* between $H(i)$ and $H(j)$ in the sense that $H(i) = H(j)$ and to equal 0 otherwise.
- (3 points) For every $i \leq j$, compute $\mathbb{E}[X_{i,j}]$.
 - (3 points) Define $Y = \sum_{i \leq j} X_{i,j}$ to be the total number of collisions. Compute $\mathbb{E}[Y]$ as a function of n and m . In particular your answer should imply that if $m < n^2/1000$ then $\mathbb{E}[Y] > 1$ and hence in expectation there should be at least one collision and so the function H will not be one to one.
 - (3 points) Prove that if $m > 1000 \cdot n^2$ then the probability that H is one to one is at least 0.9.
 - (3 points) Give an example of a random variable Z (unrelated to the function H) that is always equal to a non-negative integer, and such that $\mathbb{E}[Z] \geq 1000$ but $\Pr[Z > 0] < 0.001$.
 - (3 points) Prove that if $m < n^2/1000$ then the probability that H is one to one is at most 0.1.
3. (15 points) In this exercise we will work out an important special case of the Chernoff bound. You can take as a given the following facts:
- The number of $x \in \{0, 1\}^n$ such that $\sum x_i = k$ is $\binom{n}{k} = \frac{n!}{k!(n-k)!}$.
 - Stirling's approximation formula: for every $n \geq 1$,

$$\sqrt{2\pi n} \left(\frac{n}{e}\right)^n \leq n! \leq 2\sqrt{2\pi n} \left(\frac{n}{e}\right)^n$$

where $e = 2.7182\dots$ is the base of the natural logarithm.

Do the following:

- (5 points) Prove that for every n , $\Pr_{x \leftarrow \mathcal{R}}[\sum x_i \geq 0.6n] < 2^{-n/1000}$

The above shows that if you were given a coin of bias at least 0.6, you should only need some constant number of samples to be able to reject the “null hypothesis” that the coin is completely unbiased with extremely high confidence. In the following somewhat more challenging questions (which can be considered as bonus exercise) we try to show a converse to this:

- Let P be the uniform distribution over $\{0, 1\}^n$ and Q be the $1/2 + \epsilon$ -biased distribution corresponding to tossing n coins in which each one has a probability of $1/2 + \epsilon$ of equalling 1 and probability $1/2 - \epsilon$

of equalling 0. Namely the probability of $x \in \{0, 1\}^n$ according to Q is equal to $\prod_{i=1}^n (1/2 - \epsilon + 2\epsilon x_i)$.

- i. (5 points) Prove that for every threshold θ between 0 and n , if $n < 1/(100\epsilon)^2$ then the probabilities that $\sum x_i \leq \theta$ under P and Q respectively differ by at most 0.1. Therefore, one cannot use the test whether the number of heads is above or below some threshold to reliably distinguish between these two possibilities unless the number of samples n of the coins is at least some constant times $1/\epsilon^2$.
- ii. (5 points) Prove that for *every* function F mapping $\{0, 1\}^n$ to $\{0, 1\}$, if $n < 1/(100\epsilon)^2$ then the probabilities that $F(x) = 1$ under P and Q respectively differ by at most 0.1. Therefore, if the number of samples is smaller than a constant times $1/\epsilon^2$ then there is simply *no test* that can reliably distinguish between these two possibilities.

Exercises from Lecture 1

4. (20 points) Prove that every encryption scheme (E, D) is perfectly secret if and only if for every plaintexts $m, m' \in \{0, 1\}^\ell$, the two random variables $\{E_k(m)\}$ and $\{E_{k'}(m')\}$ (for randomly chosen keys k and k') have precisely the same distribution.
5. (20 points- a bit harder bonus question) In the lecture we saw that any perfectly secret private key encryption scheme needs to use a key as large as the message. But we actually made an implicit subtle assumption: that the encryption process is *deterministic*. In a *probabilistic encryption scheme*, the encryption function E may be probabilistic: that is, given a message m and a key k , the value $E_k(x)$ is not fixed but is distributed according to some distribution $C_{x,k}$. The decryption function is still given only the key k and not the internal randomness used by E , and we require that for every message m , $\Pr[D_k(E_k(m)) = m] > 0.99$ where this probability is taken both over the choice of the key k and the internal randomness used by E . Prove that even a probabilistic encryption scheme cannot be perfectly secret with a key that's significantly shorter than the message. That is, show that for every probabilistic encryption scheme (E, D) using n -length keys and $n + 10$ -length messages, there exist two messages $m, m' \in \{0, 1\}^{n+10}$ such that the distributions $\{E_k(m)\}$ and $\{E_{k'}(m')\}$ are not identical.

Exercises from Lecture 2

6. (20 points) Prove the Computational Indistinguishability phrasing of computational security Theorem.

7. (20 points) Give a direct proof (not going through computational indistinguishability) in your own words for the length extension theorem in the special case $t = 2$ and when the messages are $m^0 = 00$ and $m^1 = 01$. That is, show how to transform an adversary Eve that can distinguish between the distribution $C^0 = (E'_{k_0}(k_1, 0), E'_{k_1}(k_2, 0))$ and $C^1 = (E'_{k_0}(k_1, 0), E'_{k_1}(k_2, 1))$ (for random k_0, k_1, k_2) with advantage ϵ into an adversary Eve' that runs in time polynomial in the running time of Eve and can distinguish between $E'_k(m')$ and $E'_k(m'')$ for two messages $m', m'' \in \{0, 1\}^{n+1}$ with advantage at least, say, $\epsilon/10$.