Computational security

Boaz Barak

**Disclaimer:** The MS-Word version of this text has significant formatting issues compared to the PDF and HTML versions.

# Computational Security

**Additional reading:** Sections 2.2 and 2.3 in Boneh-Shoup book. 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 bit key, with a bit message:

from itertools import product # Import an iterator for cartesian products  
from random import choice # choose random element of list  
  
# Gets ciphertext as input and two potential plaintexts  
# Returns most likely plaintext  
# We assume we have access to the function Encrypt(key,plaintext)  
def Distinguish(ciphertext,plaintext1,plaintext2):  
 for key in product([0,1], repeat = 128): # Iterate over all possible keys of length 128  
 if Encrypt(key, plaintext1)==ciphertext:  
 return plaintext1  
 if Encrypt(key, plaintext2)==ciphertext:  
 return plaintext2  
 return choice([plaintext1,plaintext2])

The program Distinguish will break any -bit key and -bit message encryption Encrypt, in the sense that there exist a pair of messages such that DistinguishEncrypt with probability at least over and .

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., bits long) with messages that can be 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 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 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 operations” and just say that there is a way to formally do so. We will want to say that a scheme has “ bits of security” if it is not possible to break it using less than operations, and more generally that it has bits of security if it can’t be broken using less than operations. Given the perfect secrecy definition we saw last time, a natural attempt for defining computational secrecy would be the following:

# 

An encryption scheme has  *bits of computational secrecy* if for every two distinct plaintexts and every strategy of Eve using at most computational steps, if we choose at random and a random key , then the probability that Eve guesses after seeing is at most .

**Note:** It is important to keep track of what is known and unknown to the adversary Eve. The adversary knows the set of potential messages, and the ciphertext . The only things she doesn’t know are whether or , and the value of the secret key . In particular, because and are known to Eve, it does not matter whether we define Eve’s goal in this “security game” as outputting or as outputting .

firstcompdef seems very natural, but is in fact *impossible* to achieve if the key is shorter than the message.

# 

Before reading further, you might want to stop and think if you can *prove* that there is no, say, encryption scheme with bits of computational security satisfying firstcompdef with and where the time to compute the encryption is polynomial.

The reason firstcompdef can’t be achieved is that if the message is even one bit longer than the key, we can always have a very efficient procedure that achieves success probability of about by guessing the key. This is because we can replace the loop in the Python program Distinguish by choosing the key at random. Since we have some small chance of guessing correctly, we will get a small advantage over half.

Of course an advantage of in guessing the message is not really something we would worry about. For example, since the earth is about 5 billion years old, we can estimate the chance that an asteroid of the magnitude that caused the dinosaurs’ extinction will hit us this very second to be about . Hence we want to relax the notion of computational security so it would not consider guessing with such a tiny advantage as a “true break” of the scheme. The resulting definition is the following:

# 

An encryption scheme has  *bits of computational secrecy*[[1]](#footnote-25) if for every two distinct plaintexts and every strategy of Eve using at most computational steps, if we choose at random and a random key , then the probability that Eve guesses after seeing is at most .

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.

# 

If has bits of computational secrecy as per compsecconcdef then for every subset and every strategy of Eve using at most computational steps, if we choose at random and a random key , then the probability that Eve guesses after seeing is at most .

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.[[2]](#footnote-27) One way to phrase it is that if the sender used a -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.

# 

Before reading the proof, try to again review the proof of twotomanythm, and see if you can generalize it yourself to the computational setting.

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

* A subset of messages

and

* An adversary such that

$$
\Pr\_{m{\leftarrow\_{\tiny R}}M, k{\leftarrow\_{\tiny R}}{\{0,1\}}^n}[ Eve(E\_k(m))=m ] > 1/|M|
$$

Then there exist two messages and an adversary such that $\Pr\_{b{\leftarrow\_{\tiny R}}{\{0,1\}},k{\leftarrow\_{\tiny R}}{\{0,1\}}^n}[Eve'(E\_k(m\_b))=m\_b ] > 1/2$.

To adapt this proof to the computational setting and complete the proof of the current theorem it suffices to show that:

* If the probability of succeeding was then the probability of succeeding is at least .
* If can be computed in operations, then can be computed in operations.

This will imply that if ran in polynomial time and had polynomial advantage over in guessing a plaintext chosen from , then would run in polynomial time and have polynomial advantage over in guessing a plaintext chosen from .

The first item can be shown by simply doing the same proof more carefully, keeping track how the advantage over for translates into an advantage over for . As the world’s most annoying saying goes, doing this is an excellent exercise for the reader.

The second item is obtained by looking at the definition of from that proof. On input , computed (which costs operations), checked if (which costs, say, at most operations), and then outputted either or a random bit (which is a constant, say at most operations).

### Proof by reduction

The proof of twotomanycomp 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:

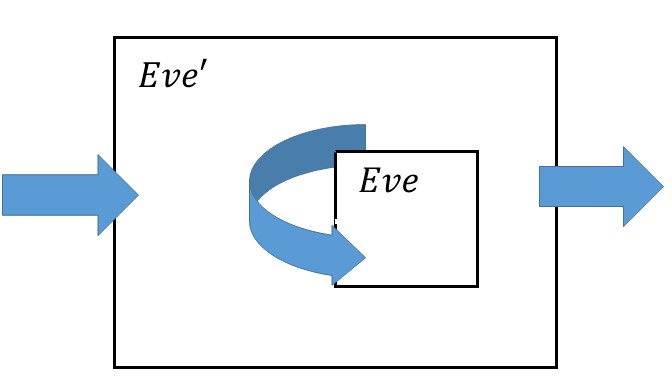
“If there is a scheme satisfying security definition then there is a scheme satisfying security definition ”

In the context of twotomanycomp, was “having bits of security” (in the context distinguishing between encryptions of two ciphertexts) and was the more general notion of hardness of getting a non-trivial advantage over guessing for an encryption of a random . While in twotomanycomp the encryption scheme was the same as , this need not always be the case. However, all of the proofs of such statements will have the same global structure— we will assume towards a contradiction, that there is an efficient adversary strategy demonstrating that the scheme violates the security notion , and build from a strategy demonstrating that violates . This is such an important point that it deserves repeating:

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

For computational secrecy, we will always want that will be efficient if is, and that will usually be the case because will simply use 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 .
* Analyzing the probability of success and in particular showing that if had non-negligible advantage then so will .



We show that the security of implies the security of by transforming an adversary breaking into an adversary breaking .

Note that, just like in the context of NP completeness or uncomputability reductions, security reductions work *backwards*. That is, we construct the scheme based on the scheme , but then prove that we can transform an algorithm breaking into an algorithm breaking . Just like in computational complexity, it can sometimes be hard to keep track of the direction of the reduction. In fact, cryptographic reductions can be even subtler, since they involve an interplay of several entities (for example, sender, receiver, and adversary) and probabilistic choices (e.g., over the message to be sent and the key).

## 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. In practice we would usually like to ensure that when we use a smallish security parameter such as in the few hundreds or thousands then:

* The *honest parties* (the parties running the encryption and decryption algorithms) are extremely efficient, something like 100-1000 cycles per byte of data processed. In theory terms we would want them be using an or at worst time algorithms with not-too-big hidden constants.
* We want to protect against *adversaries* (the parties trying to break the encryption) that have much vaster computational capabilities. A typical modern encryption is built so that using standard key sizes it can withstand the combined computational powers of all computers on earth for several decades. In theory terms we would want the time to break the scheme to be (or if not, at least or ) with not too small hidden constants.

For implementing cryptography in practice, the tradeoff between security and efficiency can be crucial. However, for understanding the *principles* behind cryptography, keeping track of concrete security can be a distraction, and so just like we do in algorithms courses, 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 for some constants (or for short), which we will consider as *efficient*.
* *Exponential* running time of the form for some constants (or for short) which we will consider as *infeasible*.[[3]](#footnote-33)

Another way to say it is that in this course, if a scheme has any security at all, it will have at least bits of security where is the length of the key and is some absolute constant such as . Hence in this course, whenever you hear the term “super polynomial”, you can equate it in your mind with “exponential” and you won’t be far off the truth.

These are not all the theoretically possible running times. One can have intermediate functions such as though we will generally not encounter those. To make things clean (and to correspond to standard terminology), we will generally associate “efficient computation” with *polynomial time* in where is either its input length or the key size (the key size and input length will always be polynomially related, and so this choice won’t matter). We want our algorithms (encryption, decryption, etc.) to be computable in polynomial time, but to require *super polynomial time* to break.

**Negligible probabilities.** In cryptography, we care not just about the running time of the adversary but also about their probability of success (which should be as small as possible). If is a function (which we’ll often think of as corresponding to the adversary’s probability of success or advantage over the trivial probability, as a function of the key size ) then we say that is *negligible* if it’s smaller than the inverse of every (positive) polynomial. Our security definitions will have the following form:

*"Scheme is secure if for every polynomial and time adversary , there is some negligible function such that the probability that succeeds in the security game for is at most* "

We now make these notions more formal.

A function is *negligible* if for every polynomial there exists such that for every .[[4]](#footnote-34)

The following exercise provides a good way to get some comfort with this definition:

1. Let be a negligible function. Prove that for every polynomials with non-negative coefficients such that , the function defined as is negligible.
2. Let . Prove that is negligible if and only if for every constant , .

The above definitions could be confusing if you haven’t encountered asymptotic analysis before. Reading the beginning of Chapter 3 (pages 43-51) in the KL book, as well as the mathematical background lecture in my [intro to TCS notes](http://www.introtcs.org/public/index.html) can be extremely useful. As a rule of thumb, if every time you see the word “polynomial” you imagine the function and every time you see the word “negligible” you imagine the function then you will get the right intuition.

What you need to remember is that negligible is much smaller than any inverse polynomial, while polynomials are closed under multiplication, and so we have the “equations”

and

As mentioned, in practice people really want to get as close as possible to bits of security with an bit key, but we would be happy as long as the security grows with the key, so when we say a scheme is “secure” you can think of it having bits of security (though any function growing faster than would be fine as well).

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.[[5]](#footnote-39)

We can now formally define computational secrecy in asymptotic terms:

# 

An encryption scheme is *computationally secret* if for every two distinct plaintexts and every efficient (i.e., polynomial time) strategy of Eve, if we choose at random and a random key , then the probability that Eve guesses after seeing is at most for some negligible function .

### 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 operations. Fortunately, when we don’t really care about the difference between and, say, , then essentially every reasonable definition gives the same answer.[[6]](#footnote-42) Formally, we can use the notions of Turing machines, Boolean circuits, or straightline programs to define complexity. For concreteness, let’s define that a function has complexity at most if there is a Boolean circuit that computes using at most Boolean gates (say AND/OR/NOT or NAND; alternatively you can choose your favorite universal gate sets.) We will often also consider *probabilistic* functions in which case we allow the circuit a RAND gate that outputs a single random bit (though this in general does not give extra power). The fact that we only care about asymptotics means you don’t really need to think of gates when arguing in cryptography. However, it is comforting to know that this notion has a precise mathematical formulation.

**Uniform vs non-uniform models.** While many computational texts focus on models such as Turing machines, in cryptography it is more convenient to use Boolean circuits which are a [non uniform model](https://introtcs.org/public/lec_11_running_time.html#nonuniformcompsec) of computation in the sense that we allow a different circuit for every given input length. The reasons are the following:

1. Circuits can express *finite* computation, while Turing machines only make sense for computing on arbitrarily large input lengths, and so we can make sense of notions such as “ bits of computational security”.
2. Circuits allow the notion of “hardwiring” whereby if we can compute a certain function using a circuit of gates and have a string then we can compute the function using gates as well. This is useful in many cryptograhic proofs.

One can build the theory of cryptography using Turing machines as well, but it is more cumbersome.

Later on in the course, both our cryptographic schemes and the adversaries will extend beyond simple functions that map an input to an output, and we will consider *interactive algorithms* that exchange messages with one another. Such an algorithm can be implemented using circuits or Turing machines that take as input the prior state and the history of messages up to a certain point in the interaction, and output the next message in the interaction. The number of operations used in such a strategy is the total number of gates used in computing all the messages.

## Our first conjecture

We are now ready to make our first conjecture:

**The Cipher Conjecture:**[[7]](#footnote-48) There exists a computationally secret encryption scheme (where are efficient) with length function .

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 theorem is known:

# 

If then there does not exist a computationally secret encryption with efficient and and where the message is longer than the key.

# 

We just sketch the proof, as this is not the focus of this course. If 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 , 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:

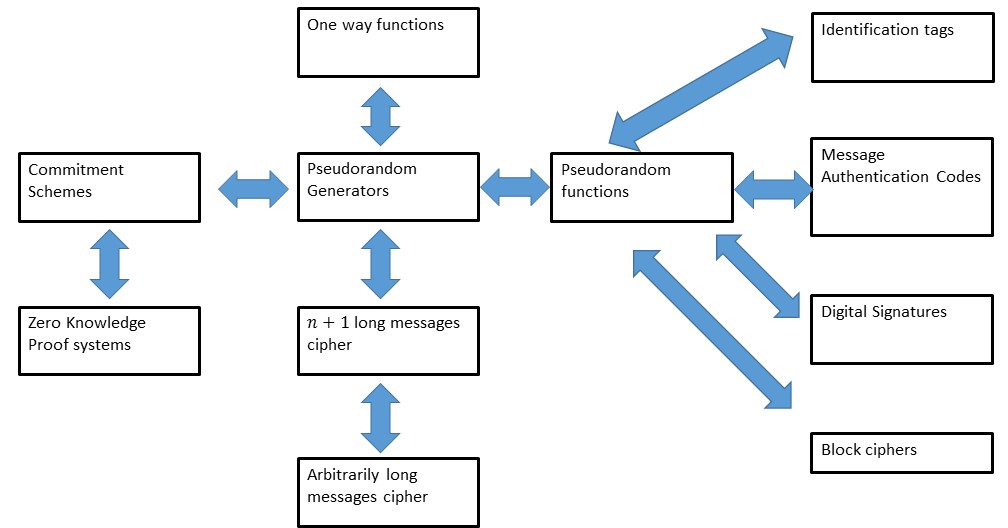
* *Intuition:* If the cipher conjecture is false then it means that for *every* possible cipher we can make the exponential time attack described above become efficient. It seems “too good to be true” in a similar way that the assumption that P=NP seems too good to be true.
* *Concrete candidates:* As 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 Edward Snowden’s revelations) 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 message, say, twice as long as the key. 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 this conjecture 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 secret, and so if we required a message of size instead , 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 -bit long keys, our communication will be smaller by a factor of (or a saving of about ) 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 secret encryption scheme with a message of size for *every* polynomial . In essence, we can fix a single -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.)



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

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 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 and an encryption of . It turns out to be useful to consider this question of when two distributions are *computationally indistinguishable* more broadly:

Let and be two distributions over . We say that and are *-computationally indistinguishable*, denoted by , if for every function computable with at most operations,

Prove that for every and as above if and only if for every -operation computable , the probability that wins in the following game is at most :

1. We pick .
2. If , we let . If , we let .
3. We give the input , and outputs .
4. *wins* if .

Working out this exercise on your own is a great way to get comfortable with computational indistinguishability, which is a fundamental notion.

For every function , let and .

Then the probability that wins the game is:

and since this is

We see that wins the game with success if and only if

Since , this already shows that if and are -indistinguishable then will win the game with probability at most .

For the other direction, assume that and are *not* computationally indistinguishable and let be a time operation function such that

Then by definition of absolute value, there are two options. Either in which case wins the game with probability at least . Otherwise , in which case the function (which is just as easy to compute) wins the game with probability at least .

Note that above we assume that the class of “functions computable in at most operations” is *closed under negation*, in the sense that if is in this class, then is also. For standard Boolean circuits, this can be done if we don’t count negation gates (which can change the total circuit size by at most a factor of two), or we can allow for to require a constant additional number of operations, in which case the exercise is still essentially true but is slightly more cumbersome to state.

As we did with computational secrecy, we can also define an asymptotic definition of computational indistinguishability.

Let be some function and let and be two sequences of distributions such that and are distributions over .

We say that and are *computationally indistinguishable*, denoted by , if for every polynomial and sufficiently large , .

Solving the following asymptotic analog of compindex is a good way to get comfortable with the asymptotic definition of computational indistinguishability:

Let and be as above. Then if and only if for every polynomial-time , there is some negligible function such that wins the following game with probability at most :

1. We pick .
2. If , we let . If , we let .
3. We give the input , and outputs .
4. *wins* if .

**Dropping the index .** Since the index of our distributions would often be clear from context (indeed in most cases it will be the length of the key), we will sometimes drop it from our notation. So if and are two random variables that depend on some index , we will say that is computationally indistinguishable from (denoted as ) when the sequences and are computationally indistinguishable.

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

# 

Let be a valid encryption scheme. Then is computationally secret if and only if for every two messages ,

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

Working out the proof is an excellent way to make sure you understand both the definition of computational secrecy and computational indistinguishability, and hence we leave it as an exercise.

One intuition for computational indistinguishability is 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 is close to and is close to then should be close to .[[8]](#footnote-60) Similarly if four distributions satisfy that is close to and is close to , then you might expect that the distribution where we take two independent samples from and respectively, is close to the distribution where we take two independent samples from and respectively. We will now verify that these intuitions are in fact correct:

# 

Suppose . Then .

# 

Suppose that there exists a time such that

Write

Thus,

and hence in particular there must exist some such that

contradicting the assumption that for all .

# 

Suppose that are distributions over such that . Then .

For every we define to be the distribution . Clearly and . We will prove that for every , , 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 and some -time such that

In other words

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

By the *averaging principle*[[9]](#footnote-64) this means that there exist some values such that

Now and are simply independent draws from the distributions and respectively, and so if we define then runs in time at most the running time of plus [[10]](#footnote-65) and it satisfies

contradicting the assumption that .

# 

The above proof illustrates a powerful technique known as the *hybrid argument* whereby we show that two distribution and are close to each other by coming up with a sequence of distributions such that , and we can argue that is close to for all . 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 or Stream Ciphers

We now turn to show the *length extension theorem*, stating that if we have an encryption for -length messages with -length keys, then we can obtain an encryption with -length messages for every polynomial . For a warm-up, let’s show the easier fact that we can transform an encryption such as above, into one that has keys of length and messages of length for every integer :

# 

Suppose that is a computationally secret encryption scheme with bit keys and bit messages. Then the scheme where and is a computationally secret scheme with bit keys and bit messages.

# 

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 indisinguishability is preserved under many samples. That is, by the security of we know that for every two messages , where is chosen from the distribution . Therefore by the indistinguishability of many samples lemma, for every two tuples and ,

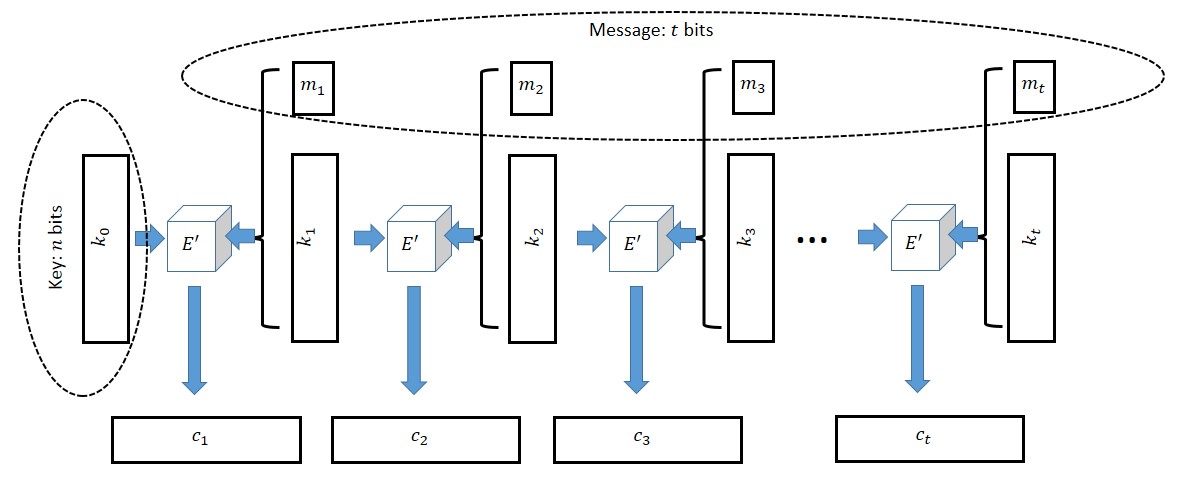
for random chosen independently from which is exactly the condition that is computationally secret.

**Randomized encryption scheme.** We can now prove the full length extension theorem. Before doing so, we will need to generalize the notion of an encryption scheme to allow a *randomized encryption scheme*. That is, we will consider encryption schemes where the encryption algorithm can “toss coins” in its computation. There is a crucial difference between key material and such “as hoc” (sometimes also known as “ephemeral”) randomness. Keys need to be not only chosen at random, but also shared in advance between the sender and receiver, and stored securely throughout their lifetime. The “coin tosses” used by a randomized encryption scheme are generated “on the fly” and are not known to the receiver, nor do they need to be stored long term by the sender. So, allowing such randomized encryption does not make a difference for most applications of encryption schemes. In fact, as we will see later in this course, randomized encryption is *necessary* for security against more sophisticated attacks such as chosen plaintext and chosen ciphertext attacks, as well as for obtaining secure *public key* encryptions. We will use the notation to denote the output of the encryption algorithm on key , message and using internal randomness . We often suppress the notation for the randomness, and hence use to denote the random variable obtained by sampling a random and outputting .

We can now show that given an encryption scheme with messages one bit longer than the key, we can obtain a (randomized) encryption scheme with arbitrarily long messages:

# 

Suppose that there exists a computationally secret encryption scheme with key length and message length . Then for every polynomial there exists a (randomized) computationally secret encryption scheme with key length and message length .



Constructing a cipher with bit long messages from one with long messages

This is perhaps our first example of a non trivial cryptographic theorem, and the blueprint for this proof will be one that we will follow time and again during this course. Please make sure you read this proof carefully and follow the argument.

The construction, depicted in cipherlengthextensionfig, is actually quite natural and variants of it are used in practice for *stream ciphers*, which are ways to encrypt arbitrarily long messages using a fixed size key. The idea is that we use a key of size to encrypt **(1)** a fresh key of size and **(2)** one bit of the message. Now we can encrypt using and so on and so forth. We now describe the construction and analysis in detail.

Let . We are given a cipher which can encrypt -bit long messages with an -bit long key and we need to encrypt a -bit long message . Our idea is simple (at least in hindsight). Let $k\_0 {\leftarrow\_{\tiny R}}{\{0,1\}}^n$ be our key (which is chosen at random). To encrypt using , the encryption function will choose random strings $k\_1,\ldots, k\_t {\leftarrow\_{\tiny R}}{\{0,1\}}^n$. We will then encrypt the -bit long message with the key to obtain the ciphertext , then encrypt the -bit long message with the key to obtain the ciphertext , and so on and so forth until we encrypt the message with the key .[[11]](#footnote-73) We output as the final ciphertext.[[12]](#footnote-74)

To decrypt using the key , first decrypt to learn , then use to decrypt to learn , and so on until we use to decrypt and learn . Finally we can simply output .

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

Our discussion above looks like a reasonable intuitive argument, but to make sure it’s true we need to give an actual proof. Let be two messages. We need to show that . The heart of the proof will be the following claim:

**Claim:** Let be the algorithm that on input a message and key works like except that its block contains where is a *random* string in , that is chosen *independently* of everything else including the key . Then, for every message

Note that 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. Since both and are randomized encryption schemes (with using bits of randomness for the ephemeral keys and using bits of randomness for the ephemeral keys ), we can also write lengthextendclaimeq as

where we use to denote a random variable that is chosen uniformly at random from and independently from the choice of (which is chosen uniformly at random from ).

Once we prove the claim then we are done since we know that for every pair of messages , and but since is essentially the same as the -times repetition scheme we analyzed above. Thus by the triangle inequality we can conclude that as we desired.

**Proof of claim:** We prove the claim by the hybrid method. For , let be the distribution of ciphertexts where in the first blocks we act like and in the last blocks we act like . That is, we choose independently at random from and the block of is equal to if and is equal to if . Clearly, and and so it suffices to prove that for every , . Indeed, let and suppose towards the sake of contradiction that there exists an efficient such that

where is noticeable. By the averaging principle, there exists some fixed choice for such that still holds. Note that in this case the only randomness is the choice of $k\_{j-1}{\leftarrow\_{\tiny R}}U\_n$ and moreover the first blocks and the last blocks of and would be identical and we can denote them by and respectively and hence write as

But now consider the adversary that is defined as . Then is also efficient and by it can distinguish between and thus contradicting the security of . This concludes the proof of the claim and hence the theorem.

### Appendix: The computational model

For concreteness sake let us give a precise definition of what it means for a function or probabilistic process mapping to to be computable using operations.

* If you have taken any course on computational complexity (such as Harvard CS 121), then this is the model of Boolean circuits, except that we also allow randomization.
* If you have not taken such a course, you might simply take it on faith that it is possible to model what it means for an algorithm to be able to map an input into an output using “elementary operations”.

In both cases you might want to skip this appendix and only return to it if you find something confusing.

The model we use is a Boolean circuit that also has a gate that outputs a random bit. We could use as the basic set of gates the standard , and but for simplicity we use the one-element set . We represent the circuit as a straightline program, but this is of course just a matter of convenience. As shown (for example) in the [CS 121 textbook](http://introtcs.org), these two representations are identical.

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

* foo = NAND(bar, baz) where foo,bar,baz are variable identifiers.
* foo = RAND() where foo is a variable identifier.

Given a program , we say that its *size* is the number of lines it contains. Variables of the form X[] or Y[] are considered input and output variables respectively. If the input variables range from to and the output variables range from to then the program computes the probabilistic process that maps to in the natural way. If is a (probabilistic or deterministic) map of to , the *complexity* of is the size of the smallest program 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 , 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 neurons, so should in principle be simulatable by a program that has up to a few order of magnitudes of 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*. If we have a process that cannot be computed by a straightline program of length shorter than 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.

**Advanced note: non uniformity.** The computational model we use in this class is [*non uniform*](https://introtcs.org/public/lec_11_running_time.html#nonuniformcompsec) (corresponding to Boolean circuits) as opposed to *uniform* (corresponding to Turing machines). If this distinction doesn’t mean anything to you, you can ignore it as it won’t play a significant role in what we do next. It basically means that we do allow our programs to have hardwired constants of bits where is the input/key length. In fact, to be precise, we will hold ourselves to a higher standard than our adversary, in the sense that we require our algorithms to be efficient in the stronger sense of being computable in uniform probabilistic polynomial time (for some fixed polynomial, often or )), while the adversary is allowed to use non uniformity.

**Quantum computing.** An 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. As discussed in [the CS 121 text](https://introtcs.org/public/lec_26_quantum_computing.html), 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, namely that the cipher is infeasible to break even for quantum computers. All current evidence points toward this strong form being true as well. The field of constructing encryption schemes that are potentially secure against quantum computers is known as [post quantum cryptography](https://en.wikipedia.org/wiki/Post-quantum_cryptography) and we will return to this later in the course.

1. Another version of “ bits of security” is that a scheme has bits of security if for every , an attacker running in time can’t get success probability advantage more than . However these two definitions only differ from one another by at most a factor of two. This may be important for practical applications (where the difference between and bits of security could be crucial) but won’t matter for our concerns. [↑](#footnote-ref-25)
2. The latter property is known as “semantic security”, see also section 3.2.2 of Katz Lindell on “semantic security” and Section 2 of Boneh-Shoup “computational ciphers and semantic security”. [↑](#footnote-ref-27)
3. Some texts reserve the term *exponential* to functions of the form for some and call a function such as, say, *subexponential* . However, we will generally not make this distinction in this course. [↑](#footnote-ref-33)
4. Negligible functions are sometimes defined with image equalling as opposed to the set of non-negative real numbers, since they are typically used to bound probabilities. However, this does not make much difference since if is negligible then for large enough , will be smaller than one. [↑](#footnote-ref-34)
5. Note that there is a subtle issue here with the order of quantifiers. For a scheme to be efficient, the algorithms such as encryption and decryption need to run in some *fixed* polynomial time such as or . In contrast we allow the adversary to run in *any* polynomial time. That is, for every , if is large enough, then the scheme should be secure against an adversary that runs in time . This is in line with the general principle in cryptography that we always allow the adversary potentially much more resources than those used by the honest users. In practical security we often assume that the gap between the honest use and the adversary resources can be *exponential*. For example, a low power embedded device can encrypt messages that, as far as we know, are undecipherable even by a nation-state using super-computers and massive data centers. [↑](#footnote-ref-39)
6. With some caveats that need to be added due to *quantum computers*: we’ll get to those later in the course, though they won’t change most of our theory. See also [this discussion in my intro TCS textbook](https://introtcs.org/public/lec_04_code_and_data.html#PECTTsec) and [this presentation of Aaronson](https://www.scottaaronson.com/talks/bernays2.ppt) on the “extended Church Turing thesis”. [↑](#footnote-ref-42)
7. As 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. [↑](#footnote-ref-48)
8. Results 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 , the distance from to is not larger than the distance from to plus the distance from to . In other words, the edge of the triangle is not longer than the sum of the lengths of the other two edges and . [↑](#footnote-ref-60)
9. This is the principle that if the average grade in an exam was at least then *someone* must have gotten at least , or in other words that if a real-valued random variable satisfies then . [↑](#footnote-ref-64)
10. The cost is for the operations of feeding the “hardwired” strings , into . These take up at most bits, and depending on the computational model, storing and feeding them into may take steps for some small constant . In the future, we will usually ignore such minor details and simply say that if runs in polynomial time then so will . [↑](#footnote-ref-65)
11. The keys are sometimes known as *ephemeral keys* in the crypto literature, since they are created only for the purposes of this particular interaction. [↑](#footnote-ref-73)
12. The astute reader might note that the key is actually not used anywhere in the encryption nor decryption and hence we could encrypt more bits of the message instead in this final round. We used the current description for the sake of symmetry and simplicity of exposition. [↑](#footnote-ref-74)