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. d 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 assasination 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

[1] Because 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.

[2] Traditionally, *cryptography* was the name for the activity of *making* codes, while *cryptoanalysis* 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.

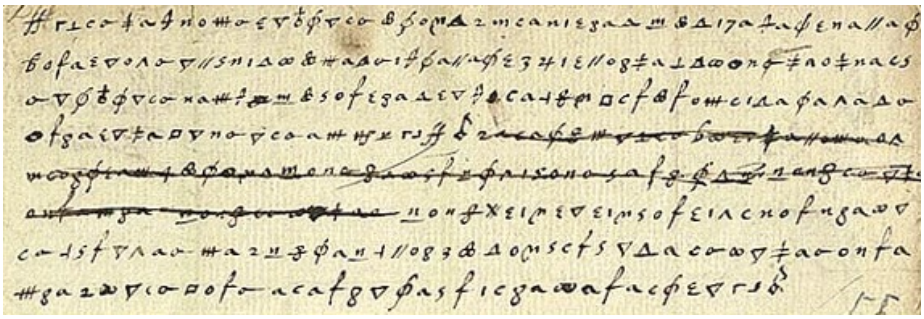

Figure 1: Snippet from encrypted communication between queen Mary and Sir Babington

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 "inscrutable" 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 subsitution cyphers - 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 $n+1^{st}$ letter is again encoded with the first cipher. The key is usually a word or a phrase of n letters, and the i^{th} substition 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 cryptanalzed by Union officers.

Figure 3: Confederate Cipher Disk for implementing the Vigenère cipher

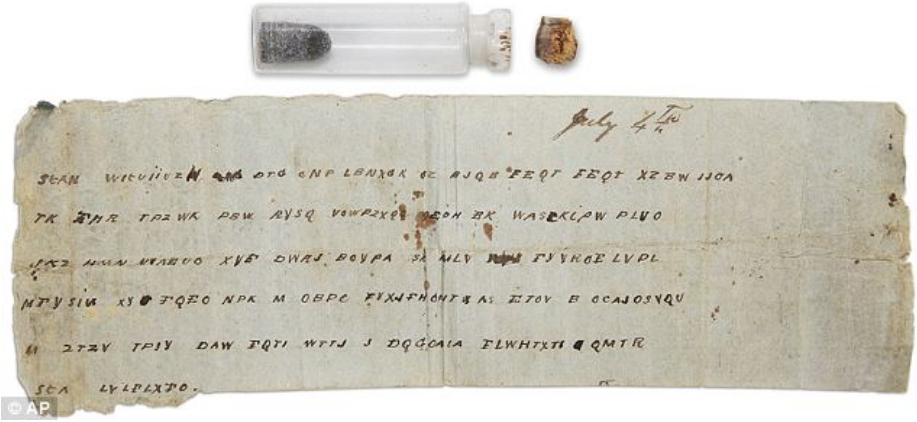

Figure 4: Confederate encryption of the message "Gen'l Pemberton: You can expect no help from this side of the river. Let Gen'l Johnston know, if possible, when you can attack the same point on the enemy's lines. Inform me also and I will endeavor to make a diversion. I have sent some caps. I subjoin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

[3] Here 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 $m \in zo^\ell$,

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

> *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 lets 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.

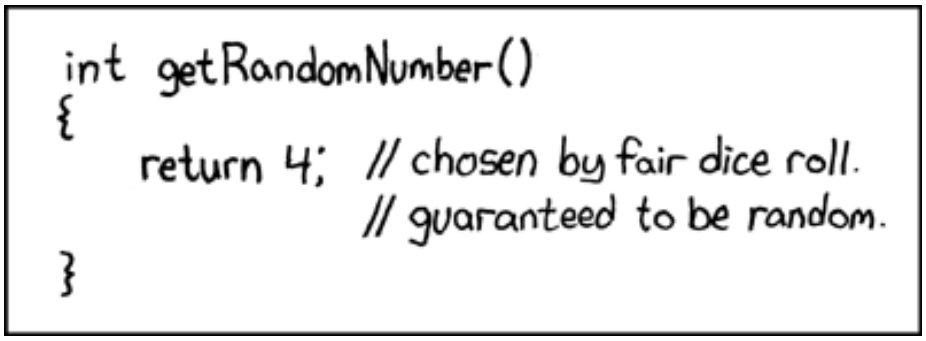

Figure 5: XKCD Cartoon: Random number generator

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 estimte 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 use the following "unpredicatable" 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 htier 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 Frguson, 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) ingeniuity 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 encyption scheme (E, D) is n-*secure* if no matter what method Eve employs, the probability that she can

[4] There 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 diamter 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 outputing 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 $Eve : \{0,1\}^\ell \to \{0,1\}^n$ and for every $m \in \{0,1\}^\ell$, the probability that $Eve(E_k(m)) = s$ is exactly 2^{-n}.

Proof: This follows beacuse $E_k(m) = m$ 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 encyption 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 remeber that we are being anal, and truly insist that the definition holds as stated, namely that for every plaintext message m and every function $Eve : \{0,1\}^o \to \{0,1\}^\ell$, 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 encyption 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 guressing 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 ciphtertext 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 possiblities 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, Virgnia 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, Alger 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 sounds 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* carefuly 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 impossiblity 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 $Eve : \{0,1\}^o \to \{0,1\}^\ell$ 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 $f_1, ..., f_N$ mapping $\{0,1\}^o$ to $\{0,1\}^\ell$, and some probabilities $p_1, \ldots, p_N$ (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 impossiblity 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).