Cryptography is the science of secure communication and computation in insecure environments. Cryptographic methods underlie the security of modern computer systems, enabling private communication in the presence of eavesdroppers, identification in digital environments, and ensuring data integrity. These technologies have become such an integral part of our daily experience — the facebook login, the swipe of the octopus card, the online purchase, the installation of a new (untrusted) app — that we rarely pause to think what makes them possible.

The design of cryptographic algorithms is a delicate task because such algorithms are meant to operate in *adversarial* environments. The behavior of an attacker, be it the hacker who wants to impersonate you on facebook, the user who tries to double-spend his octopus card balance, or the vendor who wants you to install a counterfeit app, is in general unpredictable. Moreover, attackers are often inclined to expend much larger resources in compromising a system than what legitimate users may put into maintaining its integrity. Good cryptographic algorithms must work under the threat of wide range of adversarial attacks, most of which cannot be anticipated at the time of the design.

The most salient characteristic of modern cryptography, which distinguishes it from just about any other branch of computer science, is that it imposes no constraints on how an attacker might operate (beyond extremely generous assumptions on the amount of computational resources that are available to it). Although this modeling decision places seemingly impossible design constraints, it is what makes cryptographic solutions so powerful. It also enables the development of a rigorous theory of cryptography in which the notion of a cryptographic attack is at the same time realistic and has a precise mathematical meaning.

The objective of the first few lectures will be to introduce the basic elements of this theory and apply it to develop encryption, identification, and authentication schemes. These will lead to some natural generalizations and conceptualizations that may be less familiar but are fundamental in modern cryptography. One specification that captures all of the above examples and many more is that of *secure multiparty computation*. Here several parties that do not trust one another want to perform some joint computation without revealing their private inputs. For example these may be voters in an election, or hospitals comparing the effectiveness of their treatments, or gamblers playing an online game of poker in the absence of a trusted card dealer. Identification protocols in which you prove your identity without publicly revealing your "password" lead to the notion of *zero-knowledge proofs*, a method by which one can prove the veracity of any statement (like the knowledge of your password) without revealing any additional information (like what is the first letter in your password).

Despite its considerable reach, cryptography does not provide solutions that meet all imaginable security requirements. One familiar example of a problem without a cryptographic solution is copy protection (i.e. the prevention of duplicating data like movies, books, and computer programs). Another task whose impossibility is much less obvious is *program obfuscation*, namely the ability to compile source code to an executable that completely hides the source code. This is somewhat unfortunate because many desirable cryptographic functionalities (for instance, various enhanced forms of encryption) can be implemented fairly easily using a program obfuscator. Much recent research in cryptography is concerned with the feasibility of restricted forms of obfuscation which are sometimes sufficient for the desired applications.

Although this course is of a foundational nature, the concepts and ideas are relevant to several recent trends in computer science. In many "cloud computing" scenarios, computationally restricted

clients outsource storage or computation to powerful servers. This brings out various security issues (keeping the client's data private, trusting the outcome of the computation) that are amenable to cryptographic solutions. Cryptography is also closely related to machine learning: The problem of breaking a cryptographic system can in a very precise sense be viewed as learning secret information about it from traces of its behavior. This connection reveals important insights on the limitations of learning algorithms. Last but not least, there has been much recent interest in distributed consensus algorithms, especially blockchains which combine cryptographic tools and incentives to make some people very rich! I hope to touch upon some of these connections in the course.

1 Secret sharing

Our first example of a cryptographic functionality will be *secret sharing*. This is good playground to practice cryptographic reasoning while avoiding some of the complexities that will arise later on. Informally, a secret sharing scheme is a mechanism by which a dealer distributes a secret piece of information among a number of parties so that the secret can be recovered, but subsets of parties learn no information about the secret.

Before we define the concept formally let us give an example. Suppose the secret information consists of a single bit $s \in \{0,1\}$ that the dealer wants to split among three parties. The dealer samples *shares* X_1, X_2, X_3 for the three parties as follows:

$$\begin{aligned} &\text{if } s = 0, \quad \text{sample } X_1X_2X_3 \text{ uniformly from } \{000, 011, 101, 110\} \\ &\text{if } s = 1, \quad \text{sample } X_1X_2X_3 \text{ uniformly from } \{111, 100, 010, 001\}. \end{aligned}$$

After sampling the dealer communicates the shares X_1, X_2, X_3 to the three parties respectively. If the three parties get together, they can reconstruct the secret uniquely from their shares X_1, X_2, X_3. On the other hand, regardless of the value of the secret s, the first party's share X_1 is a uniformly random bit (0 with probability half and 1 with probability half). The same is true for the shares X_2 and X_3. In fact, more is true: If parties 1 and 2 get together, the pair of values X_1X_2 that they observe is uniformly distributed over the outcomes $\{00, 01, 10, 11\}$, regardless of the value of the secret. The same is true for X_2X_3 and X_1X_3.

In conclusion, the mechanism that we described has the following two properties: (1) All three parties can recover the secret bit and (2) The information visible to any single party and any pair of parties is independent from the value of the secret bit.

This simple example already illustrates a basic premise of cryptography: Randomness is essential for hiding information. Moreover it is implicitly assumed that this randomness, as well as all intermediate states of a computation (such as the sampling process carried out by the dealer) can be kept private from the other parties. These assumptions may appear uncontroversial but they are not always realistic in implementations. Some famous attacks on cryptographic systems exploit weaknesses in random number generation or the leakage of "side information" that is not meant to be public. In this course we will not concern ourselves with such issues: We will assume that parties have access to private uniform independent random bits and that they can carry out their local computations without leaking any extraneous information.

The above example illustrates a particular *implementation* of a secret sharing scheme. Modern cryptography evaluates the strengths and weaknesses of implementations in the context of security definitions that precisely describe the requirements of cryptographic systems. A cryptographic definition typically has two components: A *functionality requirement* that specifies how the system should behave under proper use, and a *security requirement* that captures the types of adversarial behavior that the system must withstand.

The functionality requirement is usually easier to specify. In the context of secret sharing, we can model the functionality as a pair of algorithms $(Share, Rec)$ for sharing and reconstructing the secret, respectively.

Definition 1. A secret-sharing scheme for secret alphabet Σ, share alphabet Γ, and n parties is a pair of algorithms $(Share, Rec)$, where $Share$ is a randomized algorithm that takes an input $s \in \Sigma$ and outputs shares $X_1, \ldots, X_n \in \Gamma$ and Rec is an algorithm that takes n inputs from Γ and produces an output in Σ such that for every $s \in \Sigma$, $Rec(Share(s)) = s$ with probability one.

In the above example, the sharing procedure is described explicitly but the reconstruction procedure is not. One simple implementation is $Rec(x_1, x_2, x_3) = x_1 + x_2 + x_3 \bmod 2$.

2 Definitions and proofs of security

The security requirement should capture the condition that any proper subset of the parties does not "know" the secret. At the very least this means the secret cannot be recovered by an unauthorized subset of parties. But should we allow some partial leakage of information, for example should the parties be allowed to tell in the absence of prior information) that the secret bit is 60% likely to be a 1 and 40% likely to be a 0?

Cryptographic security definitions disallow even such partial release of unauthorized information. One reason is that while the effect of a single leak might not be particularly harmful, cryptographic systems are often used many times and the composed effect of multiple leaks might lead to a complete break. It is preferable to disallow this possibility. In the context of secret sharing security, this means proper subsets of the parties should not find out any information about the secret.

In cryptography there are two useful paradigms for capturing the absence of information: *indistinguishability* and *simulatability*.

Indistinguishability-based security definitions compare the adversary's views of the system under different instantiations of the secret information. If the adversary cannot tell apart any two such views then he learns no information about the secret. For secret sharing this means the following:

Definition 2. A secret sharing scheme is *IND-secure* if for every proper subset S of parties and for every two secrets s, s' the random variables $(X_i)_{i \in S}$ and $(X'_i)_{i \in S}$ are identically distributed, where $(X_1, \ldots, X_n)$ and $(X'_1, \ldots, X'_n)$ are the outputs of $Share(s)$ and $Share(s')$, respectively.

Here, the notation $(X_i)_{i \in S}$ denotes the collection of random variables X_i for all $i \in S$ under some canonical ordering, e.g. $(X_i)_{i \in \{2,3,5\}} = (X_2, X_3, X_5)$. In words, the joint view of any proper subset of the parties is identical regardless of the value of the secret.

The simulation paradigm describes "absence of information" by requiring that whatever the adversary can observes in the actual run of the system, he can *simulate* on his own without knowledge of the secret information. Therefore anything he learns must be independent of the secret. In the context of secret sharing, the adversary observes the joint distribution of shares $(X_i)_{i \in S}$. The definition requires that he can sample from this distribution without knowing the secret:

Definition 3. A secret sharing scheme is *SIM-secure* if for every proper subset S of parties there exists a randomized algorithm Sim_S (that knows S but takes no additional inputs) such that for every secret s, the output of Sim is identically distributed to $(X_i)_{i \in S}$, where $(X_1, \ldots, X_n)$ is the output of $Share(s)$.

In the case of secret sharing the two definitions are equivalent:

Proposition 4. *A secret sharing scheme is IND-secure if and only if it is SIM-secure.*

Proof. First let us assume that $(Share, Rec)$ is IND-secure. We show that it is SIM-secure. The algorithm Sim_S runs $Share(s_0)$ for some arbitrary but fixed value of the secret $s_0 \in \Sigma$ and outputs the collection of shares indexed by the parties in S. By IND-security, for any input s, the distribution of the outputs of $Share(s)$ indexed by the parties in S is identical to the output of the simulator, so the scheme is SIM-secure.

Now suppose $(Share, Rec)$ is SIM-secure. Then for every proper S the distributions of the random variables $(X_i)_{i\in S}$ and $(X'_i)_{i\in S}$ in Definition 2 are both identical to the output distribution of Sim_S, so they are identical to one another. Therefore $(Share, Rec)$ is IND-secure. □

Since IND-security and SIM-security are equivalent, we'll call a secret sharing scheme that satisfies either definition *secure*. We now construct a secret sharing scheme for one-bit secrets and prove its security. The algorithm $Share(s)$ samples the string $X_1X_2\cdots X_n$ uniformly at random from $\{0,1\}^n$ conditioned on $X_1+\cdots+X_n = s \bmod 2$ and assigns share X_i to party i. The reconstruction algorithm outputs $Rec(x_1,\ldots,x_n) = x_1+\cdots+x_n \bmod 2$.

Theorem 5. $(Share, Rec)$ *is a secure secret sharing scheme with alphabets* $\Sigma = \Gamma = \{0,1\}$.

Proof. The functionality requirement $Rec(Share(s)) = s$ for all s is clearly satisfied. We show SIM-security. For every set S, the simulator Sim_S outputs uniform independent random bits, one for every party in S. To prove security we need to argue that for every proper subset S the random variables $(X_i)_{i\in S}$ are uniform and independent even when conditioned on $X_1+\cdots+X_n = s \bmod 2$. This is true because for every possible fixing of the variables $(X_i)_{i\in S}$ there are exactly $2^{n-|S|-1}$ possible assignments to the other variables $(X_i)_{i\notin S}$ that satisfy the equation $X_1+\cdots+X_n = s \bmod 2$, so all assignments to $(X_i)_{i\in S}$ are equally likely to occur. □

Now that we have a secret sharing scheme for single-bit secrets we can extend it to obtain one for arbitrarily long secrets by applying it independently to each bit of the secret. More precisely, given a long secret $s_1\cdots s_k \in \{0,1\}^k$, the algorithm $Share'(s_1\cdots s_k)$ applies $Share(s_1),\cdots,Share(s_k)$ using independent randomness and gives party i the collection of the i-th shares produced by these k executions. To recover the i-th bit, the reconstruction algorithm Rec' applies Rec to the i-th bit of the share given to all the parties. The proof of security should be straightforward but notationally tedious so I will omit it.

Theorem 6. $(Share', Rec')$ *is a secure secret sharing scheme with alphabets* $\Sigma = \Gamma = \{0,1\}^k$.

3 Threshold secret sharing and cryptographic games

A client can use the scheme from Theorem 6 to store a k-bit file on a collection of $n \geq 2$ servers with the following guarantees: No proper subset of the servers obtains any information about the file, but the client can reconstruct the file after communicating with all the servers. What if some of the servers break down? It may be desirable that the client can still reconstruct her file from the remaining data. A solution that comes to mind is a variant of secret sharing in which not all n parties are required participate in the reconstruction, but some lower threshold r is sufficient.

We will describe the relevant requirements using the notion of a *game*. This is an imagined execution of a cryptographic protocol which, in addition to the honest parties, involves an adversary who intends to derail the correct execution or find out some forbidden information. In the example of

secret reconstruction with some of the shares missing the adversary might operate according to the following game.

Erasure game with threshold r

1. Adversary chooses a secret $s \in \Sigma$ and submits it to the protocol.
2. Protocol runs $Share(s)$ to output sequence of shares $X_1, \ldots, X_n \in \Gamma$.
3. Adversary looks at $X_1, \ldots, X_n$ and replaces at most $n - r$ of them with the special symbol $\perp$ (indicating a missing share), obtaining a sequence $\tilde{X}_1, \ldots, \tilde{X}_n$.
4. Protocol runs $Rec(\tilde{X}_1, \ldots, \tilde{X}_n)$ and outputs $\tilde{s} \in \Sigma$.

The parameter r in the erasure game quantifies the minimal number of shares that the adversary must leave intact for reconstruction to possibly succeed; apart from that we make no assumption on the adversary's operation. In this game, the adversary's decision on which shares to erase may even depend simultaneously on the values of all the shares.

Definition 7. The secret sharing scheme $(Share, Rec)$ is an r-threshold scheme if for every adversary in the erasure game, the protocol output $\tilde{s}$ equals the secret s with probability one.

Definition 7 combines a functionality requirement (correct reconstruction) and a security requirement (resilience to erasures). How do these relate to the security notions from Definitions 2 and 3, which concern leakage of information by subsets of the parties? Since in r-threshold schemes any r parties have enough information to reconstruct the secret, IND-security and SIM-security can possibly hold only against subsets of size $r - 1$ or less.

Definition 8. An r-threshold scheme is *secure* if it satisfies Definition 3 (equivalently, Definition 2) but only when applied to subsets S of size less than r.

Now that we have all the definitions in place we can describe a solution that satsifies all the requirements.

Theorem 9. *There exists an r-threshold secret sharing scheme for every n, every $1 \leq r \leq n$, and every alphabet size $|\Sigma| = |\Gamma|$ that is larger than n and is the power of a prime number.*

Proof. We identify the alphabets Σ and Γ with the finite field $\mathbb{F}_q$, where q is the power of a prime number larger than n. (If q is a power of two then both the secret and the shares are bit sequences.) We also fix n arbitrary distinct elements $a_1, \ldots, a_n \in \mathbb{F}_q$, all different from zero.

To share a secret s, the dealer samples a random polynomial p of degree $r - 1$ over $\mathbb{F}_q$ conditioned on $p(0) = s$. One way to do so is to pick uniformly random coefficients $s_1, s_2, \ldots, s_{r-1}$ from $\mathbb{F}_q$ and set $p(x) = s + s_1 x + s_2 x^2 + \cdots + s_{r-1} x^{r-1}$. The share sent to party i is the value $p(a_i) \in \mathbb{F}_q$.

On input $x_1, \ldots, x_n$ the reconstruction procedure ignores the missing shares and computes a polynomial $\tilde{p}$ of degree $r - 1$ such that $\tilde{p}(a_i) = x_i$ for all the remaining shares. If at least r shares are provided, p and $\tilde{p}$ are both polynomials of degree at most $r - 1$ whose values agree on r inputs, so they must be the same polynomial (because their difference $p - \tilde{p}$ can have at most $r - 1$ roots). The reconstruction procedure outputs $\tilde{s} = \tilde{p}(0)$. Since p and $\tilde{p}$ are the same polynomial, $\tilde{s}$ must equal s, proving that this is an r-threshold scheme.

We now argue SIM-security. Without loss of generality it is enough to handle subsets S of exactly $r - 1$ parties. The simulator Sim_S outputs uniformly random $\mathbb{F}_q$-elements for all parties in S. We

need to prove that this distribution is identical to the distribution of the actual shares of the parties in S. These parties observe the values $X_i = p(a_i)$ for all $i \in S$, where p is a random degree-$(r-1)$ polynomial subject to $p(0) = s$. Since a polynomial of degree $r-1$ is uniquely determined by its values on any set of r inputs, there is exactly one choice of p that satisfies the conditions $p(0) = s$ and $p(a_i) = X_i$ for all $i \in S$, so all views by the parties in S occur with the same probability q^{-r+1}. □

The proof of Theorem 9 shows the existence of the desired secret sharing scheme, but does not explicitly describe the reconstruction algorithm, namely the procedure that reconstructs the polynomial p from its evaluations $p(a_i)$. The polynomial can be recovered by solving a system of linear equations in which the unknowns are the coefficients. Alternatively, the value $p(0)$ can be calculated directly by the Lagrange interpolation formula

$$p(0) = \sum_i p(a_i) \prod_{j \neq i} \frac{a_i}{a_i - a_j}$$

where the sum ranges over the indices i for which the value $p(a_i)$ was not erased.

4 Encryption

Going back to secure file storage, what happens if the client wants to store her file on a single server? It appears that the client should be forced to reveal the whole file to the server. But what if the client is allowed to keep a small amount of local information—a *secret key*—that would help her encode the file so it is unintelligible to the server but readable by herself?

This is the problem of *encryption*: Alice wants to send a message to Bob in a way that keeps the message secret from Eve. In the above example, Alice and Bob both play the role of the client, while Eve is the server. Candidate encryption schemes predate computers by millenia.

The Roman emperor Julius Caesar used the following cipher: Shift the position of every letter of the alphabet by three symbols forward. So `a` becomes `d`, `b` becomes `e`, and so on. One of Caesar's famous quotes is "I came, I saw, I conquered", which in his native Latin would have been:

`venividivici`

(blank spaces omitted), instead he would write

`yhqlylglylfl`.

Needless to say Caesar's cipher can be easily broken.[1] In fact his scheme has a fundamental flaw which is unrelated to the specific choice of cipher: it is *deterministic*, i.e. there is no secret key. A deterministic encryption scheme can never be useful: If Alice "encrypts" a message to Bob and Bob can decrypt it, then so can Eve because in the absence of "secret information" possessed by Bob, Eve can impersonate Bob in any interaction.

Here is a randomized variant of Caesar's scheme. If we identify the English/Latin alphabet symbols with the numbers 0 to 25, then Caesar's cipher is $x \to x + 3 \mod 26$. To *decrypt*, one applies the inverse transformation $y \to y - 3 \mod 26$. One thing Alice and Bob can do is to agree on a secret key $i \in \{0, \ldots, 25\}$ and apply the encoding $x \to x + i \mod 26$ instead. So now the encrypted message may look like this:

[1] Perhaps it made *some* sense to use this cipher in Roman times: If most of the population was illiterate, yet they could recognize certain words by sight, shifting things a bit may have made it more difficult.

```
nwfanavanaua.
```

This scheme is somewhat better than the previous one. If Eve now sees a *ciphertext* sent from Alice to Bob, she cannot decode it, because she does not know the key. However, she knows that the ciphertext could have come from at most 26 possible messages. She can now write out the list of these 26 messages. Chances are at most one of them will make any sense, in which case she has broken the scheme again.

However, notice that breaking the new cipher requires Eve to do 26 times the amount of work it took her to break the original one. This suggests that we could make the cipher harder to break by using a longer key. For example, instead of a one-letter key we could use a four-letter key like this:

```
    venividivici
+   skltskltsklt
------------------
    noybnsobnsnb
```

Another scheme could work like this: To obtain the ciphertext, we still substitute symbols one by one, but instead of a cyclic shift, the public key is now an arbitrary permutation P of the alphabet, like

x:	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
$P(x)$:	l	t	r	e	w	v	s	g	m	q	u	d	f	h	i	o	b	z	x	c	p	y	k	j	a	n

and the encoding is obtained by applying the transformation $x \to P(x)$ to each symbol separately.

In general, it seems reasonable that the longer you make the key, the harder it becomes to break the scheme; and in fact for short messages, these schemes may be difficult to break. However, when we try to use them to encrypt longer messages, the schemes become easier to break. In fact the message can be recovered without too much effort by analyzing the frequency of occurrences of the different symbols (or pairs or triplets of consecutive symbols) and matching them to the frequencies in standard text.

Even with short messages, the ciphertext could give us various clues about the message. For example, notice that the second and third block of four letters differs in exactly one position, which says the same must be true of the plaintext. If the message is a list of directions to the hideaway of a terrorist, like

```
leftrightleftleftrightleft
```

then it should not be hard to convince yourself that in any of these schemes, the message can be easily recovered. For more examples of historical encryption ciphers and ways to break them take a look at Chapter 1 of the textbook *Introduction to modern cryptography.*

Security by obscurity One implicit assumption that we made is that the encryption and decryption algorithms – the procedures used to encode and decode messages – are public and therefore known to the adversary. Shouldn't we be able to achieve better security by keeping the algorithms hidden from the adversary?

Most cryptographers believe that, in fact, the opposite is true: We can only gain more confidence in the security of the scheme by making its description public. In practice, this will invite more people to study the scheme and uncover its vulnerabilities. As a consequence, if after a long period

of study no vulnerabilities are found, this should give us more confidence that the scheme is indeed secure.

A historical example of a "secret" encryption scheme are the *enigma codes* used by the German military in the 1930s. This scheme was implemented by an elaborate machine whose mode of operation was kept secret. It took years to break it by a group of Polish crackers. In World War II a new version was deployed, but this one was broken too by scientists in Britain including Alan Turing.

A more technical reason, at least in private-key cryptography, is that essentially nothing is gained by making the algorithms of the scheme secret, as the description (i.e. the code) of these algorithms can always be provided as part of the private key.

5 The one-time pad

If we want to improve the security of encryption, using longer keys seems like a good idea. Taken at an extreme, if Alice and Bob have a random key K that is as long as the message M to be communicated, Alice can encode M using the *one-time pad*

$$C = M + K = (M_1 + K_1, \ldots, M_k + K_k)$$

where you can think of M and K as k-bit strings and addition as bitwise XOR. To recover the message Bob computes $C + K = M$. This scheme should be impossible to break: Eve can observe is the string $C = M + K$, and because K is random, from Eve's point of view, the ciphertext is *statistically independent* of the message. Alternatively, Eve can simulate her view in the interaction between Alice and Bob by generating a truly random string C.

The one-time pad requires Alice and Bob to agree on a key that is as long as the message they are planning to exchange. This key is not reusable, If Alice wants to send a different message to Bob, she must use a different key. In effect, this requires Alice and Bob to agree in advance on a key which is as long as the length of the messages they are ever going to exchange. This is impractical in many applications. Even if a method of distributing the keys can be arranged (say by physically shipping them, which has been done in the past), it may be difficult to get hold of enough randomness to generate such long keys.

Can we get away with using shorter keys? If the objective is to have a ciphertext that is statistically independent of the message, then the answer is no. The intuitive reason is that the key has fewer "bits of information" than the message, so the ciphertext cannot hide the message completely.

Here is a way to make this reasoning more rigorous: Suppose Alice wants to send an m-bit message to Bob, but they have only exchanged a key of length k. There are 2^m possible messages, and they can be encoded using 2^k different keys. So any ciphertext C could have arisen from at most 2^k different messages. Let M be one of these messages. Since $k < m$, there must be some message M' that does not encode to c under any key. So from the perspective of Eve, the ciphertexts of M and M' are *distinguishable*: While M can be encrypted into C with some probability (over the choice of key), M' can never be encrypted to C.[2]

But perhaps we do not need to worry so much: Although *in principle* Eve can always distinguish between the encryptions of M and M', in practice it may be very difficult for her to do so. Perhaps the only way to figure out if C can arise as the encryption of a message M is to go over all possible

[2]This argument assumes that the encryption and decryption algorithms are *deterministic*, while in practice they can be randomized. It can be extended to handle randomized algorithms.

keys and check if any of them works. This would take an amount of work proportional to 2^k, which even for reasonably small values of k, say $k = 128$, is forbiddingly large.

In conclusion, even though we cannot achieve *perfect* security when the key is shorter than the message, there is hope: Eve may not be able to break the encryption simply because it takes too much work for her to do so.

6 Definitions of encryption and security

We now define encryption, starting with the functionality requirement:

Definition 10. A *private-key encryption scheme* for messages of length m and key length k is a pair of algorithms (Enc, Dec) such that for every key K of length k and message M of length m, $Dec(K, Enc(K, M)) = M$.

We now want to define security. Here is the indistinguishability-based definition:

Definition 11. (Enc, Dec) is *perfectly secure* if for every k and every pair of messages M, M' of length m, the random variables $Enc(K, M)$ and $Enc(K, M')$ are identically distributed, where K is chosen uniformly at random from $\{0,1\}^k$.

As we discovered, if a private-key encryption scheme is perfectly secure, then $k \geq m$. We will now consider ways to relax the definition of perfect security in a way that may allow us to bypass this restriction.

One thing we can do is relax the requirement that the two distributions are identical. Instead, we can merely require that they are close to one another. What does it mean for two distributions to be close? There are various measures of distance between distributions, but the one that is most relevant for us is *statistical distance*. Two distributions are statistically close if no observer can distinguish samples coming from one distribution and samples coming from the other one with good confidence.

Definition 12. Random variables X and X' are *ε-statistically close* if for every function D that outputs 0 or 1, $|\Pr[D(X) = 1] - \Pr[D(X') = 1]| \leq \varepsilon$.

We think of the function D as a *distinguisher*: It tries to tell apart the random variables X and X' as well as it can. Perfect security requires that for every pair of messages M and M' the random variables $X = Enc(K, M)$ and $X' = Enc(K, M')$ are perfectly indistinguishable (i.e., 0-statistically close). What if we relax this impossible to satisfy requirement to, say, $\varepsilon = 0.01$? It turns out that it is still impossible to achieve: As long as the key is shorter than the message, there will at least two messages M and M' so that the statistical distance between $Enc(K, M)$ and $Enc(K, M')$ is at least $1/2$. In the next lecture we will consider a further relaxation of this definition that makes encryption with short keys possible.