CSCI 5440: Cryptography **Lecture 1**
The Chinese University of Hong Kong 12 September 2012

Cryptography is the science that allows us to securely achieve various objectives while operating in an insecure environment. Let's try to picture some concrete scenarios where this may be desirable:

1. You are on the phone with your overseas friend and want to share a bit of gossip about your roommate, but you are worried that the roommate may be listening in on the conversation.
2. Your have a wireless router that you always use to access the internet, but your upstairs neighbor is also within the range of your router. You are worried about two things. Your neighbor may be snooping in on your internet traffic. Or he could be making use of your connection without contributing anything.
3. You and your nine friends want to have a lucky draw for Chinese New Year – the winner gets a prize. However you are all in different places around the world, so you can only do it on the internet. Someone suggests to perform the draw at random, but you are all very suspicious that he will miraculously emerge as the "lucky" winner. Can you figure out a way to do a fair draw of a random winner?
4. On election day Sunday you receive a phone call from your boss. He tells you to vote for his friend, otherwise he'll cut your pay. You had other plans in mind. Should you trust your boss will not find out who you voted for? Do you know that your vote will be counted properly?

While these scenarios are very different, they all concern some objective to be achieved by legitimate parties via interaction, but in a way that prevents malicious parties from gaining advantage by interfering with the interaction in unintended ways. The objective is, by its nature, conflicting: It needs to be easy to achieve for legitimate users, but difficult to attain for the malicious ones. However, the participants are all people, or computers, so they all have access to essentially the same amount of resources. If anything, malicious users are often willing to spend *more* resources. (For example, a credit card holder expects to complete his online payment securely in a few seconds, while a criminal may be willing to spend weeks or months attempting an illegitimate purchase.) How can you make the same task easy for one group of users (the legitimate ones), and hard for another one (the cheating ones)?

As it should already become apparent from these examples, there is wild variation in the tasks to be achieved, modes and media of interaction, and adversarial roles. Some take place over the phone, some on the internet, some in the physical world. The malicious users also vary greatly in power. For example, in the first scenario it may be reasonable to assume that the adversary is just a passive eavesdropper. In the fourth scenario, some voters may be trying to cheat the system, for example, by casting multiple votes. Voting officials may also try to cheat, say by revealing the privacy of the voters.

One highlight of cryptography is that is has been able to reduce these seemingly incomparable problems to a small number of *cryptographic primitives*, namely simple functionalities that, when used and combined properly, can provide secure solutions to a wide variety of problems.

A preview of some things to come

The simplest cryptographic tasks involves two participants, called Alice and Bob who are interacting over an insecure channel. An adversary, called Eve, is tampering with the channel in some way; she may be passively listening to the channel, or actively participating in the exchange. Alice wants to send some message to Bob in a way that will prevent Eve from "messing" with it.

One objective that is desirable to achieve is **privacy**. Roughly speaking, privacy means that while Bob can completely recover the message, Eve does not find out any information about it. To achieve this, at least Bob must have some secret called the *key* that Eve doesn't have. If Bob doesn't have a key, then Eve can *simulate* Bob. At every step of the interaction she does whatever Bob would do, so by the end of the day she will have learned everything Bob has learned himself, in particular the message from Alice.

In private-key cryptography the secret key is some common information shared by both Alice and Bob. Typically the key is a random string that Alice and Bob agree on in advance which is not known to Eve. Using this key, it is often possible to hide information from Eve which makes her task impossible, or at least very difficult. In all but the simplest scenarios, figuring out what to do with the key is far from obvious. In the first couple of lectures we will see that unless we make some assumptions about the *computational power* of Alice, Bob, and most importantly Eve, even with secret keys very sensible notions of privacy are impossible to achieve. But if we are willing to make some seemingly reasonable assumptions of this kind, then not only privacy but a lot more will become possible.

Sometime in the late 1970s cryptographers came to the surprising realization that even without sharing keys in advance, private communication may be possible. In public-key cryptography, there is a pair of keys, a private key and a public key. The public key is known to everyone, including Eve, while the private key is known to only one of the participants. This allows for applications where Alice and Bob can communicate privately without having met in advance: Bob creates a pair of keys, keeps the private key to himself and distributes his public key to all parties involved. Alice can then (in certain settings and under certain assumptions) encode the message to Bob using his public key in a manner that is unintelligible to Eve, but perfectly understandable Bob.

Clearly public key cryptography is the more desirable alternative. So why do we bother to study the private key variant? One reason is that the private key setting is simpler, so it will allow us to introduce our fist cryptographic concepts – definitions of security, the role of computational assumptions – without having to worry about other complications. A more important reason is that although public-key cryptography is more powerful, this power comes at a price. From a practical perspective, public-key schemes are more difficult to design, so they tend to be less efficient and possibly easier to break. From a theoretical perspective, there is evidence that public-key cryptography requires us to make strictly stronger assumptions about the computational power of the adversary. We may want to avoid making such assumptions, unless it is absolutely necessary.

Another desirable objective in many communication scenarios where secrecy is involved is **authentication**. When Alice sends a message to Bob, Bob wants to be sure the message was indeed send by Alice and not by Eve. Although authentication is a very different objective from privacy, the tools we develop for achieving privacy will turn out to be very useful for obtaining authentication without too much effort, once we have understood what it is that we want to achieve.

Beyond the basics Privacy and authentication are simple cryptographic tasks. However, the conceptual and technical tools that are used to achieve them can solve much more complicated problems. Preserving privacy and certifying authenticity of messages are very special cases of the problem of *secure mulitparty computation.* This is a general setting where a number of parties, some honest and some malicious, what to jointly perform some computational task in a way that keeps each participant's data private.

In the 1980s cryptographers gave a general definition of secure multiparty computation and gave a sound blueprint on how it can be accomplished. This is one of the holy grails of theoretical cryptography: It says that under very general conditions, any computation that many parties want to do jointly can be performed securely.

The setting of multiparty computation is general enough to capture many scenarios of interest. In voting, the parties are the voters, each of which inputs a vote, and the output is, say, the identity of the candidate that collected the largest number of votes. In an auction, the parties are the bidders, their inputs are their bids, and the winner is determined by the auction mechanism (e.g. the highest bidder wins). Theoretical cryptography gives a way to obtain secure implementations of any such scenario.

While these tools are extremely powerful, one reason they are not more widely used in practice is that they are somewhat inefficient. In recent years researchers have made progress in coming up with reasonable implementations, especially for specific computations like auctions.

Cryptography is a vast subject with connections to other areas, and there is a number of exciting topics that we may not have enough time to address. If time permits we will touch upon one of these topics near the end of the course.

- **Database privacy** In cryptographic settings we often expect to achieve absolute privacy, that is no infomation about the secret data ought to be leaked to the malicious parties. There are settings in which privacy is desirable but absolute privacy is impossible to achieve. For example, a database may contain sensitive data about individuals. For the database to be useful we must allow some queries to it, but these queries may reveal unintended private information. Concepts from cryptography can help us reason about what kinds of tradeoffs between privacy and utility are achievable in such settings.

- **Cryptography and game theory** Both cryptography and game theory analyze scenarios in which multiple participants interact in a protocol or game. Cryptography usually takes a manichean view where the participants are either totally legitimate (they always follow the rules) or completely malicious (they can cause damage in unpredictable ways). In game theory participants are not good or bad but selfish, namely they do whatever is in their best interest. In some scenarios both perspectives come in useful. For example online auctions demand secure implementations of game-theoretic protocols. In the other direction, cryptogaphic parties may need to be given certain incentives to participate and act honestly in the protocol.

- **Cryptography and computation** Virtually all cryptographic protocols require us to make some computational assumption about the strength of the adversary. A principal objective of cryptography is to *reduce* the security a of construction to a small number of well-studied assumptions. But some of these assumptions are stronger than others. In general we would like to know what is the weakest possible assumption that enables a given cryptographic task.

The role of definitions

In cryptography, we always begin with a rigorous definition of the task we are trying to achieve. A cryptographic definition always consists of two parts: a *functionality requirement* and a *security requirement*. The functionality requirement describes the task that the legitimate parties want to achieve, while the security requirement specifies the kinds of things that we want to prevent the adversary from doing.

While definitions are arguably important in all fields of computer science, this is nowhere more so than in cryptography. For example, if we are looking at algorithms for maximum flow, the concept of "blocking flow" may be important because once we know it, the task of designing the algorithm becomes easier (maybe we try to greedily improve blocking flows) and the correctness of the algorithm is more transparent. However, even without knowing this concept, we would still be able to design and argue correctness of our algorithms, albeit in a more roundabout way.

In contrast, without definitions, we cannot even begin to think about cryptography. For instance consider scenario 1 above. We want to have some way of communicating over the phone securely, but what does "securely" even mean? It depends a lot of what the adversary is allowed to do. Perhaps every time Eve picks up the phone, Alice hears a little "click" and at that point she stop talking about Eve until she is out of the house. Perhaps there is no "click", but Alice can talk in French which Eve doesn't understand. But maybe Eve will use a tape recorder and she will later ask someone to translate the conversation for her. Or maybe Eve has made a recording of Alice last time Alice and Bob talked on the phone and today she will replay that recording pretending she is Alice. Or maybe Eve will point a gun to Alice's head and actively control what she says to Bob.

Then there is the question of what it is exactly that we are trying to prevent Eve from finding out. Certainly we do not want Eve to find out all the details about the gossip, but maybe even merely saying Eve's name over the phone will arouse her suspicion. Maybe even the absence of any reference to Eve in the conversation will arouse her suspicion: Eve will find this unnatural and begin to suspect that she is being talked about behind her back at other times. How can we even begin to think about all these possibilities, and hundreds of other ones that we haven't even considered?

The point is that, unlike in most of computer science where we are only concerned with legitimate users of whatever system we are designing, in cryptography we want the scheme to be secure against *all* "reasonable" behaviors of the adversary. But we cannot start designing such a system until we know what a reasonable behavior is.

A good definition can capture all these requirements in a succinct manner. This succinctness is important because it not only brings clarity of thought to the problem at hand, but it also allows us to reason rigorously about it. This reasoning often leads to surprising results. For example, an adversary that can participate in the communication (let's call her adversary E) is clearly more powerful than one that can merely observe it (adversary F). So a definition that guarantees security against adversary E should intuitively be harder to achieve than one that only concerns adversary F. However, surprisingly often in cryptography it turns out that if you can achieve security against E, you can also achieve security against F! But unless we have a precise understanding of the power of E and F, there is no way to describe this phenomenon, let alone explain it.

Another important aspect of cryptographic definitions is that they implicitly describe the limitations of the security model we have in mind, and of cryptographic solutions to security problems in general. This is especially important when we want to deploy cryptosystems in the real world. It could be that existing cryptographic solutions are too weak for the task at hand, yet stronger ones are impossible to achieve. In such cases system designers may need to resort to non-cryptoraphic solutions, such as economic incentives or legal measures. One such example is *copy protection*: There is no reasonable cryptographic solution to the problem of illegitimate copying of data, like music sharing.

1 Some attempts at encryption

Let us begin our study with the problem of *encryption*: Alice wants to send a message to Bob in a way that keeps the message secret from Eve. This is a task that people have been interested in way before the discovery of computers.

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:

`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.

[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.

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:

$$\begin{array}{rl} & \texttt{venividivici} \\ + & \texttt{skltskltsklt} \\ \hline & \texttt{noybnsobnsnb} \end{array}$$

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. The breaking of the Enigma codes is a fascinating story, and we may come back to it to motivate some cryptographic definitions.

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.

2 The one-time pad

As we observed, if we want to improve the security of encryption, it seems a good idea to use a longer key. Let's take this idea to an extreme. Suppose Alice wants to send Bob a message m consisting of k symbols. The *one-time pad* is the following encryption scheme: The private key is a random string r of k symbols. Alice sends the message

$$c = m + r = (m_1 + r_1, \dots, m_k + r_k)$$

and to recover the message, given ciphertext c, Bob computes $c - r$.

This scheme should be very difficult to break: What Eve can observe is the string $c = m + r$, and because r is random, from Eve's point of view, the ciphertext is *statistically independent* of the message. This requirement on the distribution of the ciphertexts is known as *perfect security*; we will define it formally below.

The problem with this scheme is, of course, that it 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 on a memory stick), it is 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. In short here is why. Let's assume (as we shall from now on) that the alphabet in which both messages and ciphertexts are written is $\{0, 1\}$. Suppose Alice wants to send an n-bit message to Bob, but they have only exchanged a key of length k. There are 2^n 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 $k < n$, 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 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 forbidding.

[2]This argument makes one unreasonably simplistic assumption: It assumes that the encryption and decryption algorithms are *deterministic*, while in practice they can be randomized. It is not too difficult to extend the argument so that it applies to randomized encryption and decryption algorithms as well.

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

3 Definitions of encryption and security

Let us try to formalize the discussion on encryption based on our intuition about the one-time pad.

Recall that any cryptographic definition has two components: A *functionality requirement*, which essentially describes the problem we are trying to solve and the behavior of the honest parties – Alice and Bob, and a *security requirement*, which describes the security guarantees we want to obtain against the adversary.

In general, the functionality requirement is easier to state. Here we have Alice who wants to encrypt a message M using a key K. The functionality requirement says that when Alice and Bob share the same key, decryption does the opposite of encryption.

Definition 1. A *private-key encryption scheme* for message 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, $Dec(K, Enc(K, M)) = M$.

One annoying thing about this definition is that it only works for a fixed message length and key length. What we really want is a single algorithm that we can apply to messages of arbitrary length, provided the key is long enough:

Definition 2. A *private-key encryption scheme* is a pair of algorithms (Enc, Dec) and a function $m(k)$ such that for every integer k, every key K of length k and every message m of length $m(k)$, $Dec(K, Enc(K, M)) = M$.

Here k is called the *security parameter*: By making it larger, we hope to encrypt longer messages, and at the same time make the task of breaking the encryption harder for the adversary.

We now want to define security. Recall the kind of security we get from the one-time pad: For a random key, the distribution of the ciphertext is completely independent of the message that was encrypted. One way to say this is that for every pair of messages, the distributions of ciphertexts are identical.

Definition 3. A private-key encryption scheme (Enc, Dec) is *perfectly secure* if for every k and every pair of messages M, M' of length $m(k)$, 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 $k \geq m(k)$. 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*. Informally, two distributions are close in statistical distance if no observer can distinguish samples coming from one distribution and samples coming from the other one very often. More formally, we say the distributions of two random variables X and X' are ε-close in statistical distance if

$$|\Pr[A(X) = 1] - \Pr[A(X') = 1]| \leq \varepsilon$$

for every function A.

So perhaps instead of asking that $Enc(K, M)$ and $Enc(K, M')$ in the definition of perfect security are identically distributed, we can merely require that they are ε-close in statistical distance for some small ε. Unfortunately, this is not good enough: It turns out that 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 1/2 or more.