ISA 562: Information Security, Theory and Practice

Lecture 1

1 Encryption schemes

1.1 The semantics of an encryption scheme.

A symmetric key encryption scheme allows two parties that share a secret key to encrypt and decrypt messages. Such a scheme consists of 3 algorithms, one for generating the secret key that they will share, one for encrypting a message using that key, and one for decrypting a ciphertext, using the same key. Associated with an encryption scheme are 3 sets, which we call the keyspace, $\mathcal{K}$, the message space, $\mathcal{M}$, and the ciphertext space, $\mathcal{C}$. The three algorithms have the following input/output behavior.

$\mathsf{Gen}(\kappa) \to k$ takes as input a security parameter[1] κ and outputs the secret key $k \in \mathcal{K}$.
$\mathsf{Enc}(k, m) \to c$: on input a secret key $k \in \mathcal{K}$, and a message $m \in \mathcal{M}$, it outputs a ciphertext $c \in \mathcal{C}$.
$\mathsf{Dec}(k, c) \to m$: on input a secret key $k \in \mathcal{K}$, and a ciphertext $c \in \mathcal{C}$, it outputs a message $m \in \mathcal{M}$.

We say that such a scheme has *correctness* if for all $m \in \mathcal{M}$, for all $k \in \mathcal{K}$, $\mathsf{Dec}(k, \mathsf{Enc}(k, m)) = m$. Note that even if an encryption scheme has correctness, it might not be secure. We address security below, but first give a few examples of encryption schemes (both secure and insecure).

1.2 Examples of insecure encryption schemes

An insecure shift cipher. We start with an encryption scheme that has keyspace $\mathcal{K} = \{0, \ldots, 25\}$, message space $\mathcal{M} = \{a, \ldots, z\}^*$, and ciphertext space $\mathcal{C} = \{A, \ldots, Z\}^*$. However, it will be convenient to also treat message and ciphertext characters as integers between 0 and 25. This is done in the obvious way, with 'a' and 'A' both mapping to 0, 'b' and 'B' both mapping to 1, and so forth. The three algorithms in this scheme are as follows.

$\mathsf{Gen}()$: sample a random value $k \leftarrow \mathcal{K}$.

$\mathsf{Enc}(k, m)$: parse m as $m = m_1 m_2 \ldots m_\ell$, where $\ell = |m|$ is the length of the input message. For each $i \in \{1, \ldots, \ell\}$, let $c_i = m_i + k \pmod{26}$.

$\mathsf{Dec}(k, c)$: parse c as $c = c_1 c_2 \ldots c_\ell$, where $\ell = |c|$ is the length of the input ciphertext. For each $i \in \{1, \ldots, \ell\}$, let $m_i = c_i - k \pmod{26}$.

To give an example, if $k = 3$ and $m = zoo$, then $m_1 =$ 'z', $m_2 =$ 'o', and $m_3 =$'o'. Treating these characters as integers, and performing addition, we have $c_1 = 3 + 25 \pmod{26} = 28 \pmod{26} = 2 \pmod{26}$. Similarly, $c_2 = 3 + 15 \pmod{26} = 18 \pmod{26}$. Since 2 corresponds to 'C' and 18 corresponds to 'R', $\mathsf{Enc}(3, \text{'zoo'}) =$'CRR'.

[1] Actually, for technical reasons, this is usually written as $\mathsf{Gen}(1^\kappa)$, where, recall, 1^κ is a string of κ 1's. The reason for doing it this way is because the runtime of an algorithm is measured as a function of it's input size. So this notation indicates that Gen is allowed to be polynomial in κ, rather than polynomial in $\log \kappa$. But we will be sloppy and allow ourselves to ignore this detail.

Even though we have yet to define security, we can easily see that this scheme is not secure. Suppose an adversary sees the ciphertext $c = EPH$. What plaintext might this correspond to? A brute-force over the keyspace leaves only 26 possible plaintexts! If the key were $k = 4$, then $\mathsf{Dec}(k, c) = ALD$. If $k = 3$, then $\mathsf{Dec}(k, c) = BME$. If $k = 2$, then $\mathsf{Dec}(k, c) = CNF$. If $k = 1$, then $\mathsf{Dec}(k, c) = DOG$. Bingo! This is very likely the plaintext, as the other options aren't in the english dictionary.

The insecure mono-alphabetic substitution cipher. In this encryption scheme, the keyspace is the set of all permutations on $\{a, \ldots, z\}$. We might write this formally as $\mathcal{K} = \{\pi \mid \pi : \{a, \ldots, z\} \to \{a, \ldots, z\} \text{ is a bijection}\}$. Note that the size of the keyspace is 26!. We can represent a permutation however we like: a natural way is to just store an array of the 26 characters, treating the first element of the array as the value that is mapped to by 'a', the second element as the value that is mapped to by 'b', etc. For example, if we have key $k =$ cgefaqwityruponmzlxvsdhjbk, then our permutation maps 'a' to 'c', 'b' to 'g', 'c' to 'e', 'd' to 'f', 'e' to 'a', etc. Since a permutation is a function, we will use notation that suggests that: $k(a) = c$, $k(b) = g$, etc. Similarly, we can consider the inverse function: $k^{-1}(e) = c$. The message space and ciphertext space are the same as before, consisting of arbitrary length strings over the alphabet: $\mathcal{M} = \{a, \ldots, z\}^*$, and $\mathcal{C} = \{A, \ldots, Z\}^*$. The scheme is as follows.

$\mathsf{Gen}()$: sample a random permutation k on $\{a, \ldots, z\}$.
$\mathsf{Enc}(k, m)$: parse $m = m_1 \ldots m_\ell$, and for $i \in \{1, \ldots, \ell\}$, output $c_i = k(m_i)$.
$\mathsf{Dec}(k, c)$: parse $c = c_1 \ldots c_\ell$, and for $i \in \{1, \ldots, \ell\}$, output $m_i = k^{-1}(c_i)$.

For example, using the same k as above, $\mathsf{Enc}(k, dog) = FNW$, and $\mathsf{Enc}(k, cat) = ECV$. Note that a brute-force attack is now much harder to perform, because the keyspace is $26! \approx 4 \cdot 10^{26} \approx 2^{88}$ (i.e. it has 88 bits of security). Also, this scheme has another nice property. Suppose the adversary were given FNW and wanted to figure out the plaintext. Even if he could search the full keyspace and find k such that $\mathsf{Dec}(k, FNW) = dog$, note that there also exists $k' =$ ngfeaqvityrupocmzlxwsdhjbk such that $\mathsf{Dec}(k', FNW) = cat$, so, for certain short ciphertexts – specifically, ciphertexts that have no repeating characters – a brute-force attack does not help. We'll return to this point again below.

Nevertheless, this scheme is also insecure. Consider the ciphertext $\mathsf{Enc}(k, zoo) = KNN$. There is no key in the keyspace that would encrypt 'dog' or 'cat' to KNN. An adversary that sees this ciphertext can immediately learn something about the plaintext; namely that the last two characters of the plaintext repeat. Interestingly, note that even if an adversary is willing to try 2^{88} keys, and even if he is given some small number of plaintext / ciphertext pairs (such as (dog, FNW), (cat, ECV)), he cannot possibly recover the exact key. This is because there are many keys that would encrypt 'dog' to 'FNW' and 'cat' to 'ECV'. (Count them!). Despite that, the scheme is clearly not secure. So the size of the keyspace is not, by itself, a good measure of security.

2 Perfectly secure encryption

A pefectly secure shift cipher. We now modify the shift cipher described above, and show that the resulting scheme has perfect security (which we will define below). We modify the message and ciphertext space, forcing both to have some fixed upper bound on their length, which we'll call ℓ.

(ℓ can be any integer.) We also modify the keyspace, allowing keys to have length ℓ. Intuitively, we will use a different shift value for each character of the message. Formally, $\mathcal{K} = \{0, \ldots, 25\}^\ell$, $\mathcal{M} = \{a, \ldots, z\}^\ell$, and $\mathcal{C} = \{A, \ldots, Z\}^\ell$.

$\mathsf{Gen}()$: sample a random value $k \leftarrow \mathcal{K}$. Below we will use k_i to denote the ith element in the key.
$\mathsf{Enc}(k, m)$: parse m as $m = m_1 m_2 \ldots m_\ell$. For each $i \in \{1, \ldots, \ell\}$, output $c_i = m_i + k_i \pmod{26}$.
$\mathsf{Dec}(k, c)$: parse c as $c = c_1 c_2 \ldots c_\ell$. For each $i \in \{1, \ldots, \ell\}$, output $m_i = c_i - k_i \pmod{26}$.

To give an example, if $\ell = 3$, $k = 3|15|18$ and $m = zoo$, then we have $c_1 = 3 + 25 \pmod{26} = 28 \pmod{26} = 2 \pmod{26}$. Similarly, $c_2 = 15 + 15 \pmod{26} = 4 \pmod{26}$, and $c_3 = 18 + 15 \pmod{26} = 7 \pmod{26}$. Since 2 corresponds to 'C', 4 corresponds to 'D', and 7 corresponds to 'G', we have $\mathsf{Enc}(k, zoo) = CDG$. Intuitively, because we are now using a separate shift value for every character in our plaintext, we've removed any correlation between characters in the ciphertext. So, if we want to think about what a ciphertext reveals to the adversary, we can consider encryptions of single characters. (When formally define and prove security of this scheme below, we will not do that. But it is helpful for the intuition.) Looking at a single character of the ciphertext, c_i, there is no way to learn anything about plaintext character m_i, because for *any* possible value, there is exactly 1 key that maps that character to c_i. Each of these key values were equally likely to have been chosen during key generation, so there is no way for the adversary to know which one it was, and consequently, no way to learn anything about the plaintext character. We now formally define perfect security of an encryption scheme.

2.1 Perfect security

Following the above intuition, we want our security definition to capture the fact that a ciphertext reveals nothing at all about the plaintext. For starters, our scheme must allow any ciphertext to be "explained" as an encryption of any plaintext. Technically, this would mean that for every $m \in \mathcal{M}$ and every $c \in \mathcal{C}$, there exists some key k such that $\mathsf{Dec}(k, c) = m$. In fact we need something a bit stronger: suppose that for some ciphertext c, there are 3 keys that decrypt c to plaintext 'cat', but there is only 1 key that decrypts c to 'dog'. Then, statistically, it is more likely that the encrypted value was 'cat'! (We are assuming that each of these keys was equally likely to be chosen during key generation.) So, not only should we be able to explain every ciphertext as the encryption of any plaintext, we want that for any fixed ciphertext c, every plaintext should have the same number of keys that map it to c. Put another way, if an adversary sees an encryption of 'dog', it should be the case that it is equally likely to be an encryption of 'cat', or 'fox', or any other 3 letter word. We now give a formal definition.

Definition 1 *An encryption scheme* $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ *with keyspace* $\mathcal{K}$*, message space* $\mathcal{M}$*, and ciphertext space* $\mathcal{C}$*, is perfectly secret if for every* $m, m' \in \mathcal{M}$*, and every* $c \in \mathcal{C}$*,* $\Pr[\mathsf{Enc}(k, m) = c] = \Pr[\mathsf{Enc}(k, m') = c]$*, where the probability is over the choice of* $k \leftarrow \mathcal{K}$*.*

We claim that the shift cipher described above is perfectly secure. To see this, note that for any $m \in \mathcal{M}$ and any $c \in C$, there is exactly one key k such $\mathsf{Enc}(k, m) = c$. There are a total of 26^ℓ keys in the keyspace. (Count them!) Therefore, for any pair of messages, m, m' and any ciphertext c, $\Pr[\mathsf{Enc}(k, m) = c] = \Pr[\mathsf{Enc}(k, m') = c] = 1/26^\ell$.

The one time pad (OTP). We give one more example of a perfectly secure encryption scheme. This is a very famous example, and will show up throughout the next few lectures. Actually, it is exactly the scheme we presented at the end of the previous subsection, but using a binary alphabet instead of a 26 character alphabet. $\mathcal{K} = \mathcal{M} = \mathcal{C} = \{0,1\}^\ell$.

$\mathsf{Gen}()$: sample a random value $k \leftarrow \mathcal{K}$.
$\mathsf{Enc}(k, m)$: output $k \oplus m$.
$\mathsf{Dec}(k, c)$: output $k \oplus c$.

Here, $\oplus$ denotes the bitwise $\oplus$. So, for example, if $\ell = 5$, $k = 10110$ and $m = 01100$, then $\mathsf{Enc}(k, m) = 11010$. Verify for yourself that $\oplus$ is addition modulo 2, and you'll see that this is the same encryption scheme as before. The proof of security is also the same, though you should verify for yourself that for every $m \in \mathcal{M}$ and every $c \in C$, there is exactly one key k such that $k \oplus m = c$. (In fact, that key is easily described as $k = c \oplus m$.)

Returning briefly to the mono-alphabetic substitution cipher, it is interesting to note that if we modified our plaintext space to only include strings that do not have any repeating characters, then we could in fact prove that the construction were perfectly secure. We leave this as an exercise for the reader.

The drawback of perfect secrecy. We stress that perfectly secure encryption cannot be broken, regardless of how much computation time or power the adversary may have. Unfortunately, such encryption schemes have a severe limitation: the key length has to be as large as the plaintext length. In practice, this is a serious drawback, since sharing key material is not always easy, logistically, and we may not always know how large our messages will be at the time that we are able to share. In applications where we aren't concerned by these limitations, the one-time pad is a perfect solution to the problem. But, generally, we would like to develop encryption schemes where the key can be very short (e.g. 256 bits), regardless of how many bytes of data we will encrypt. Sadly, it is impossible to have this property while also ensuring perfect security. We won't quite give a formal proof, but to see the intuition, note that for any fixed ciphertext, c, there are at most $|\mathcal{K}|$ ways to decrypt this ciphertext, resulting in a set of at most $|\mathcal{K}|$ plaintexts. If $|\mathcal{K}| < |\mathcal{M}|$, then there is some message that is not in that set. That is, there is some message m which cannot possibly result from decrypting c, regardless of which key is used. Therefore, upon seeing ciphertext c, the adversary knows, for certain, that the plaintext could not be m, and he has learned something from the ciphertext.

It is worth considering what happens if we ignore this fact, and try to re-use our key in the OTP scheme, in order to encrypt arbitrary length messages. Suppose we wish to encrypt messages $m = m_1||m_2$, where $||$ denotes concatenation. (Equivalently, we can think about encrypting two distinct messages, m_1 and m_2.) $\mathsf{Enc}(k, m) = k \oplus m_1||k \oplus m_2$. Seeing this ciphertext, the adversary can compute $(m_1 \oplus k) \oplus (m_2 \oplus k) = m_1 \oplus m_2$. (Verify this for yourself if it is not clear.) While this does not reveal either m_1 or m_2, it does tell him quite a lot about both of them. In particular, if the ith bit of $m_1 \oplus m_2$ is 1, it means that m_1 and m_2 differ in the ith bit, and if the ith bit is 0, then they were the same in that bit. Depending on what else the adversary happens to know about the plaintext, this could really pose a serious threat. For example, he might learn for certain that the plaintext was not 'eat||cat'.

2.2 Computational security

If we wish to have short keys, and to support the encryption of arbitrary length messages, we will have to allow some relaxations. We will assume a polynomial-time adversary that cannot try every key in the keyspace, or attempt other exponential-time attacks on the scheme. We stress that this does *not* mean we are now only interested in preventing a full key recovery. We will still provide a stronger security definition than that, requiring, roughly, that the adversary learns nothing from a ciphertext. But, because we don't have perfect security, our assumption that the adversary cannot try all keys is a necessary *minimum* assumption.

Before we give the security definition, we comment that the idea of basing cryptography on some computational hardness assumption is relatively new. This approach was first taken in 1978, and has revolutionized the field of cryptography. Cryptographers now have a general methodology for constructing *provably secure* schemes. The methodology looks something like this:
1) Specify some particular computational hardness assumption. E.g. "factoring large numbers is hard."
2) Define some security game (as we will do below).
3) Construct a scheme, relying on the hard problem from 1).
4) "Plug" your scheme into the security game, and prove that if there exists some adversary that can win in the security game, then there exists some adversary that can break the hardness assumption specified in 1).

This last step is called a *security reduction*, as it reduces the problem of breaking the encryption scheme to the problem of solving some hard computational task, which we may have good reason to believe is impossible. We won't learn how to construct such proofs in this class, but the methodology should become somewhat more clear in the next lecture.

In defining security, we want to capture the same idea as before: that ciphertexts reveal nothing about the plaintexts. One difficulty in the computational setting is that we don't know what the adversary already knows. For example, what if he has already seen us encrypt some message yesterday, and today we intend to encrypt the same message? If the encryption is the same both days, we can't really claim that he didn't learn anything from the ciphertext. He would learn that we've sent the same message 2 days in a row, which could reveal quite a lot. Suppose, for example, that after the message was sent yesterday, he was able to learn the message, through observation of the recipient's actions. Today, when he sees the same ciphertext, he will know the message even *before* observing the recipient's actions, which could be catastrophic. We will ensure that our definition covers such attacks, and others.

We define the following security game, which takes place between an adversary and a challenger. The game is generic, in the sense that we can "plug in" any encryption scheme and see how an adversary will fare. The requirement is that NO (polynomial-time) adversary should be able to win in this game with probability significantly better than 1/2. Although it is not so simple to prove that there is no such adversary, what we can do is show that if we plug-in a broken encryption scheme, that there is an adversary that wins in this game with high probability.

<u>IND-CPA:</u>
1) $k \leftarrow Gen(1^n)$
2) $\mathcal{A}$ is given 1^n and oracle access to $Enc_k()$. Outputs (m_0, m_1), $|m_0| = |m_1|$
3) $b \leftarrow \{0,1\}$, $c \leftarrow Enc_k(m_b)$ is given to $\mathcal{A}$.
4) $\mathcal{A}$ continues to have oracle access to $Enc_k()$, and outputs b'

5) the output of the experiment is 1 if $b = b'$.

To come:
I will show that re-use of OTP does not meet this definition.
I will show that NO deterministic encryption scheme can meet this definition.