In the last lecture we examined the possibility of achieving secured communications when facing a computationally unbounded adversary, Eve. In this lecture we continue the analysis, and then introduce the more realistic notion of a computationally bounded adversary. We examine the various ways of modeling limited computational capabilities, and provide an example of a basic cryptographic primitive (the One Way Function), which utilizes the model.

Various material taken from the lecture notes for the graduate version of the course.

1 Definitions of Perfect Security

To re-cap from the end of the last lecture, we proposed the following definition for perfect security, also known as Shannon security:

**Definition 1 (Shannon Security)** Let $M \in \mathcal{M}$ be a random message and $C \in \mathcal{C}$ be the ciphertext of $M$, that is, $C = E_s(M)$. For any $m \in \mathcal{M}$ and $c \in \mathcal{C}$, an encryption system is called perfectly secure in the Shannon sense if from the perspective of the attacker, $\Pr_{M,s}(M = m | C = c) = \Pr_M(M = m)$. This means that Eve’s probability of guessing $M$ remains unchanged after seeing any particular outcome $C = c$.

The intuition here is that a system is Shannon secure if the adversary learns nothing at all from seeing the ciphertext. We will now suggest an alternative definition of perfect security which provides a slightly different intuition.

**Definition 2 (Perfect Secrecy)** An encryption system achieves perfect secrecy if, for any two messages $m_0, m_1 \in \mathcal{C}$ and any ciphertext $c$, we have that $\Pr_s(E_s(m_0) = c) = \Pr_s(E_s(m_1) = c) = \Pr_s(c)$. namely, the probability of observing $c$ does not depend on the message transmitted. Intuitively, this means that Eve cannot tell any information about which of the two messages was sent after seeing the ciphertext.

In the above definition, $\Pr_s(\cdot)$ denotes the probability computed over the entire space of shared secrets $s \in \mathcal{S}$, since Eve does not know anything about the secret, thus all secrets are equiprobable to her. The new intuition provided by this definition is that one can consider a system perfectly secure if an adversary cannot tell which of two messages a ciphertext is more likely to correspond to.

Both of these definitions seem to make sense, so which definition is “right”? Both of them.

**Theorem 3** Shannon Security is equivalent to Perfect Secrecy.

The proof is left as an exercise for the reader (at least convince yourself it is true).
2 PROBLEMS WITH SHANNON SECURITY

There are several practical problems with perfectly secure systems.

a) **Theorem 4 (Shannon Impossibility Result)** For an encryption on messages \( m \in \mathcal{M} \) using a shared secret \( s \in \mathcal{S} \), it is required that \( |S| \geq |M| \) in order to achieve Shannon Security.

This means that the number of bits in the shared secret must be at least as big as the number of bits in the message to being encrypted.

**Proof:** Let \( A \) be the number of messages that can be correctly represented by a particular (valid) ciphertext \( c \). We know that for any fixed secret \( s \), there is at most one message \( m \) which corresponds to \( c \) (in particular, it is \( m = D_s(c) \) by the correctness property of encryption). Thus, we have shown that \( |S| \geq A \). Perfect secrecy requires that Eve must believe that \( c \) could possibly decrypt to any message \( m \in \mathcal{M} \), which implies that \( A \geq |M| \). Putting these two inequalities together, we have \( |S| \geq A \geq |M| \), and the proof is complete.

b) No stateless encryption is possible.

If Bob wishes to send Alice more than one message (using a shared secret at least as long as all the messages in total), both Alice and Bob have to remember some information about their state at the end of the last message. Namely, the encryption cannot be stateless. For example, they can prepare a new one-time pad for each new message, but then they have to be in sync over which message is “currently” transmitted. The formal (and simple) proof of this is given to you in the homework.

c) Key distribution is difficult.

In order for Bob to communicate with Alice, he first had to meet her in person to agree on a shared secret (otherwise, Eve would have eavesdropped on the secret as well). If Bob wants to send secret messages to Carol instead of Alice, he will have to meet her in person and share a totally different secret with her. Bob will have to keep track of all these different secrets he is sharing, each of which must be at least as long as the total length of all messages he wishes to send to that person securely. Distributing these shared secrets to everyone Bob wishes to communicate with, and maintaining a record of them (along with state information, as was discussed in item b) is very troublesome for Bob. Ideally, Bob would like to be able to send secret messages to Alice and Carol even if he never had a chance to meet them in person (allowing him to circumvent Eve).

3 THE WAY OUT OF SHANNON IMPOSSIBILITY

Perhaps our earlier assumption that Eve is computationally unbounded was too generous. After all, in the real world, no one has unlimited time and resources. We would like to try and escape

\footnote{Since excluding some message \( m \) would give Eve some information (i.e., the message is not \( m \)) which she did not know a-priori.}

L2-2
from the problems posed by perfect secrecy requirements by taking advantage of Eve’s real world limitations. Before we can do this, we must first model Eve’s limitations in some fashion. We will start with the assumption that Eve can only execute algorithms on “large” problems that can be done in Probabilistic Polynomial Time (PPT), which will be defined below. First, we define the notion of a polynomial time algorithm.

**Definition 5 (poly-time (Polynomial Time) Algorithm)** If an algorithm $A$ gets an input of size $k$, it is considered polynomial time if it runs in $O(k^c)$ time where $c$ is a constant. We write $y = A(x)$ to denote the output of $A$ on input $x$.

With this definition in hand, we can proceed to define PPT.

**Definition 6 (PPT (Probabilistic Polynomial Time) Algorithm)** It is a polynomial time algorithm $A$ that is randomized. Namely, it is allowed to flip coins during its computation. We write $y = A(x; r)$ to denote the output of $A$ on input $x$, when $r$ were the internal coin tosses made by $A$. We write $y \leftarrow A(x)$ to denote the random variable $y$ which corresponds to the randomized output of $A$ on input $x$. This means that $r$ was chosen at random and $y = A(x; r)$ was computed.

It is not enough to simply state that Eve has limited computational resources, we must also allow for the fact that Eve might randomly happen to “guess” the solution to a problem. If we haven’t achieved perfect secrecy, it might be possible for Eve to simply break our system by luck (this comes partly from the Probabilistic aspect to PPT). We would like to somehow quantify Eve’s chances, and say that she has a very small, “negligible” chance of succeeding. We now define “negligible” in terms of the problem size $k$ (as in the above definitions).

**Definition 7 (Negligible in terms of $k$ (negl($k$)))** An arbitrary function $v(k)$ (possibly a type of probability function) is negl($k$) if:

$$(\forall c > 0) \ (\exists k') \ (\forall k \geq k') \ \left[ v(k) \leq \frac{1}{k^c} \right]$$

In other words, negl($k$) means something which shrinks faster than $1/\text{ANY polynomial in } k$ as $k$ grows. An example of a negligible function in $k$ would be $2^{-k}$, which shrinks exponentially fast. So in the following, we will usually omit the specific function $v(k)$, and just write negl($k$) in place of any such function. We will be more precise later in the course.

In general, we can now model everything in terms of some “security parameter” $k$ which represents the size of the problem that Eve is faced with if she wants to break our system. Typically, $k$ will grow as the size of the problem increases, so we can measure Eve’s chance of “breaking” the system as a function of $k$. The system is “secure”, if this chance — usually called the advantage of Eve — is negligible in the security parameter $k$.\(^2\) Finally, we will often say “adversary” instead of Eve.

We note briefly that polynomial time algorithms which invoke other polynomial time algorithms are still polynomial time (i.e. poly(poly($k$)) = poly($k$)), and any polynomial in $k$ times a a negligible function in $k$ is still negligible (i.e. poly($k$) · negl($k$) = negl($k$)). This explains why we

\(^2\)Of course, we also have to ensure that good parties, e.g. Alice and Bob, can do everything in time polynomial in $k$, since assuming otherwise would be unrealistic, as well as unfair to Eve.
choice such generic notions for our model. Intuitively, PPT should be synonymous to “feasible” or “efficient”, which “negligible” should be synonymous to “irrelevant”, “tiny”, “unimportant” or “not worth even bothering with”.

Before returning to a more complicated problem of encryption, we now discuss an example of a very important “primitive” we can use to build practically secure cryptographic systems. This primitive will be very naturally defined using the notions introduced above, and is the first cryptographic notion we formally define.

4 One Way Functions and permutations (OWFs and OWPs)

A function is One-Way if it is “easy to compute, but difficult to invert”. Not surprising, this notion is a very important building block for many things to come. More formally...

Definition 8 (OWF) A function \( f : \{0, 1\}^* \rightarrow \{0, 1\}^* \) (or, a function \( f \) that maps from one set of binary numbers to another, not necessarily the same) is a one-way function (OWF) if it satisfies two properties:

1. \( \exists \) poly-time algorithm which computes \( f(x) \) correctly \( \forall x \). (Thus, easy to compute.)

2. \( \forall \) PPT Algorithm \( A \), the advantage of \( A \), denoted by \( \text{Adv}(A) \), is negligible in the security parameter \( k \), where

\[
\text{Adv}(A) \overset{\text{def}}{=} \Pr(f(z) = y \mid x \leftarrow R \{0, 1\}^k, y = f(x); z \leftarrow R A(y, 1^k))
\]

where \( \leftarrow R \) means randomly chosen. (So \( x \) is randomly chosen from the set of \( k \)-bit numbers, and \( z \) is randomly output from algorithm \( A \) when it has \( y \) as input.) Thus, \( f \) is hard to invert. So, in polynomial time (in \( k \)) Eve has probability \( \text{negl}(k) \) or less of figuring out any preimage of \( f(x) \).

This essentially states that if Eve is given the output of \( f \) on some random input (which Eve is not told), she cannot find any input to \( f \) which produces that same output in polynomial time with better than some very small (negligible) probability. These OWFs will prove to be very useful in constructing practical cryptographic systems.

Keep in mind that no proof derived yet shows that these OWF’s exist (even if we were to assume \( P \neq NP \)). However, there’s good evidence OWF’s do exist. And in later lectures, candidate OWF’s will be shown. We next describe a minor variation on OWFs.

Definition 9 (OWP) A OWF which is also a permutation (i.e., every \( y \) has a unique preimage \( x \)) is called a one-way permutation (OWP).