These notes define the problem of secure communication, identify the relevant agents, and offer some assumptions about those agents. An initial effort at a solution involves the agents sharing a secret, and produces a private-key method called the One Time Pad algorithm, which seems a good candidate for secure communication. To check if this is true, the notion of perfect security is then defined more precisely and OTP is proved to be secure. Although OTP is secure, it has several undesirable features, such as unbounded key lengths. (We also prove that no secret key cryptosystem is secure in the Shannon sense unless the key length is unbounded. This is the Shannon Theorem.) We attempt to circumvent the key length problem by changing the assumptions about the agents. Namely, we no longer assume a computationally unbounded adversary. This leads to a new class of "public key" encryption methods where the agents do not share a secret. The requirements of public key encryption motivate the use of One Way Functions and trapdoor functions, whose properties are described.

1 DESCRIPTION OF PROBLEM

Assume there are three agents, Bob, Alice, and Eve. Bob wants to send Alice a private letter that only Alice can read. Eve is an adversary who may intercept the letter, but reading it should not enable her to reconstruct Bob's message to Alice. This is the essence of the problem of secure communication. A more formal definition of "secure" will be given later.

To meet these requirements, Bob must somehow alter his message to Alice so it cannot be understood by anyone else. This alteration is called encryption. If m is the message to be encrypted, (also known as the "plaintext" or the "cleartext") then c, the encrypted message, (also known as the "ciphertext") is produced by an encryption function $E(m, ???) \to c$, which Bob uses. Alice uses a symmetric decryption function $D(c, ???) \to m$.

The question marks in the function definitions stand for other pieces of information that may be provided to the encryption/decryption functions, depending on the specific method being used. Some possibilities are:

- S_b: A piece of secret information known to Bob but not to Eve.

- S_a: A piece of secret information known to Alice but not to Eve.

- PK: A piece of public information, available to everyone.

- R: A random factor, presumably drawn from some pseudo-random distribution. This variable is only applicable on Bob's end; Alice deterministically decrypts the ciphertext (since we want errorless communication).

Notice, Eve does not know S_a or S_b, even though S_a and S_b could have a common piece of *shared secret s*. This s is not known to Eve, but is known to both Alice and Bob.

2 INITIAL ASSUMPTIONS

To describe the nature of E and D more precisely, some assumptions about the agents must first be made.

- The encryption and decryption algorithms are public information known by all agents.

- Alice must *always* be able to recover the correct message.

- Eve is computationally strong, i.e. has an unbounded computational power. This assumption will be relaxed later to allow for more realistic modeling, but we assume it for now.

3 OBSERVATIONS

Some observations follow from these assumptions:

1. **Alice must have a secret from Eve.**

   Rationale: Alice must be able to decrypt the message from Bob. If Eve knows everything Alice knows, then Eve is functionally equivalent to Alice. Namely, if Alice can read the message then so can Eve. This contradicts the problem definition.

2. **Bob must have a secret from Eve. In fact, Alice and Bob must <u>share</u> a secret s not known to Eve.**

   Rationale: Let Bob's message be m, and let X be all the other information he uses to produce c (includes his randomness, possible secret key, etc.). Eve can go through all candidate values m' for m and X' for X, until $E(m', X') = c$. When this happens, Eve is sure that $m = m'$ unless Alice knows a part of X. Indeed, otherwise and if m is different from m', from Alice's point of view the message could be both m and m', which contradicts unique decodability. Thus, the only way Eve should still be unsure if $m = m'$ is if Alice knows a part of X, i.e. Alice and Bob share a non-empty secret s not known to Eve.

4 ONE TIME PAD

Now we have a pattern for encryption where Bob and Alice must have a shared secret s. We next describe a specific implementation of such an encryption system, called the *One Time Pad* (OTP). (Note that OTP is a specific scheme for achieving our goal but not the only such scheme possible.) In One Time Pad, we assume that the message m is somehow chosen from the domain $\mathcal{M} = \{0,1\}^k$, while a shared secret key s is chosen *at random* from the domain $\mathcal{S} = \{0,1\}^k$. Notice, in OTP Alice and Bob do not need any other secret information (beside the shared s which is secret from Eve).

DEFINITION 1   The One Time Pad (OTP) encryption function is easily described; simply take the exclusive OR of the message string and the key s. This produces the cipher c, which can be decrypted by XORing it with the key s:

$$
\begin{aligned}
E(m, s) &= m \oplus s \\
D(c, s) &= c \oplus s
\end{aligned}
$$

$$\diamondsuit$$

Intuitively, OTP is good because every message $m \in \mathcal{M}$ and ciphertext $c \in \mathcal{C}$ correspond to the unique key $s = m \oplus c$. So without the key, all messages are indistinguishable to Eve.

Remark 1 *Before the OTP, people had many other cryptosystem, including Ceasar cipher, shift cipher, substritution cipher, hill cipher, vigenere cipher, permutation cipher, and many others. Most of these ciphers are "insecure" (whatever this means for each of them). It is somewhat entertaining to study these ciphers and see why each of them is not secure. However, since this will not be useful for the remainder of the course, we will not do this here. Interested readers are referred to Chapter 1 of the Stinson's book.*

5   DEFINING PERFECT SECURITY

This seems like a good system, but we'd like stronger verification. What do we mean when we say that a system is secure? By "system" we don;t just mean the OTP system, but any encryption where Alice and Bob (necessarily) share some secret key. Since in our assumptions the adversary (Eve) can access the encrypted text, an intuitive definition of perfect security is this: A method is secure iff the "odds" of the adversary to figure out m are the same whether or not he has c, i.e. seeing c does not increase these "odds". Of course, the problem is in formalizing the above informal definition. To do this, assume a message M was chosen by Bob from some probability distribution over $\mathcal{M}$ and Bob publicly announced this distribution. As we will see, the the definition below makes sense for any message distribution M, but for simplicity — and because it will suffice for our purposes — the reader may assume that M is chosen by Bob uniformly at random from $\mathcal{M}$. At this stage, from Eve's point of view each possibility $M = m$ is equally likely. Now assume Bob computes the ciphertext $C = E_s(M)$ and gives the particular outcome (which happens to be some $c \in \mathcal{C}$) to Eve. We now estimate again the probability that $M = m$ *after Eve has learned that $C = c$.* If the system is ideally secure, this probability should not change (i.e. remain uniform over $\mathcal{M}$), irrespective of specific m and c. Notice, the above probability is also taken over the random choice of the shared key s. Formally:

DEFINITION 2   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** if from the perspective of the attacker, $Pr(M = m \mid C = c) = Pr(M = m)$. This means that Eve's probability of guessing M remains unchanged after seeing any particular outcome $C = c$. $\diamondsuit$

We can now formally show that OTP is secure:

Proof: Take any $m \in \mathcal{M}$ and $c \in \mathcal{C}$.

$$
\begin{aligned}
Pr(M = m \mid C = c) &= \frac{Pr(M = m \wedge C = c)}{Pr(C = c)} \\
&= \frac{Pr(M = m) \cdot Pr(C = c \mid M = m)}{Pr(C = c)} \\
&= \frac{Pr(M = m) \cdot Pr(C = c \mid M = m)}{\sum_{\tilde{m} \in M} \left(Pr(M = \tilde{m}) \cdot Pr(C = c \mid M = \tilde{m}) \right)}
\end{aligned}
$$

We obtain step 1 from Bayes's law. Step 2 comes from the definition of conditional probability. In step 3, we expand out $Pr(C = c)$, which is the sum of all the conditional probabilities that $Pr(C = c)$, weighted by the probability of the condition. Namely, we sum over all possible $\tilde{m}$'s.

We then note that $Pr(C = c \mid M = \tilde{m}) = Pr(s = c \oplus \tilde{m})$, as $\tilde{m} \oplus s = c$ iff message $\tilde{m}$ and unique key s produce ciphertext c. Since every $s \in \{0, 1\}^k$ is just as likely, the probability is uniform over the key space: $Pr(s = c \oplus \tilde{m}) = \frac{1}{2^k}$. Hence,

$$
\begin{aligned}
Pr(M = m \mid C = c) &= \frac{Pr(M = m) \cdot \frac{1}{2^k}}{\sum_{\tilde{m} \in M} \left(Pr(M = \tilde{m}) \cdot \frac{1}{2^k} \right)} \\
&= \frac{Pr(M = m)}{\sum_{\tilde{m} \in M} Pr(M = \tilde{m})} \\
&= \frac{Pr(M = m)}{1}
\end{aligned}
$$

Thus, $Pr(M = m \mid C = c) = Pr(M = m)$, so OTP's security has been proven. $\qquad\square$

Remark 2 *As we can see, the fact that M is uniformly random was not used anywhere in the proof. Thus, OTP is perfectly secure for* any *message distribution M.*

6 PROBLEMS WITH ONE TIME PAD

Even though the OTP method is secure, it has several undesirable features. First, each message must use a new key. Indeed, if Bob sends using the same key s two ciphertexts, $c_0 = m_0 \oplus s$ and $c_1 = m_1 \oplus s$, then we have $c_0 \oplus c_1 = (m_0 \oplus s) \oplus (m_1 \oplus s) = m_0 \oplus m_1$. Thus, Eve certainly learns at least $m_0 \oplus m_1$, which is a lot of information! In particular, if m_0 is later revealed to Eve (say, it is no longer important), Eve will learn m_1, which could still be important. Clearly, this should not happen in the the ideal system.

Thus, if OTP is to remain secure with multiple message, it seems like a new key has to be used per each message. But since the length of each key in OTP is equal to the length of the encrypted message, it means that the overall length of the shared secret must be equal to the total number of secretly communicated bits. This seems to be really prohibitive for long term communication. The immediate question that comes to mind is whether this negative phenomenon is a specific feature of the OTP (and maybe we could do better with a better scheme), or there is some deeper reason for this inefficiency that will prevail in *any* perfectly secure system. Unfortunately, the latter is the case as was shown by Shannon.

Theorem 1 (Shannon) *For* **any** *perfectly secure scheme where Alice and Bob share a key s from space $\mathcal{S}$ and can encrypt any message m from space $\mathcal{M}$, we must have $|\mathcal{S}| \geq |\mathcal{M}|$. Thus, OPT is optimal in this regard.*

Proof: Take any valid ciphertext c (which, say, could correspond to some message m_0 under the appropriate choice of the key). Let us count the number of messages m that could result from the decryption of c under some valid secret key s. Call this number A and let us estimate A in two different ways. On the one hand, each key s in $\mathcal{S}$ can correspond to at most one m, since Alice should decrypt c in at most one way for each choice of s (else unique decodability is gone).

Thus, $A \leq |\mathcal{S}|$. On the other hand, we claim that $A = |\mathcal{M}|$, i.e. *every $m \in \mathcal{M}$ can result in* producing c when encrypted by Bob. Indeed, if this was not the case for some m, then the a-priori $\Pr(M = m) > 0$, while if $C = c$ (which could happen with non-zero probability, say if M was equal to m_0 that could in turn result in $C = c$ by assumption), then $\Pr(M = m \mid C = c) = 0$, which would contradict the perfect security. Thus, $A = |\mathcal{M}| \leq |\mathcal{S}|$, completing the proof. $\qquad\square$

The above result shows some severe limitations of perfect security: the key must be as long as the message. Hence, the problem of *key exchange* (i.e., Alice and Bob agreeing in advance of the key) becomes a real issue. Clearly, we would like to find a better, more practical way to encrypt our data.

7   COMPUTATIONALLY BOUNDED ADVERSARIES

These previous negative results do not mean that encryption has no hope of succeeding. Indeed, the assumption that Eve has unbounded computing power is perhaps too strong. In reality, Eve does not, so we make the reasonable assumption on Eve's computing power. In turns out, the realistic and convenient way to model this is to say that Eve has only *probabilistic polynomial time* (PPT) computing power. To be fair, we shall restrict Alice and Bob to PPT as well.

Before defining this concept, let us by first defining a *Polynomial Time Algorithm*.

DEFINITION 3 [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. $\qquad\diamond$

Now, let us define *probabilistic polynomial time* (PPT).

DEFINITION 4 [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.[1] 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. $\qquad\diamond$

Finally, only bounding the running time of Eve will not be sufficient to get out of the Shannon's impossibility result. We will make it a bit more formal later, but, intuitively, this is because Eve can always try to guess the value which is hard to compute, and this way get a "negligible" chance to break the resulting cryptosystem, even though it would take Eve a very long time to break it with any "significant" probability. In other words, although the "advatange" of Eve over an ideal system might be non-zero, it will be so small (as long as Eve is PPT), that for all effective purposes it can simply be ignored.

This leads to the following definition of a *negligible function*.

DEFINITION 5 [Negligible Function (negl(k))] A function $v(k)$ is called negligible, and denoted negl(k), if:

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

$\diamond$

---
[1] Notice, the length of r must necessarily be polynomial in k.

In other words, we will use the notation $\mathsf{negl}(k)$ to say that some function of interest $v(k)$ (typically, the probability of Eve "breaking the system on inputs of size k") is less than the inverse of any polynomial for sufficiently large k.

The reason for this choice is because negligible probability of success stays negligible after even a polynomially many attempts to break the system: notationally, $\mathsf{poly}(k) \cdot \mathsf{negl}(k) = \mathsf{negl}(k)$.

WHAT IS k? In algorithms, k usually denotes the "size of the problem", which is natural for the problem at hand. E.g., for sorting it is the size of the array to be sorted, for graph algortihms it is the number of vertices or edges of the graph, and so on. In cryptography, most tasks usually have many participants who take different inputs. For example, in our scenario we already had Alice, Bob, and Eve, who all had different inputs. Despite that, for most cryptographic problems it is also natural to define "the size of the problem", which in cryptography is usually called the *security parameter*. After the meaning of this parameter k is specified, technically, every value of k defines a different and "larger" scenario. For example, for the one-time pad cryptosystem we chose k to denote the message size. Then, different k's correspond to different and "larger" cryptosystems encryptinglonger and longer messages.

More generally, in more complex situations, once natural k is chosen as a security parameter, we insist that all the honest parties (like Alice and Bob) run in some *fixed* (probabilistic) polynomial time in k (e.g., for the OTP case Alice and Bob where linear in k). On the other hand, Eve can run in PPT in k, and should have at most negligible probability of breaking the system, as a function of k. This is why k is called the "Security parameter", since by increasing k we get higher and higher security (although also slightly degraded efficiency for "honest" users). Thus, there is a tradeoff in determining the "correct" value of k: we do not want to make it low, so that the system is secure, but also do not want to make it unnecessarily large, so that we do not waste the resources of honest parties if the system is already "secure" for a smaller value of k.

So how should once choose k in practice? Well, after analyzing a given system, we hope that we can upper bound the advantage ε of Eve as a function of Eve's running time t (say, 2^{60}) and the security parameter k. In a good cryptosystem, this function is negligible in k (e.g., perhaps decaying like 2^{-k}). Thus, for a relatively small k, Eve's advantage ε will be smaller than some "acceptable" and "unnoticeable" threshold (say, 2^{-60}). And this is precisely the value k one chooses as a security parameter in practice. If later this k is too small, one increases k to make the system more secure (at the expense of having Alice and Bob run a bit slower). Ideally, though, one leaves enough "slack" in the initial choice k, so that reasonable increases in computer power still leave the system secure for decades to come!

In any event, at this stage this is a bit abstract, and we will come back to this point later, but we summarize our discussion as follows:

- One usually does not build a single cryptosystem, but parametrizes one's cryptosystem by a conveniently chosen security parameter k. Then "efficient" means "polynomial in k", and "negligible" means "negligible function of k".

- Intuitively, concrete k is later chosen in practice such that no attacker running in time "significantly less" than 2^k has advantage significantly greater than 2^{-k}. In a good system, this k will still be small enough, so Alice and Bob, who run in time poynomial in k (say, $O(k^2)$), still run pretty fast.

Remark 3 *Finally, we mention a trivial notional convention which we will repeatedly use without further explanation from now on. As we said, all participants are allowed to run in time polynomial in k. But how do they "know" k? Technically, we will have to explicitly supply k as part of the input. However, instead of giving them k in binary, which is only $\log k$ bits long, we will give k in unary, which is indeed k bits long. This is done to ensure that all our algortihms — which are polynomial in their* input length *— are allowed to run in time polynomial in the security parameter k. Thus, providing k in unary is a convenient way to ensure that polynomial in k time is allowed. Notationanly, k in unary is denoted 1^k. Thus, when an algortihm takes this "mysterious" 1^k, this is simply to explicitly tell it that the security parameter is equal to k.*

8   FINDING A BETTER SYSTEM

In light of these new relaxed assumptions on Eve, we must re-examine the observations we made earlier.

1. **Alice has a secret from Eve.**

   Clearly this observation must still hold, for the same reason given initially. Indeed, since Alice is now PPT, Eve still has enough power to decrypt the message if the assumption was false.

2. Bob has a secret from Eve. In fact, Alice and Bob must <u>share</u> a secret not known to Eve.

   This observation now does not seem to be necessarily true: if Eve is not strong, then she can no longer resort to brute force methods to reverse-engineer Bob's encrypted message. Indeed, we will give strong evidence that this observation is false.

For now, though, we define two important cryptographic models, depending on whether or not we assume the (no longer required) Observation 2 above. If we do, we come back to the same problem of secure communication with the shared key. This encryption setting is called *Symmetric-Key (or Private-Key)* Encryption (SKE). Since OTP is still a valid solution, but the proof of Shannon's theorem no longer holds when Eve is computationally bounded, the main goal of SKE is to **achieve "secure communication" with the secret key** *much shorter* **than the length of the messages exchanged**. We will come back to this challenging problem, but for now we discuss the new model, where we no longer assume that Alice and Bob share the secret (i.e., Observation 2 is false).

9   NOT SHARING A SECRET (PUBLIC-KEY ENCRYPTION)

In fact, we will assume that *Eve knows everything that Bob does.* From Observation 1, Alice should still have a secret S_a (which we now denote by SK, to stand for the "secret key") from Eve (and, hence, from Bob as well). We also allow for some public information PK (stands for "public key") available to all the parties. Notice, since Eve knows what Bob does, Eve, and in fact *anyone else*, can encrypt messages for Alice. However, we want *only Alice to be able to understand them*. This encryption setting is called *Public-Key (or Asymmetric-Key)* Encryption (PKE). Since the PKE model was impossible when Eve was unbounded, the main goal of PKE is **to achieve it at all** (only later concentrating on the secondary efficiency issues).

We now briefly compare the the SKE and the PKE. In SKE, at least an inefficient solution (OTP) clearly exist, but the problem is that of secure key-exchange. In PKE, the solution could not exist in the unbounded setting. However, if we are to find a solution, it will not have the key exchange problem. Indeed, since *anyone* can encrypt messages for Alice using her known public key PK, Alice simply needs to publish her PK once, and does not need to meet each possible recipient. On the other hand, the communication is *asymmetric*: in SKE both Alice and Bob could send the messages to each other using the shared key, while in PKE all the messages are intended for Alice, i.e. a new PKE structure is need for Bob to get messages. Finally, we will later see that in practice, the solutions to SKE seem to be much more efficient than those for PKE. Thus, PKE adds convenience at the cost of efficiency. More on this later in the course.

Since it seemed more interesting to do something that was not possible before at all, rather than break the "Shannon barrier" for something which was principally possible, historically people attacked the PKE first (even though we will later see that SKE is an *easier* problem in some sense). We will follow the same historic approach for now.

10 TOWARDS PKE… TO ONE-WAY FUNCTIONS

We start by making some specific assumptions to simplify the problem as much as we can. We start by assuming that the encryption algorithm E is not probabilistic. As we'll see soon, this assumption is *wrong*, but it would be a good simplification for now. Also, since the public key PK is fixed once originally chosen, we we define a function $f(m) = E_{PK}(m)$. The question is what properties should such a function f satisfy to yield a "reasonably good" PKE? Going though our desiderata, we claim that f should be:

1. **Invertible:** It must be possible for Alice to decrypt encrypted messages.

2. **Efficient to compute:** It must be reasonable for people to encrypt messages for Alice.

3. **Difficult to invert:** Eve should not be able to compute m from the "encryption" $f(m)$.

4. **Easy to invert given some auxiliary information:** Alice should restore m using SK.

It turns out that properties 2 and 3 (easy to compute but difficult to invert) capture the main difficulty in designing such an f. Therefore, functions satisfying just properties 2 and 3 alone received some special treatment. Such functions are called *one-way functions* (OWF's). Adding property 1 on top will yield the notion of *one-way permutations* (OWP's), while finally throwing property 4 gives *trapdoor permutations* (TDP's).

For syntactic convenience, in the definitions below the input x will replace the "message" m and the trapdoor T — the "secret key" SK. Also, $\mathsf{poly}(k)$ denote some polynomial function in k, whose particular form is unimportant (as long as it exists).

DEFINITION 6 A function $f(x)$ is a *One-Way Function* iff there exists a polynomial time algorithm to compute $f(x)$ and for any PPT algorithm A (this is the adversary's algorithm) trying to "invert" f, we have

$$P(f(z) = f(x) \mid x \in \{0,1\}^k, y \leftarrow f(x), z \leftarrow A(y, 1^k)) = \mathsf{negl}(k)$$

Namely, any PPT A succeeds with only a negligible probability in in finding *any* valid preimage z of $y = f(x)$. In case when f is also one-to-one, f is called a *One-Time Permutation*. $\diamond$

We remark that in the definition of the OWF we had to require that A succeeds if $f(x) = f(z)$, rather than the "natural" $x = z$. This is because we did not require f to be a permutation yet. Indeed, if we changed $f(x) = f(z)$ to $x = z$, the trivial function $f(x) \equiv 0$ will trivially satisfy the definition (why??). Of course, this problem does not arise for OWP's, since there the condition $f(x) = f(z)$ is clearly equivalent to $x = z$ (i.e. x is only preimage of $y = f(x)$).

DEFINITION 7 A *Trapdoor Permutation* f is a OWP with the extra property that for every n, there exists a string T (the "trapdoor") of polynomial length, $|T| \leq \mathrm{poly}(k)$, and a polynomial-time "inversion" algorithm I such that $I(f(x), T) = x$, for any $x \in \{0, 1\}^k$. $\diamond$

Notice, that since inverting f is hard without anything and easy with the trapdoor, it means that it *must* be hard to compute the trapdoor (i.e., find the "secret key"). Notice, the existence of the trapdoor seems like the essence for constructing PKE: everyone has access to the encrypting function, but the only person possessing the "auxiliary information" has the ability to invert $E_{PK}(x)$ and thus decrypt encrypted messages.

Next lecture we will see the specific number-theoretic examples of the OWF's, OWP's and TDP's, as well as try to see if they indeed suffice for solving our problem of secure communication.

Remark 4 *Mathematically speaking,* OWF*s,* OWP*s and* TDP*s should also include a* PPT *key generation algortihm* Gen*, which, on input* 1^k*, outputs the public key* PK *desribing* f*, and, in case of* TDP*s, also the corresponding trapdoor information* T*. This way we know how to practically sample a corresponding hard-to-invert function. Also, although we will assume for now that the message space of* f *is* $\{0, 1\}^k$*, in general the domain of* f*, denoted* $\mathcal{M}_k$*, could also be generated as part of the key generated algortihm* Gen*, as long as one can efficiently sample a random element* x *from* $\mathcal{M}_k$*. We will come back to this point later.*