CMSC456 Notes

Hersh Kumar

Contents

1 Symmetric Key Encryption (Ciphers) 2
- 1.1 Historical Ciphers . . . 3
- 1.2 Frequency Analysis . . . 3
- 1.3 Vigenere Cipher . . . 4
- 1.4 Formal Definition of Symmetric Key Schemes . . . 4
- 1.5 One-Time Pad (Vernam's Cipher) . . . 6
- 1.6 The Computational Approach . . . 8
- 1.7 Definition of a Private Key Encryption Scheme . . . 9
- 1.8 Pseudorandom Number Generators . . . 10
- 1.9 CPA Security . . . 12
- 1.10 Block Ciphers/Pseudorandom Permutations . . . 13
- 1.11 Block Ciphers . . . 13

Cryptography is centered around providing information security. We are mainly concerned with protecting information, which is a very broad set of contexts and settings. In this class, we only cover the method of protecting information on the internet, such as the basics of the TLS protocol.

We have a set of different concerns. Firstly, we have data privacy, we don't want our private data being leaked (such as credit card information or passwords). We also can be worried about data integriy and authenticity, where data being transferred can be modified. These are the two types of issues that we are worried about.

For data privacy, if Bob is receiving a message to Alice, we want to be assured that an eavesdropper cannot see the contents of the message. As for data integrity and authenticity, we want to make sure that the message really originates from Alice, and that the message has not been modified in transit. Typically, when we interact online, we care about both of these issues at the same time.

We will explore how to define security, and define what it means for a scheme to be secure. Once we have a formal definition, we can explore how to prove security, based on information theory proofs and reduction via computational assumptions. We will also learn about tools for building secure schemes, such as tools for practical symmetric key constructions. We will also see lots of constructions of cryptographic schemes, such as symmetric key encryption, MAC, hash functions, key exchange, etc.

In the first half of the course, we will be looking at the private key setting, dealing with privacy using symmetric key encryption, and then dealing with integrity and authenticity with message authentication codes. In the second half, we will be working with the public key setting, with public key encryption and digital signatures, with an interlude midway dealing with number theory basics.

1 Symmetric Key Encryption (Ciphers)

Kerchoffs' Principle, stated in the 1800's, states that "the cipher method must not be required to be secret, and it must be able to fall into the hands of the enemy without inconvience".

In modern terms, we assume that the adversary knows the algorithm by which we encrypt and decrypt the message. This motivates the notion of a key, which is kept secret from the adversary. The encryption algorithm takes both the message and a key, and similarly the decryption algorithm takes in both the ciphertext and key. If someone doesn't know the key, they should be unable to decrypt the ciphertext. This is known as symmetric or private key encryption, where both the sender and receiver have the same key, which is private to the sender and receiver.

There are advantages of an open cryptographic design, and in fact all modern cryptographic systems utilize Kerchoffs' principle.

- Open systems are more suitable for large-scale use, all pairs of communicating parties can use the same scheme, just with a different key.
- Published designs undergo public scrutiny and are therefore likely to be stronger.
- Public design enables the establishment of standards

NIST for example, is running a Post-Quantum Cryptography competition to find a scheme to define a new standard.

Ciphers have been used for millennia, so we shall look at some examples of historical ciphers.

1.1 Historical Ciphers

We can look at the **Atbash** cipher, which originated around 600 B.C. This cipher works by mapping every character to its corresponding character at the end of the alphabet, so the letter A maps to Z, B maps to Y, etc. This cipher actually does not have a key, the encryption and decryption methods do not require a secret key. Thus the key space, the space of keys is empty, and thus has size 0. This is completely insecure, if we assume Kerchoffs' principle, where the adversary would instantly be able to encrypt and decrypt at will, since there is no key.

The shift or **Caesar** cipher takes the message and shifts each character forwards in the alphabet by the key value, using modulo 26. This key can be any integer between 0 and 25, and thus the key space size is 26 (25 if you remove the trivial nonencrypted case). This cipher is more secure than Atbash, but in modern times, we can just try all possible shifts, there are only 26 of them, so even if the key is unknown, we can just try every single one until we find something that is readable. This method is known as a brute force search.

Thus in order for us to have security from brute force search methods, we would need a key space that is *sufficiently* large, in this case a key space size of 26 is trivially not feasible.

The **Scytale** cipher works by taking a staff of some certain diameter, and wrapping paper around it, and writing the message lengthwise. The ciphertext is given by unraveling the paper. To decrypt, we wrap the ciphertext around a different staff, of the same diameter, and read lengthwise. In this case, the secret key is the diameter of the staff, and thus the key space size is essentially infinite, or at the very least, very large. To break this, all we need to know is how many letters fit on one loop, and thus we can just try all possible choices of taking the nth letter. Even though the key space is theoretically infinite, it is still not that difficult to break. This is because instead of doing a brute force search, the ciphertext itself leaks information about the diameter of the staff, because it must be an integer between 1 and the length of the ciphertext.

Monoalphabetic substitution works by mapping each character in the alphabet to a ciphertext letter, which can be chosen arbitrarily. Thus the key is the one-to-one mapping from plaintext letters to ciphertext letters. The key space size is 26!. This is quite large, and thus brute force search is intractable, but there is a better way to break this cipher. The better way of breaking this, is frequency analysis, where we assume the language or characteristics of the plaintext, and we look at the frequency of letters in that language, and do the same for the ciphertext, and then match frequencies.

1.2 Frequency Analysis

Let us attempt to use frequency analysis to break the shift cipher. To do this, we match the frequencies of English text to the frequencies of letters in the ciphertext, and attempt to obtain the shift. This works better than for the monoalphabetic substitution cipher, since if we get 1 letter correct, we have everything correct. This is a bit overkill, because we can do an easy brute force search over the 26 possible shifts.

There is an improved attack, where we can associate letters of the English alphabet with numbers, and then let p_i denote the probability of the i-th letter in English text. We can then use the frequencies of letters in English, and compute that

$$\sum_{i=0}^{25} p_i^2 \approx 0.065$$

We can then look at the ciphertext, and let q_i denote the probability that a letter in the ciphertext will be the i-th letter (number of occurences over total length). We can then compute I_j, defined as:

$$I_j = \sum_{i=0}^{2} 5o_i \cdot q_{i+j}$$

We can compute this for all values of j, 0 through 25, and then find the one that matches 0.065 the closest. This will indicate the shift value, or the key.

1.3 Vigenere Cipher

The Vigenere cipher is poly-alphabetic, each plaintext character maps to different ciphertext characters at different points. The Vigenere cipher applies multiple shift ciphers in sequence.

To break this, we can apply the same method we just used, based on the periodicity of the key. Once we know the length of the key, we can just apply the frequency analysis method to every letter spaced out based on the length of the key. To find the length of the key, we can try different values of the key length τ, and consider the stream of every τth letter. We can compute the sum of squares of the frequencies in this stream, and check whether the value is approximately 0.065, as before. The value of τ that is closest to 0.065 will be the likely length of the key.

We have shown that a secure encryption scheme must have a key space that cannot be search exhaustively in a reasonable amount of time. This also shows that designing secure ciphers is difficult, all historical ciphers can be completely broken.

Now let us address the question of what it means for a scheme to be secure. In the symmetric key setting, we have a key generation function, Gen, and we have functions that encrypt and decrypt using a key and a message, Enc and Dec. One naive idea is that if the scheme protects the secret key, we are secure. However, there are trivial schemes that do not use the secret key, and thus the plaintext is obvious even with no knowledge of the secret key. Another idea is that if the attacker cannot find the plaintext it is secure, but if the attacker can get a sizable portion of the message, we would not consider that secure. Thus we generalize this to the statement that "an encryption scheme is secure if no adversary learns meaningful information about the plaintext after seeing the ciphertext."

How do we formalize the idea of "learning meaningful information"?

1.4 Formal Definition of Symmetric Key Schemes

Let us formally define an encryption scheme. An encryption scheme is define by 3 algorithms, Gen, Enc, and Dec. We also specify the message space, $\mathbf{M}$, with $|\boldsymbol{M}| > 1$. The key generation algorithm Gen is probabilistic, and takes no input. It outputs a key k according to some distribution, drawn from the key space $\boldsymbol{K}$, the set of all possible keys.

The encryption algorithm Enc takes an input key $k \in \boldsymbol{K}$, and a message $m \in \boldsymbol{M}$. This algorithm may be probabilistic, and outputs a ciphertext $c \in \boldsymbol{C}$, where $\boldsymbol{C}$ is the space of all ciphertexts.

The decryption algorithm Dec takes in k and an encrypted message c, and outputs the original message, m.

We can discuss the distribution over the spaces. For example, for $k \in \boldsymbol{K}$, we can define some probability that the key output by Gen will be k, $\Pr[K = k]$.

Likewise, we can define a distribution over $\boldsymbol{M}$, where for some message $m \in \boldsymbol{M}$, we have some $\Pr[M = m]$. This models a priori knowledge about the message.

The distributions over $\boldsymbol{K}$ and $\boldsymbol{M}$ are independent. From these two, we can determine the distribution over $\boldsymbol{C}$:

$$C \leftarrow \mathsf{Enc}_K(M)$$

We can now define perfect secrecy, which was first defined by Claude Shannon.

Definition 1.1. *An encryption scheme (Gen, Enc, Dec) over a message space $\boldsymbol{M}$ is **perfectly secret** if for every probability distribution over $\boldsymbol{M}$, every message $m \in \boldsymbol{M}$, and every ciphertext $c \in \boldsymbol{C}$ for which $Pr[C = c] > 0$:*

$$Pr[M = m \,|\, C = c] = Pr[M = m]$$

Essentially, we are stating that in every case, if the adversary is given the ciphertext, the probability of them obtaining the message is no better than if they had no information. The right side of the expression represents the a priori knowledge of the adversary. The left side is the a posteriori knowledge of the adversary, the knowledge after seeing the ciphertext. Perfect secrecy states that these are exactly the same. Note that this holds for all message distributions, whether or not the message is English or gibberish, perfect secrecy implies that the adversary gets no information. Another way of phrasing this is that a scheme is perfectly secret if, after seeing the ciphertext, the adversary has not gained any more knowledge than they already knew.

Lemma 1.1. *An encryption scheme (Gen, Enc, Dec) over a message space $\boldsymbol{M}$ is perfectly secret if and only if for every probability distribution over $\boldsymbol{M}$, every message $m \in \boldsymbol{M}$, and every ciphertext $c \in \boldsymbol{C}$:*

$$Pr[C = c | M = m] = Pr[C = c]$$

Proof. This is an iff statement, and thus we have to prove this from both sides. Let us begin by proving that perfect secrecy implies the conclusion of the lemma.

To prove this, we must recall Baye's theorem:

$$\Pr[A|B] = \frac{\Pr[B|A]\ \Pr[A]}{\Pr[B]}$$

We can apply this to the definition of a perfectly secret scheme. Suppose we have a perfectly secret scheme, with some arbitrary distribution over $\boldsymbol{M}$, with some $m \in \boldsymbol{M}$ and $c \in \boldsymbol{C}$. We can then write out the definition of perfect secrecy, and apply Baye's Theorem:

$$\begin{aligned} \Pr[M = m \,|\, C = c] &= \Pr[M = m] \\ \frac{\Pr[C = c | M = m]\ \Pr[M = m]}{\Pr[C = c]} &= \Pr[M = m] \\ \Pr[C = c | M = m] &= \Pr[C = c] \end{aligned}$$

□

Let us now prove that if a scheme satisfies the conclusion of the lemma, the scheme is perfectly secret.

Assume we have some scheme, with some arbitrary distribution over $\boldsymbol{M}$, with some $m \in \boldsymbol{M}$ and $c \in \boldsymbol{C}$, and we have that the conclusion of the lemma is true:

$$\begin{aligned} \Pr[C = c | M = m] &= \Pr[C = c] \\ \frac{\Pr[M = m | C = c]\Pr[C = c]}{\Pr[M = m]} &= \Pr[C = c] \\ \Pr[M = m | C = c] &= \Pr[M = m] \end{aligned}$$

□

This lemma is stating that the ciphertext contains no information about the message from the perspective of the adversary.

We also have another lemma that is useful:

Lemma 1.2. *An encryption scheme over a message space $\boldsymbol{M}$ is perfectly secret if and only if for every probability distribution over $\boldsymbol{M}$, every $m_0, m_1 \in \boldsymbol{M}$, and every ciphertext $c \in \boldsymbol{C}$:*

$$Pr[C = c | M = m_0] = Pr[C = c | M = m_1]$$

We will use this lemma to prove that the One-Time Pad is perfectly secret.

1.5 One-Time Pad (Vernam's Cipher)

In 1917, Vernam patented a cipher now called the one-time pad, that obtains perfect secrecy. There was no proof of the secrecy at the time, until 25 years later, when Shannon introduced the idea of perfect secrecy.

Let us do an example. Suppose our message space is messages of length 3 bits, $\{0,1\}^3$, for example, $m = 011$. The Gen function will output a random key:

$$k \in \{0,1\}^3$$

The encryption algorithm generates the ciphertext by computing a bitwise `xor` of the key and message:

$$c = k \oplus m$$

And the decryption algorithm reverses the operation:

$$m = k \oplus c$$

Let us now prove that it is perfectly secure.

Theorem 1.3. *The one-time pad encryption scheme is perfectly secure.*

Proof. We will use the previous lemma for this proof. For all distributions over $\boldsymbol{M}$, and for any pair of messages $m_0, m_1 \in \boldsymbol{M}$, and for all ciphertexts $c \in \boldsymbol{C}$:

$$\Pr[C = c | M = m_0] = \Pr[C = c | M = m_1]$$

We will now fix an arbitrary distribution over $\boldsymbol{M}$, as well as an arbitrary pair of messages. We can then write out the probabilities in the lemma, starting from the left side of the equality:

$$\begin{aligned}\Pr[C = c | M = m] &= \Pr[K \oplus M = c | M = m_0] \\ &= \Pr[K \oplus m_0 = c] \\ &= \Pr[K = c \oplus m_0]\end{aligned}$$

We have no information on c and m_0, but we do know that the key is chosen fully probabilistically. This means that this probability is just the reciprocal of the size of the space:

$$\Pr[C = c | M = m_0] = \Pr[K = c \oplus m_0] = \frac{1}{|\boldsymbol{K}|}$$

Since this holds for any message, the same relationship is true for m_1:

$$\Pr[C = c | M = m_1] = \frac{1}{|\boldsymbol{K}|}$$

Since these two are both equal to $\frac{1}{|\boldsymbol{K}|}$, the two statements must be equal:

$$\Pr[C = c | M = m_0] = \Pr[C = c | M = m_1]$$

Which by the lemma makes the one-time pad perfectly secure. □

While one-time pad is perfectly secure, there are some drawbacks. The first is that the key length is the same as the message length. This means that for every bit communicated over a public channel, a bit must be shared over a private channel. This is actually an inherent problem in perfectly secret encryption schemes. Another major drawback is that the key can only be used once (hence the name). This turns out to also be an inherent problem.

Theorem 1.4. *Let (*Gen*,* Enc*,* Dec*) be a perfectly secret encryption scheme over a message space* $\boldsymbol{M}$*, and let* $\boldsymbol{K}$ *be the key space as determined by* Gen*. Then,* $|\boldsymbol{K}| \geq |\boldsymbol{M}|$*.*

Proof. To prove this, we will attempt to prove the contrapositive, if we assume that the size of the key space is smaller than the size of the message space, then we cannot be perfectly secret. To do this, we need to prove that there exists a distribution over $\boldsymbol{M}$ and there exists m, c such that $\Pr[M = m | C = c] \neq \Pr[M = m]$.

Consider the uniform distribution over $\boldsymbol{M}$. Let use choose an arbitrary c such that $\Pr[C = c] > 0$. Let us now define a set $\boldsymbol{M}(c)$, which is a set of messages that can be reached from c. This is determined by taking every possible key in $\boldsymbol{K}$, and using Dec:

$$\boldsymbol{M}(c) = \{m \mid \exists k \in \boldsymbol{K} \text{ For which } m = \mathsf{Dec}_k(c)\}$$

Let us now attempt to find the size of the set $\boldsymbol{M}(c)$. The maximum size that this set could have is the size of the set of all the keys, $|\boldsymbol{M}(c)| \leq |\boldsymbol{K}|$. Now we use the assumption we made, that $|\boldsymbol{K}| < |\boldsymbol{M}|$. Thus we have that

$$|\boldsymbol{M}(c)| \leq |\boldsymbol{K}| < |\boldsymbol{M}|$$

Let m^* be the elements in $\boldsymbol{M}$ and not in $\boldsymbol{M}(c)$, which must exist due to the above inequality. We can then compute the probabilities:

$$\Pr[M = m^*] = \frac{1}{|\boldsymbol{M}|}$$
$$\Pr[M = m^* | C = c] = 0$$

Where we leverage the fact that the message distribution is uniform, and the message m^* cannot be obtained by starting with ciphertext c. Since these two are not equal to each other, we have completed the proof. □

The idea of this proof is that we are considering brute-force searching, and we have shown that we need a key space at least as large as the message space in order to be perfectly secret.

Theorem 1.5 (Shannon's Theorem)**.** *Let (*Gen*,* Enc*,* Dec*) be an encryption scheme with message space* $\boldsymbol{M}$*, for which* $|\boldsymbol{M}| = |\boldsymbol{K}| = |\boldsymbol{C}|$*. The scheme is perfectly secret if and only if:*

1. *Every key* $k \in \boldsymbol{K}$ *is chosen with equal probability* $\frac{1}{|\boldsymbol{K}|}$ *by algorithm* Gen*.*
2. *For every* $m \in \boldsymbol{M}$ *and every* $c \in \boldsymbol{C}$*, there exists a unique key* $k \in \boldsymbol{K}$ *such that* $\mathsf{Enc}_k(m)$ *outputs* c*.*

Note that this theorem only applies if the size of all 3 spaces is the same.

1.6 The Computational Approach

Our definition of perfect secrecy is a little too restrictive, there is not that much we can do if we require perfect secrecy. Instead, we make two main concessions:

1. Security is only guaranteed against efficient adversaries that run for some feasible amount of time (generally only consider polynomial-time algorithms).
2. Adversaries can potentially succeed with some very small probability.

These relaxations are made in order to obtain something that is more useful in practice. This approach was spearheaded by Diffie and Hellman, and went hand-in-hand with the development of computational complexity.

Usually when we consider the runtime of an algorithm, we consider the runtime with respect to the input size (Sorting algorithms are judged by how the runtime scales with the size of the input list). For cryptographic schemes, we actually divorce the length of the message and the runtime of the algorithm. This leads to the introduction of the security parameter, which parameterizes the scheme and all involved parties. When honest parties initialize a scheme, they choose some value n for the security parameter (a typical parameter is $n = 128$). We allow the adversary to run in polynomial-time with respect to the security parameter.

When we consider polynomial-time, we state that there is some polynomial p such that the adversary runs for time at most $p(n)$ when the security parameter is n. We also assume that the honest parties run in polynomial-time. The adversary may be much more powerful than the honest parties. The relative difficulty of the two is inherent in the fact that there is a single runtime complexity, such as the encryption and decryption taking place on the order of $\mathcal{O}(n^2)$, but this must be secure against all polynomial adversary runtimes, $\mathcal{O}(n^3)$, n^4, etc.

The second claim is that there is a negligible probability of success. We define a function f to be negligible if for every polynomial p and all sufficiently large values of n, it holds that $f(n) < \frac{1}{p(n)}$. For example, the function $\frac{1}{n^2}$ is non-negligible. A trick to check is to take the reciprocal and check whether it is polynomial or less than polynomial, it is non-negligible. The function $\frac{1}{2^n}$ is negligible, the reciprocal is super-polynomial.

What are the practical implications of computational security? For example if the key size is n, and any adversary running in time $2^{n/2}$ can break the scheme with probability $\frac{1}{2^{n/2}}$, then if $n = 128$, then the adversarial runtime is on the order of 2^{64}. Meanwhile, perhaps the encryption and decryption run in time n^2. In this case, the scheme runtime for $n = 128$ is on the order of $16,384$, which is far better than the 2^{64} of the adversary.

1.7 Definition of a Private Key Encryption Scheme

A private-key encryption scheme is a tuple of probabilistic polynomial-time algorithms $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ such that:

1. The key-generation algorithm takes as input security parameter 1^n and outputs a key k denoted $k \leftarrow \mathsf{Gen}(1^n)$. We assume that $|k| > n$.
2. The encryption algorithm Enc takes as input a key k and a message $m \in \{0,1\}^l$ and outputs a ciphertext c, denoted $c \leftarrow \mathsf{Enc}_k(m)$.
3. The decryption algorithm Dec takes as input a key k and ciphertext c and outputs a message m denoted $m := \mathsf{Dec}_k(c)$.

We have a gurantee of correctness, for every n, every key $k \leftarrow \mathsf{Gen}(1^n)$, and every $m \in \{0,1\}^l$, it holds that $\mathsf{Dec}_k(\mathsf{Enc}_k(m)) = m$.

Let us look at the first computational definition, which is indistinguishability in the presence of an eavesdropper.

Definition 1.2. *Consider a private-key encryption scheme $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$, any adversary A, and any value n for the security parameter. We have a random variable P, which is set by a game between the honest parties, the challenger, and the adversary. In this game, the adversary outputs two messages, m_0 and m_1, of the same length. The challenger then responds by choosing a key, via Gen, and then it will randomly select a bit $b \in \{0,1\}$, which indicates whether it will encrypt m_0 or m_1, and generate a ciphertext c. The adversary will attempt to determine which message was encrypt, by sending over a bit $b' \in \{0,1\}$. If the adversary guessed correctly, $b = b'$, then our random variable will be 1, and if the adversary guessed incorrectly, the random variable will be 0.*

A private key encryption scheme Π has indistinguishable encryptions in the presence of an eavesdropper if for all probabilistic polynomial-time adversaires A, there exists a negligible function negl

such that

$$Pr[P = 1] \leq \frac{1}{2} + negl(n)$$

How do we formalize the idea of the adversary obtaining meaningful amount of information? There are several different ways that were developed, such as semantic security, which has a really nasty definition, but luckily the idea of indistinguishability and semantic security are biconditional on each other, the two definitions are the same, so we can work with the game-based definition.

1.8 Pseudorandom Number Generators

A pseudorandom number generator (PRG) is a deterministic algorithm G which takes as input a short random seed s, and outputs a long string $G(s)$. The output is not random, it is pseudorandom, we "stretch" the randomness over the larger string. We will show that these generators will allow us to beat the Shannon bound, $|\mathcal{K}| > |\mathcal{M}|$. We want to have encryption schemes that have a key space smaller than the message space. These will allow us to create computationally secure encryption schemes, with $|\mathcal{K}| < |\mathcal{M}|$.

Let us define a game that will let us formally define a PRG. We have an adversary D (the distinguisher), who will be in one of two worlds. We have the ideal world, and we have the real world. In the real world, we run the PRG, which takes an input, the seed. The seed will be sampled as a truly random string of length n :

$$s \in \{0,1\}^n$$

We define the bitstring to be truly random if each bit is selected with a uniform distribution, like flipping a coin to decide 0 or 1.

We then pass the output of G to the distinguisher, D. The length of the output is defined to be some $l(n)$, which must be larger than n, $l(n) > n$. In the ideal world, we generate a truly random bitstring r of length $l(n)$, and pass this to the distinguisher as well. The goal of the distinguisher is to now attempt to distinguish the output of the PRG from the true random bitstring. We define a PRG as secure if any efficient (polynomial-time) D cannot distinguish between the two:

$$\left| \Pr[D(r) = 1] - \Pr[D(G(s)) = 1] \right| \leq \text{negl}$$

Let us now construct a computationally secure encryption scheme. The Enc algorithm will take in a message, and will run a PRG on the key generated by Gen, and treat the output as a pad:

$$c = m \oplus G(k)$$

To decrypt, we simply `xor` the ciphertext with the output of the PRG on the same key:

$$c \oplus G(k) = m$$

The difference between this and the one-time pad is that we are no longer constrained by the length of the key, since before we needed a pad that was as long or longer as the message, but now we can take keys that are shorter than the message, and use the PRG to generate a pad that is longer than the message.

Let us do some analysis of this.

Theorem 1.6. *If G is a PRG, then the construction above is a fixed-length private key encryption scheme that has indistinguishable encryptions in the presence of an eavesdropper.*

Proof. Generally in security proofs, we will do proofs by reduction, we prove the contrapositive. Let us attempt to determine what the P and Q are for this theorem. In this case, we assume that G is a PRG, and the implication is that the construction is secure. Thus the contrapositive is that if the construction is insecure, then G is not a PRG.

Let us first invert the definition of indistinguishability in the presence of an eavesdropper. Therefore, there exists a probabilistic polynomial-time adversary, and there exists a non-negligible function ρ, such that

$$\Pr[P = 1] \geq \frac{1}{2} + \rho(n)$$

We can now invert the PRG statement, there exists a probabilistic polynomial-time adversary, and there exists a non-negligible function ρ such that

$$|\Pr[D(r) = 1] - \Pr[D(G(s)) = 1]| \geq \rho(n)$$

We can think of this proof as building a distinguisher D out of an adversary A. We can think of the distinguisher as a box that takes in y, which is either r or $G(s)$, and it outputs either a 0 or 1. D must also invoke A, the adversary, which first spits out two messages, m_0 and m_1, then takes in a ciphertext c, and then outputs its guess, b'.

We have to specify how D generates the ciphertext c, known as the challenge ciphertext. We also have to specify what D does when it receives b' from A. When we do this, we have to then analyze the probabilities of the distinguished outputs.

We begin having D choose b at random. We then pass $c = m_b \oplus y$ to the adversary. Afterwards, the adversary will give us an output, b'. Essentially, if we are passed a truly random y, we are basically passing a one-time pad to A. If $b' = b$, we output 1, and if $b' \neq b$, D outputs 0.

Let us now analyze the probabilities:

$$\Pr[D(G(s)) = 1] = \Pr[P(n) = 1] \geq \frac{1}{2} + \rho(n)$$
$$\Pr[D(r) = 1] = \frac{1}{2}$$

The first probability is due to the fact that if we are passed $G(s)$, we are honestly encrypting either m_0 or m_1. Thus the probability that we get it correct is the same as the probability that the adversary gets it correct, we are just running the indistinguishability game. We then use our assumption, and we know that this is $\geq \frac{1}{2} + \rho(n)$. The second probability is $\frac{1}{2}$ because we are just computing a one time pad. We can now subtract the two of these and take the absolute value:

$$\left| \frac{1}{2} - \left(\frac{1}{2} + \rho(n) \right) \right| = \rho(n)$$

This is a contradiction for the PRG, since this must be non-negligible. □

We can use PRGs in stream ciphers. Stream ciphers send information bit by bit, each bit sent via a one time pad, where the pad is generated by a PRG. The sender and receiver both have a state bitstring, s, and they use a PRG G that outputs exactly 1 extra bit compared to the input. This extra bit is used to encrypt and decrypt the one-time pad. The PRG is always run on the state.

1.9 CPA Security

Consider a private-key encryption scheme $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$, any adversary A, and any value n for the security parameter. We can once again define a game, with the challenger running the key generation algorithm, to choose a key k. The adversary A gets oracle access to the encryption scheme, A can choose messages and encrypt them, and observe the ciphertext. The adversary can query the oracle as many time as it wants, for whatever messages. The challenger then generates a truly random bit b, and the adversary sends over two messages. The challenger then encrypts the corresponding message (based on b), and then the adversary guesses which message was encrypted. If the adversary guesses correctly, then the output of the game is 1, otherwise it is 0.

This leads to the definition of CPA security.

Definition 1.3. *A private key encryption scheme is CPA-secure if for all probabilistic polynomial-time adversaries, there exists a negligble function negl such that*

$$Pr[P(n) = 1] \leq \frac{1}{2} + negl(n)$$

Theorem 1.7. *A CPA secure encryption scheme cannot have deterministic algorithms for* Enc.

Proof. Assume that the Enc algorithm is deterministic. In the CPA game, the adversary can now simply query m_0, see its ciphertext, and then be able to instantly tell if the ciphertext that the challenger provides is m_0. Therefore the adversary can win the game with probability 1. □

To solve the issue that arises from have a probabilistic Enc function, instead of having a bijective mapping from $\mathcal{M}$ to $\mathcal{C}$, we instead make $|\mathcal{C}| \gg |\mathcal{M}|$, and then partition the space $\mathcal{C}$ into different subspaces that map to messages in $\mathcal{M}$. In order to maintain security, we need each subspace to be of size 2^n or higher, where n is the security parameter.

We can now look at pseudorandom functions, which are keyed functions $F : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^*$, is a two-input function, where the first input is called the key and denoted k. We will not see constructions of these functions until later in the course, but it turns out that if you have a PRG, you can always generate a PRF from it. These pseudorandom functions are stateless, we don't need to remember the output of the last call to the PRF, unlike what we saw with the PRG. We will often denote a PRF with the key already set as $F_k(\cdot)$.

If we compute $F_k(1)$, $F_k(2)$, and $F_k(3)$, the outputs of these 3 functions will be random and uncorrelated. The outputs will be deterministic, $F_k(1)$ will always be the same if the key is fixed.

To test a PRF, we do a game that is similar to the one for the PRG. We have two worlds, the real and ideal worlds. In the ideal world, we have a function f chosen at random from all functions from $\{0,1\}^n \to \{0,1\}^n$. In the real world, we have our PRF, $F_k(\cdot)$, where the key is chosen at random, $k \in \{0,1\}^n$. The distinguisher queries both worlds x times, and if $F_k(\cdot)$ is a PRF, then the efficient distinguisher will not be able to tell which world it is in.

$$|\Pr[A^f() = 1] - \Pr[A^{F_k}() = 1]| \leq \text{negligible}$$

Note that we are not passing implementations of the two functions to the adversary, we are giving it oracle access only. When we look at the ideal function, we can just generate one of the possible

functions by picking a random output string for every possible input string, this denotes one of the possible functions in our space, and we can compute the fact that there are 2^{n2^n} possible functions in our space (2^n inputs, and for each input there are 2^n possible outputs). Writing down a description of this function would take $n2^n$ bits, which is a massive number.

Now using a PRF, we can create a CPA-secure encryption scheme. To do this, we first call Gen, which picks a random $k \in \{0,1\}^n$, and a random string $r \in \{0,1\}^n$. We then pass that string to the PRF, which generates a pad, which we then use as the key of a one-time pad. We then pass the ciphertext, along with the random string to the receiver. For the decryption, we have r as well as c. It first runs the PRF on r, and then `xor` the result against the ciphertext, to return the message.

$$m \leftarrow c \oplus F_k(r)$$

Theorem 1.8. *If F is a PRF, then the construction above is a CPA-secure private key encryption scheme for messages of length n.*

Proof. We will approach this proof by proving the contrapositive. We will assume that there is an adversary A that can break the encryption scheme, and we will use A to build a distinguisher D that can defeat the PRF. □

1.10 Block Ciphers/Pseudorandom Permutations

Definition 1.4. *A Pseudorandom Permutation (PRP) is exactly the same as a PRF, except for every key k, F_k must be a permutation and it must be indistinguishable from a random permutation.*

Essentially, the function will have to be one-to-one and onto (a bijection). We once again have our security game, with the distinguisher trying to tell whether it is in the real world, which has a public keyed function F_k and its inverse, F_k^{-1}, (with some randomly chosen key k), and in the ideal world we have some f chosen at random from all permutations over $\{0,1\}^n$, as well as its inverse, f^{-1}. We allow the adversary to make queries to F_k, or make queries to F_k^{-1}.

1.11 Block Ciphers

We previously saw that we have CPA secure encodings via the use of a PRF, but we also noticed that the message length was restricted by the PRF. To get around this, we can split the message up into smaller chunks, and then chain the