CMPT 404 review notes

Jack ‘jryzkns‘ Zhou

1 Review on Probability

1.1 Distributions and Events

A **Sample Space** is defined as a finite set. For example we often will use $U = \{0,1\}^n$.
A **Probability Distribution** P over U is a function $P : U \to [0,1]$ such that $\sum_{x\in U} P(x) = 1$, or colloquially the entries in P sums to 1.
An **Event** is described as a subset of the sample space. For an event A, the probability of the event happening is defined as $\mathbb{P}[A] = \sum_{x\in A} P(x) \in [0,1]$. Trivially, $\mathbb{P}[U] = 1$. For events A_1 and A_2, the probability that either event happens is no greater than the sum of each event happening, that is: $\mathbb{P}[A_1 \cup A_2] \leq \mathbb{P}[A_1] + \mathbb{P}[A_2]$. We refer to this as the **Union Bound**.
Two events are considered to be **Independent** if the probability that they both occur is the same as the product of the two events happening separately. That is, $\mathbb{P}[A_1 \wedge A_2] = \mathbb{P}[A_1] \cdot \mathbb{P}[A_2]$.

1.1.1 Random Variables and Expectations

A **Random Variable (r.v.)** is defined as a function $X : U \to V$. A random variable X induces a distribution on V: $\mathbb{P}[X = v] := \mathbb{P}[X^{-1}(v)]$. One particular random variable is the uniform random variable **r** where we denote a random bitpattern. It is defined as $\forall a \in U\ \mathbb{P}[r = a] = |U|^{-1}$. The concept of independence for random variables also apply. For two random variables X and Y that take on values in V, X and Y are said to be independent randomv variables if $\forall a, b \in V\ \mathbb{P}[X = a \wedge Y = b] = \mathbb{P}[X = a] \cdot \mathbb{P}[Y = b]$.
A property of random variables that we can exploit is working with their **Expectations**, it can be interpreted as a "mean". The expectation of a random variable X over V is defined as $\mathbf{E}[X] = \sum_{a\in V} a * \mathbb{P}[X = a]$. It is important to note that the expectation is a linear operator.

1.1.2 Bernoulli Trials and Repeated Experiments

For a r.v. X that only has a binary outcome (with outcomes enumerated as 0 and 1; e.g. A coin toss), and $\mathbb{P}[X = 1] = p$, we denote such experiment as a Bernoulli trial. The sample space after carrying out n Bernoulli trials is $0, 1^n$. With this setting, we have some properties:

- Number of outcomes with k successes is $\binom{n}{k}$
- Probability of getting 1 in any trial is p, and getting 0 is $(1-p)$
- Probability of getting exactly k 1's out of n trials is $\binom{n}{k}p^k(1-p)^{n-k}$
- Probability of getting the first success on exactly the k^{th} trial is $p(1-p)^{k-1}$

Suppose we name a r.v. X such that it indicates which trial we obtain success on a Bernoulli trial, we can compute the expected amount of trials needed to obtain a successful outcome as $\mathbb{E}[X] = 1/p$. For example, with a fair coin we know that the probability of landing heads is 1/2. We expect 2 coinflips to grant us an outcome of heads.

1.2 Markov's Inequality

If a r.v. X is non-negative, then

$$\mathbb{P}[X \geq k] \leq \frac{\mathbb{E}[X]}{k}$$

A quick proof follows that the expectation of a r.v. X can be broken down in the following way: $\mathbb{E}[X] = \mathbb{E}[X < a] \cdot \overline{a} + \mathbb{E}[X \geq a] \cdot a$ where $\overline{a}, a$ encompasses all of X's domain. Then $\mathbb{E}[X] = 0 \cdot \overline{a} + \frac{\mathbb{E}[X]}{a} \cdot a$. Thus we can see the lower and upper bounds on $\mathbb{E}[X < a]$ and $\mathbb{E}[X \geq a]$.
A quick corollary on this inequality is called **Chebyshev's inequality**, where given the variance of the r.v., the following holds:

$$\mathbb{P}[|X - \mathbb{E}[X]| \geq a] \leq \frac{\mathbb{V}[X]}{a^2}$$

1.3 Exclusive OR (XOR)

An exclusive or operation (XOR) is a bitwise addition mod 2. It is denoted with the $\oplus$ symbol and it is central in Cryptography. We will discuss a specific property of XOR.
Suppose Y is r.v. over $0, 1^n$ and X is an independent uniform variable over $\{0,1\}^n$ as well. $Z := Y \oplus X$ is a uniform random variable on $\{0,1\}^n$ as well.

2 Historical Ciphers

In this section we introduce the notion of **Symmetric Encoding Schemes (SES)**, which constitutes of a pair

of algorithms (E, D) which is a pair of encryption and decrpytion algorithms.
The sender (uses the encryption algorithm) and receiver (uses the decrpytion algorithm) have an agreed upon key k, which is used in both the encryption and decryption (thus symmetric). The encrypted message is called a ciphertext and defined as $c := E(k, m)$. Decrypting the ciphertext is done using the same key: $m := D(k, c)$. A properly constructed SES has the property that one can always recover the plaintext given the correct key:

$$m = D(k, E(k, m))$$

There could be an eavesdropper that takes the ciphertext and performs an attack where the ciphertext is decrypted without knowledge of the key. We will discuss some historical ciphers and how they can be attacked using various methods.

2.1 Monoalphabetic Ciphers

This type of cipher replaces singular symbols in the original plaintext with another symbol. There are subtypes of this kind of cipher:

2.1.1 Shift Cipher

The most famous example of this kind of cipher is the Caesar Cipher, where each letter X in the ciphertext is mapped to $(X + 3) \mod 26$. More generally, we can change the shift to be any number in $[1, 25]$, and this shift amount can be considered to be a key.
To retrieve the key for this cipher, simply send the letter `a` which corresponds to 0, and the resulting ciphertext will reveal the key.
However, knowing that a ciphertext is encrypted in this cipher means the decrytion can be done in a brute force attack, as there are only 25 tries to make before decrypting the message.

2.1.2 Linear Cipher

This is an extension of a shift cipher. Each letter X in the plaintext is mapped to $(aX + b) \mod 26$, where a and b are now the keys to the cipher.
To retrieve the key for this cipher, send the message `ab` which corresponds to 0 and 1. The resulting ciphertext will give indication of how to solve for a and b in a linear system of equations.
To break this cipher, refer to how it is done with a general Monoalphabetic Cipher

2.1.3 Substitution Cipher

This is a more general case where each letter X in the plaintext is replaced with another arbitrary chosen letter, the mapping is recorded in a substitution table and it is considered a key.
To retrieve the key for this cipher, send the alphabet `abcdefghijklmnopqrstuvwxyz` and the subtitution table will be retrieved.
To break this cipher, refer to how it is done with a general Monoalphabetic Cipher.

2.2 Cracking Monoalphabetic Ciphers

If we know the language that the plaintext is in, we can infer lots of information about the plaintext from only looking at the ciphertext. In a sense, the linguistic structure of the plaintext is preserved in the ciphertext.
For the rest of the discussion, we will consider that the plaintext is always english.

2.2.1 Exploiting Linguistic Structure

Some rules in English that helps with breaking monoalphabetic ciphers are:

1. Single letter words are almost always `A` or `I`
2. Whenever there is an apostrophe, the following is generally followed by `S`, `T`, `D`, `M`, `LL`, or `RE`
3. Double letters are often `LL`, `EE`, `SS`, `OO`, `TT`

Usually, after a few replacement in the ciphertext, we start to see patterns that help us find the plaintext easier as part of words would be retrieved by then.

2.2.2 Frequency Analysis

Because substitution preserves the relationship between letters, letter frequencies in a language is not going to change relative the letter frequencies in the ciphertext. Therefore, the most frequently occurring letters in the ciphertext usually corresponds with the most frequently occurring letters in english. For more information, visit `https://en.wikipedia.org/wiki/Frequency_analysis`

2.3 Vigener Cipher

A cipher where the key is a n-digit string, and the string. The key is duplicated and concatenationed until it is matched in length with the plaintext. Then, the extended key-mask Y and the plaintext X is encrypted as $c = (Y + X) \mod 26$.
To recover the key from a Vigener cipher, simply send an arbitrarily long string of `a`'s (i.e. `aaaaaa...`). The resulting ciphertext will be the key-mask generated for encrypting the plaintext, and judging from redundancies the key can be easily recovered.
To crack the cipher, it would be helpful to know the length of the key. However if not, we can guess up to the length of the message. To start, generate k bins enumerated from 0 to $k - 1$ of letters for a key of length k. Each $(i \mod k)^{\text{th}}$ letter in the ciphertext gets put in the i^{th} bin.

Each of the bins now contains letters that were shifted by the same value. Perform frequency analysis on each of the bins.

3 The security should lie in the key, not the system

Due to Cryptography's origins in wartimes, having a encryption system secured is not very practical as it needs to be distributed and it could easily be stolen. The design of encryption schemes then emphasized keys to be its main security feature. It is not as compromiseable as an encrpytion system, which usually took form in some sort of machinery.

3.1 Kerchoff's Principle

> "The system must not be required to be secret, and it must be able to fall into the hands of the enemy without inconvenience."

3.2 Shannon's Maxim

> "The enemy knows the system."

4 Information Theoretic Security and Perfect Secrecy

4.1 The One Time Pad (OTP)

The **One Time Pad (OTP)** is the first example of a "secure" cipher, and it is very simple. The key space, plaintext space, and ciphertext space are all $\{0,1\}^n$. To encrypt a plaintext m, a key k is chosen and the ciphertext is defined as $c = E(k,m) = k \oplus m$. The decryption is equally as simple: $m = D(k,c) = k \oplus c$. The cipher is correct as $k \oplus k \oplus m = 0 \oplus m = m$.
Due to properties in XOR, when given the message m and the ciphertext c, the key can be retrieved easily as $k = m \oplus c$.

4.2 Perfect Secrecy

Claude Shannon, the father of Information Theory had the idea that a secure cipher should have the property that `the ciphertext should reveal no information about the plaintext`. In more rigourous terms:

> A cipher (E, D) over $(\mathcal{K}, \mathcal{M}, \mathcal{C})$ has perfect secrecy if for every distribution over $\mathcal{M}$, every $m \in \mathcal{M}$, and every $c \in \mathcal{C}$ such that $\mathbb{P}[C = c] > 0$,
>
> $$\mathbb{P}[M = m | C = c] = \mathbb{P}[M = m]$$

This essentially implies that the most powerful Adversary could learn nothing about the plaintext from simply the ciphertext (no CT-only attacks)

4.2.1 Alternative Definitions

Other ways to descrie perfect secrecy are as follows, they are equivalent and show alternate perspectives.

1. $\forall m_0, m_1 \; \forall c \; \mathbb{P}[C = c | M = m_0] = \mathbb{P}[C = c | M = m_1]$
2. $\forall m_0, m_1 \; \forall c \; \forall k \; \mathbb{P}[E(k, m_0) = c] = \mathbb{P}[E(k, m_1) = c]$

4.3 Perfect Secrecy as a Game

It is often more useful to describe Perfect Secrecy as a game that has an Adversary and a Challenger (or sometimes referred to as "Alice").
The Adversary selectively sends the Challenger two plaintexts m_1, m_2. The Challenger will pick a key k at random and randomly selects one of the m_1, m_2 to encrypt into ciphertext c and send it back to the Adversary. The task of the Adversary is to determine if c is encrypted from m_1 or m_2. We say that the encryption scheme that the Challenger is using is secure if the Adversary could not tell them apart and has to guess with a probability of $1/2$.

4.3.1 Monoalphabetic ciphers are not perfectly secure

Under the light of the game, if the Adversary sends the plaintexts `FAR` and `FEE`, any possible ciphertexts would either follow the form `XYY` or `XYZ`. It is then clear to see that the form `XYY` can only be encrypted from `FEE` and `XYZ` can only be encrpyted from `FAR`. Thus the Adversary can always come up with the correct answer when the Challenger uses this kind of cipher.

4.4 OTP is perfectly secret

We will proceed to prove this property using the second definition of perfect secrecy from section 4.2.1. We assume $\mathcal{K} = \mathcal{C} = \mathcal{M} = \{0,1\}^n$.
For any plaintext m and ciphertext c, there is exactly one key k such that $E(k,m) = c$. In otherwords, $|\{k \in \mathcal{K} | E(k,m) = c\}| = 1$. For any m_0,

$$\mathbb{P}[E(k, m_0) = c] = |\{k \in \mathcal{K} | E(k,m) = c\}| / |\mathcal{K}| = 1/2^n$$

We can apply the same argument to any m_1 that is not m_0. And as such, we can see that $\mathbb{P}[E(k, m_0) = c] = \mathbb{P}[E(k, m_1) = c]$.
Unfortunately, the perfect secrecy property of OTP only holds when the key-space is at least as big as the message space($|\mathcal{K}| \geq |\mathcal{M}|$). To quickly show this, the OTP cipher will have to first extend the key so it matches the length of the message. Suppose this is done with a function h, then the cipher is defined as $E(k,m) = k || h(k) \oplus m$ (where $||$ denotes concatenation). Therefore, a ciphertext

$c_{\overline{h}}$ constructed by $k||\overline{h}(k) \oplus m$ (where $\overline{h}$ is simply not h) is not generatable by E as we have defined. Thus, $\mathbb{P}[E(k,m) = c_{\overline{h}}] = 0$ while $\exists c\ \mathbb{P}[E(k,m) = c] \geq 0$. This violates our definition defined in 4.2.1.

4.5 Statistical Security

Due to the fact that OTP requires the same length of key as message, it is impractical. Consider encrypting a 1GB bitstream, a 1GB key will also be needed. To relax the "perfect" aspect, we will come up with new ways to quantify security.

4.5.1 Statistical Distance

Suppose we have two distributions $\mathcal{X}$ and $\mathcal{Y}$ over $\{0,1\}^m$, the statistical distance (aka. Total Variational Distance) between $\mathcal{X}$ and $\mathcal{Y}$ is defined as

$$\Delta(\mathcal{X},\mathcal{Y}) = \max_{T \subseteq \{0,1\}^m} |\mathbb{P}[\mathcal{X} \in T] - \mathbb{P}[\mathcal{Y} \in T]|$$

If $\Delta(\mathcal{X},\mathcal{Y}) \leq \epsilon$ for a negligible ϵ, then we can say that $\mathcal{X}$ and $\mathcal{Y}$ are ϵ-equivalent $\mathcal{X} \equiv_\epsilon \mathcal{Y}$.
In theory ϵ is a function $\epsilon : \mathbb{Z}^{\geq 0} \to \mathbb{R}^{\geq 0}$ that is considered non-negligible when $\exists d : \epsilon(\lambda) \geq 1/\lambda^d$ and negligible otherwise. However, in practice, we would consider ϵ to be a scalar, and something like $\epsilon \geq 1/2^{30}$ to be non-negligible as vulerabilities are likely to happen over 1GB of data. However, something to the likes of $\epsilon \geq 1/2^{80}$ is considered negligible as there are no vulnerabilities that would happen over the life of the key.

4.5.2 Statistical Security Defined

A cipher (E,D) is ϵ-statistically secure if for any two plaintexts m_0 and m_1, $E(\mathbf{k}, m_0) \equiv_\epsilon E(\mathbf{k}, m_1)$.

4.5.3 Statistical Insecurities

Let (E,D) be a cipher with keys that have one less digit than messages (suppose we are working with n bits in the message), then there are messages m_0, m_1 such that $\Delta(E(\mathbf{k}, m_0), E(\mathbf{k}, m_1)) \geq 1/2$.
To put it more simply, there are two messages where the set of their ciphertexts overlap by at most half. In the context of the game the Adversary simply has to find those two messages to send to the Challenger. If the ciphertext he receives back is not in the overlap, then the Adversary is certain of the source message. Otherwise, the Adversary guesses. There is at least $1/2$ chance that the ciphertext is not in the overlap, so the success rate of the Adversary is at least $1/2 \cdot 1 + 1/2 \cdot 1/2 = 3/4$.

5 Stream Ciphers and Pseudorandom Generators

To counteract the issue with the keysize having to be the size of the message in the OTP case, we replace the random key with a pseudorandom that "looks random"

5.1 Pseudorandom Generators

A pseudorandom generator (PRG) G is a function that generates strings of arbitrary length digit by digit that "look random" from a smaller truly random seed.

$$G : \{0,1\}^m \to \{0,1\}^n \qquad n >> m$$

G is required to be efficiently computable and deterministic. In our context efficient is simply meant as something that runs in polytime. The deterministic factor is needed for recovering the output for decryption when used in a cipher.

5.1.1 One-Way Functions

A function $f : \{0,1\}^* \to \{0,1\}^*$ is a one-way function if it is easy to compute for any $x \in \{0,1\}^*$, and it is hard to invert for a random $\{0,1\}^*$.
Due to this property, we can build a PRG with any one-way function. However, we do not have any provably one-way functions, as the existence of so would imply $P \neq NP$.
Some of the candidates for a one-way function are essentially hard problems such as:

- Integer factorization (given $x = pq$, find p, q)
- Subset sum
- Discrete logarithm

5.1.2 Security Parameter

PRG's are parametrized by a security parameter λ, where the PRG becomes increasing secure with increasing λ. Seed and output lengths increase with λ.

5.1.3 PRG Predictability

PRG's should be unpredictable. Meaning, given its previous outputs, we should not be able to predict its future outputs. We say that a PRG is predictable at position i if there exists an Algorithm A that is polytime in λ such that

$$\mathbb{P}[A(\lambda, G_\lambda(\mathbf{k})[:i]) = G_\lambda(\mathbf{k})[i+1]] \geq \frac{1}{2} + \epsilon(\lambda)$$

for some non-negligible function $\epsilon(\lambda)$. In simpler terms, a PRG is predictable if there is an algorithm that can predict the next bit of output given all of its previous inputs with greater than chances of guessing $(1/2)$.

Unpredictable PRG's are secure, but we will explore this notion later on when the concept of security of PRG's are formally introduced.
Yao's theorem states that if G is unpredictable at any position i, then G is a secure PRG.

5.2 Stream Ciphers

A stream cipher utilizes a PRG to generate a keystream, where each digit of the key is used to encrypt a digit in the plaintext with an XOR operation. It can be considered as OTP with a PRG keystream as key: $E(k,m) = G(k) \oplus m$, $D(k,c) = G(k) \oplus c$.
Stream ciphers cannot be perfectly secure, so we will need to redefine what it means to be secure. Our goal is to define what it means to have a distribution that is indistinguishable from random (uniform distribution). We know that we can work with the definition of ϵ-equivalence, but this is impossible when we have fewer keys/seeds than 2^n (from a space smaller than $\{0,1\}^n$). Instead, we are going to define the notion of computational indistinguishability.

5.3 Computational Indistinguishability

Two families of distributions are said to be computationally indistinguishable if no efficient algorithm can tell the difference between them except with a small probability. (From Wikipedia)
Suppose we have an algorithm A that will be used and referred to as a statistical test. Its purpose will be to attempt to tell apart distributions, it will run in probabilistic polynomial time (PPT), where $t(|x|) \in O(|x|^k)$ for some k.
Disregarding the implementation of any A, we are concerned with the output of A. Namely, we consider some conditions, and then if the conditions are met, A returns 1. Otherwise, it returns 0.
Suppose P_1 and P_2 are two distributions over $\{0,1\}^n$, we say that P_1 and P_2 are computationally indistinguishable iff for every efficient statistical test A,

$$|\underset{x \leftarrow P_1}{\mathbb{P}}[A(x) = 1] - \underset{x \leftarrow P_2}{\mathbb{P}}[A(x) = 1]| < \epsilon$$

for a negligible ϵ. Analogously, we can also say that G is secure if $\{k \leftarrow K : G(k)\} \approx_p uniform(\{0,1\}^n)$. The p here stands for polynomial bound on running time of A. More generally $P_1 \approx_{t,\epsilon} P_2$ denotes every A running in at most t time cannot differentiate P_1 from P_2 with probability greater than ϵ.

5.4 Adversarial Advantage

For a given statistical test A on $\{0,1\}^n$, The Adversary's advantage on a particular PRG G is defined as the following:

$$Adv_{PRG}[A,G] = |\mathbb{P}[A(G(\mathbf{k})) = 1] - \mathbb{P}[A(\mathbf{r}) = 1]|$$

As Adversarial advantages are constructed from probabilities, we can see that $0 \leq Adv_{PRG}[A,G] \leq 1$. If the advantage is close to 1, then A can tell the difference between the output of G and $\mathbf{r}$. If the advantage is close to 0, then A cannot.

5.5 PRG Security

A pseudorandom generator G is said to be secure if every efficient statistical test A satisfies that $Adv_{PRG}[A,G] < \epsilon$ for a negligible ϵ.

It is not known whether if there are provably secure PRG's, the existence of them would imply $P \neq NP$.

If a PRG is unpredictable, then it is considered secure. To show this we will prove the contrapositive: If a PRG G is predictable, then it is insecure. Suppose we have an efficient algorithm A such that

$$\mathbb{P}[A(G(\mathbf{k})[:i]) = G(\mathbf{k})[i+1]] \geq \frac{1}{2} + \epsilon$$

with a non-negligible ϵ. We define a statistical test B where it outputs 1 if A outputs the correct output (guesses the next output of G) and 0 otherwise. Then $Adv_{PRG}[B,G] = |\mathbb{P}[B(\mathbf{k}) = 1] - \mathbb{P}[B(G(\mathbf{k})) = 1]| = |1/2 - (1/2 + \epsilon)| \geq \epsilon$, of which ϵ is non-negligible so therefore G is not secure.

5.6 Semantic Security

A cipher E is said to be semantically secure if for all efficient A, $Adv_{SS}[A,E]$ is negligible. Alternatively, for all explicit m_0, m_1, the distributions induced by both are computationally indistinguishable.

As such, using a secure PRG in a stream cipher means the resulting cipher is semantically secure.

One way to see this is that for any m_0, the distribution $m_0 \oplus G(\mathbf{k})$ is computationally indistinguishable to $m_0 \oplus \mathbf{r}$, which is computationally indistinguishable to $m_1 \oplus \mathbf{r}$, which is computationally indistinguishable to $m_1 \oplus G(\mathbf{k})$. Therefore by transitivity $m_0 \oplus G(\mathbf{k}) \approx_p m_1 \oplus G(\mathbf{k})$.

6 Stream Cipher Vulnerabilities

6.1 Two Time Pad

A repeated use of the same key in OTP is insecure. Consider two ciphertexts $C_1 = m_1 \oplus PRG(k)$ and $C_2 = m_2 \oplus PRG(k)$. By applying $C_1 \oplus C_2$ we get $m_1 \oplus m_2$. There is enough redundancy in English that the messages can be separated to obtain m_1 and m_2 separately, thus making it a vulnerability. Project Venona was a historical example where Two Time Pad was used.

6.1.1 MS-PPTP

In Windows NT communications between clients and servers, messages on the client side and server side are both sent encrypted with the same key. As such, when catching messages from both ends an attacker simply needs to XOR the messages and the payloads from both ends will be exposed. For more information, visit `https://en.wikipedia.org/wiki/Point-to-Point_Tunneling_Protocol#Security`

6.1.2 802.11b WEP

WEP encryption essentially uses a stream cipher where its input to the PRG is a 24 bit IV (used as an unique value) concatenated with a 104 bit key. Each payload that is sent is encrypted with an IV which is incremented every time since the device's last reset. However, 24 bits is not enough to guarantee uniqueness over time as the counter repeats after 2^24 frames, which is only roughly 17 million frames (or the number of rgb values there are). For more information, visit `https://en.wikipedia.org/wiki/Wired_Equivalent_Privacy#Weak_security`

6.2 Malleability

Suppose an attacker knows part of the plaintext in a ciphertext stream, then the attacker could modify the ciphertext stream before redirecting it to its intended receiver. Suppose we know that the cipher stream is an encrypted email from Bob to Alice, then the attacker could modify the email content and replace `Alice` to be `Eve`. This can be done as $(k \oplus m_0) \oplus m_0 = k$ and we can inject a different plaintext m_1 simply by $(k \oplus m_0) \oplus m_0 \oplus m_1 = k \oplus m_1$.

7 Examples of Stream Ciphers

7.1 Linear Feedback Shift Register

An Linear Feedback Shift Register (LFSR) is easy to implement (and often implemented) in hardware. The algorithm is roughly as follows:

1. Initialize with the seed
2. For a few selected positions in the register, extract the values and XOR all of them, call the resulting value o_i
3. `shr` the register, set msb to be o_i
4. repeat steps 2 and 3 as desired

In practice LFSR's are used in combination. For example, DVD encryption (CSS) uses 2 LFSR's, GSM (A5/1,2) uses 3, and Bluetooth (E0) uses 4. Due to its linear nature, a n-bit LFSR can be broken after getting 2n bits of output.

The glibc random function uses a Linear Congruential Generator (LCG), which is similar to an LFSR and still vulnerable.

7.2 Content Scrambling System

The Content Scrambling System (CSS) is used for DVD encryption. It is built with two LFSR's and a 40 bit key due to a US law that prohibits exporting crypto that uses keys longer than 40 bits.
The 40 bit (5 byte) key is broken down into a 2-byte part and a 3-byte part named k_2, k_3 respectively. After prepending the keys by `1`, they are each fed to a 17-bit LFSR and a 25 bit LFSR. Each LFSR outputs 8 bits which are summed up together mod 256. The resulting 8-bits is used and the carry bit is passed to the next iteration.
Due to the fact that keys are only 5 bytes wide, the cipher can be brute forced.

7.3 Rivest Cipher 4

While the LFSR is efficient in hardware, it isn't so in software. The Rivest Cipher 4 (RC4) addresses this by being an efficient software solution that avoids the use of LFSR's.
It is used in HTTPS and WEP, but nonetheless it is insecure due to some vulnerabilities. In particular, there is a bias in the initial output that renders its PRG predictable.
For more information, visit `https://en.wikipedia.org/wiki/RC4#Biased_outputs_of_the_RC4`

7.4 Salsa 20

The Salsa 20 Stream Cipher is one that is selected by the eSTREAM project and its novelty comes with the usage of a nonce in encryption. As of writing there are no published attacks on this cipher.

7.5 Others

There are other PRG's such as the Blum-Blum-Shub PRG whose security relies on the notion that factoring is difficult, and the Subset Sum PRG whose security relies on the notion that subset sum is difficult
PRG's whose sole purpose is to generate random numbers work by continuously adding entropy to its internal state, which can come from hardware rng or peripheral inputs such as from keyboard or mouse.

8 Random Functions and Permutations

A random function $f : X \to Y$ is constructed by picking a random $y \in Y$ for each $x \in X$ and setting $f(x) = y$.

For example, a random function $f : \{0,1\}^n \to \{0,1\}^n$ can be represented by a random binary string of length $n \cdot 2^n$. Under this construction, a random function $f : X \to X$ is a random one-to-one function that we call a permutation. Let $Funs[X,Y]$ denote the set of all random functions $X \to Y$, and $Perms[X]$ denote all permutations on X.
An algorithm A that interacts with a function f is denoted as A^f. The interaction is done by sending values $x_1, x_2, ..., x_q$ (that we usually call "querying") to the function and in turn receiving $f(x_1), f(x_2), ..., f(x_q)$.
For an efficient A^f, as it runs polynomially in time it will only be able to make polynomially many queries.
Each query can depend on the answers of pervious ones. It could also depend on A^f's randomness. This is called black-box or oracle access.

8.1 Pseudorandom Function Generators

A function $F : K \times X \to Y$ is said to be a pseudorandom function generator (PRF) where for a fixed $K = k$, $F(k, \cdot)$ is a pseudorandom function. F must be deterministic and efficiently computable. In order for F to be considered secure, it must satisfy any efficient statistical test A, that

$$Adv_{PRF}[A,F] = \left| \mathop{\mathbb{P}}_{k \in K}[A^{F(k,\cdot)} = 1] - \mathop{\mathbb{P}}_{f \in F^*}[A^f = 1] \right| \leq \epsilon$$

Letting $F^* := Funs[X,Y]$ and for a negligible ϵ.
Additionally, a PRF $F : K \times X \to X$ is a pseudorandom permutation generator (PRP) if $\forall k \in K$ $F(k, \cdot)$ is a permutation and there is a deterministic efficient inversion algorithm $D : K \times X \to X$.
The notion of security of PRP's is defined in a similar way to that of a PRF, but instead of indistinguishability against random functions, we need indistinguishability against random permutations:

$$Adv_{PRP}[A,F] = \left| \mathop{\mathbb{P}}_{k \in K}[A^{F(k,\cdot)} = 1] - \mathop{\mathbb{P}}_{p \in P^*}[A^p = 1] \right| \leq \epsilon$$

Letting $P* := Perms[X]$ and for a negligible ϵ.
Suppose we have a secure PRF $F : K \times X \to \{0,1\}^{128}$, and we construct a PRF G that outputs the output of F if the input x is nonzero $G(k,x) = F(k,x)$, otherwise it will output all zeroes $G(k, 0^{128}) = 0^{128}$. We define a statistical test A that queries the PRF with 0^{128} which returns 1 if the PRF returns 0^{128}, and 0 elsewise. With such A we can see that A's advantage over G using 0^{128} as the query is $Adv_{PRF} = |1 - 1/128| = 127/128$, which is by no means negligible.
Suppose we have $X = \{0,1\}$, then $Perms[X]$ contains two functions: mapping to the same bit or the opposite. Only these two are allowed as it needs to be invertible. With a key space of $\{0,1\}$ and message space of $\{0,1\}$, the PRP defined as $F(k,x) = x \oplus k$ is a secure PRP. However, F is not a secure PRF as $Funs[X,X]$ contains 4 functions instead of 2 because there are no invertibility constraints.

8.1.1 PRF Switching Lemma

Any secure PRP is also a secure PRF for sufficiently large $|X|$. Let E be a PRP over (K,X), then for any q-query adversary A:

$$|Adv_{PRF}[A,E] - Adv_{PRP}[A,E]| < \frac{q^2}{2|X|}$$

The condition for sufficiently large $|X|$ manifests itself as $q^2/2|X|$.

8.2 Constructing a PRG from PRF

Suppose we have a secure PRF $F : K \times \{0,1\}^n \to \{0,1\}^n$, and letting $G : K \to \{0,1\}^{nt}$ be a PRG constructed in the following way:

$$G(k) = F(k,0)||F(k,1)||...||F(k,t-1)$$

Then G is also a secure PRG. This is because $F(k, \cdot)$ is indistinguishable from a random function and in turn the corresponding PRG output is indistinguishable from random.
A key property of this PRG is that it can be computed in parallel as each $F(k,i)$ are independent of each other.

8.3 Constructing a PRF from a PRG

Suppose we let $G : K \to K^2$ be a secure PRG, then we can define a 1-bit PRF F where $F(k, x \in \{0,1\}) = G(k)[x]$, letting the output $G(k)$ be defined as an array of two blocks with index 0 or 1.
If G is a secure PRG then F is a secure PRF.
We can extend this notion further. Supose we define $G_1 = G(G(k)[0])||G(G(k)[1])$, so $G_1 : K \to K^4$ and we get a 2 bit PRF $F(k, x \in \{0,1\}^2) = G_1(k)[x]$.
Similarly we can create a 3 bit PRF as $F(k, x \in \{0,1\}^3) = G(G_1(k)[0])||G(G_1(k)[1])$, and so on and so forth. The generalization of this construction is called the Goldreich-Goldwasser-Micali construction.

9 Block Ciphers

A Block cipher is a SES that encrypts a fixed-length of bits (called a block) at a time. Block ciphers are often used as primitives and are widely used to encrypt bulk data.

9.1 Iterated Block Ciphers

An iterated block cipher is one that applys a transformation on a message multiple times. The input of the transformation at a specific iteration uses a key and the output of the previous iteration.
The original key k is expanded into n chunks, each chunk k_i expanded from the original key is used in an iteration. The original input m is passed into a round function $R(k_1, m) = r_1$ for the first round, then the second round computes $R(k_2, r_1) = r_2$. This is done until the last round where $R(k_n, r_n - 1) = c$.

9.1.1 Feistel Network

An example of an iterated block cipher is a Feistel network. It has a basic mechanism which can be chained multiple times and it is used in many applications.
Suppose we have a secure PRF $F : K \times \{0,1\}^n \to \{0,1\}^n$. For n rounds we pass a specific key k_i to the PRF to form $f_i(m_i) = F(k_i, m_i)$.
For the input block (before any rounds, it would be the plaintext), it is divided in the middle to form a left half L_i and a right half R_i. The transformation of L_i, R_i toward the next iteration are as follows:

$$L_{i+1} = R_i$$
$$R_{i+1} = f_i(R_i) \oplus L_i$$

For completeness, an inverse operation to undo a layer is as follows:

$$R_i = L_{i+1}$$
$$L_i = f_i(L_{i+1}) \oplus R_{i+1}$$

To invert an entire Feistel network, all that needs to be done is to apply the keys in reverse order. For more details, consider the following link `https://en.wikipedia.org/wiki/Feistel_cipher#Construction_details`

9.2 Modes of operation

Block Ciphers operate on one block of plaintext at a time, often a plaintext may be more than just a block long. Therefore, there are operating modes for block ciphers that allow it to encrypt multiple blocks. These operating modes are agnostic to the specific cipher used.
For a more comprehensive list of operating modes, refer to `https://en.wikipedia.org/wiki/Block_cipher_mode_of_operation`

9.2.1 Electronic Code Block

Electronic Code Block (ECB) is the most naive operating mode. Each block is encrypted independently and there are security implications for this mode of operation. Due to independence, both encryption and decryption are parallelizable.
Namely: for blocks that are encryptions of the same plaintext, their corresponding ciphertexts are the same as well.

9.2.2 Cipher Block Chaining

For Cipher Block Chaining (CBC), each encryption of a block of plaintext is XOR'd with the ciphertext of the previous block. For the very first block, an initialization vector (IV) is used instead of the ciphertext of the previous block.
Due to the dependence on the previous block, encryption in a CBC cipher will not be able to be parallelized, but decryption can be parallelized with no issue.
If there is are corruptions of the ciphertext (from transmission, for example), then the block and block after will be corrupted when attempting decryption.
For any $L > 0$, if E is a secure PRP over (K, X), then E_{CBC} is a semantically secure under CPA over (K, X^L, X^{L+1}). In particular, for a q-query adversary attacking E_{CBC}, there exists a PRP adversary such that

$$Adv_{CPA}[A, E_{CBC}] \leq 2Adv_{PRP}[B, E] + \frac{2q^2L^2}{|X|}$$

Note, using CBC is only secure if $q^2L^2 << |X|$. The $2q^2L^2/|X|$ term is the probability that the inputs to $F(k,)$ are the same.
CBC is not CPA-secure if the attack can predict the IV. Previously in SSL/TLS 1.0, the IV for a particular record is the last CT block of the previous record, which makes it predictable.

9.2.3 Counter Mode

Counter Mode (CTR) operates by inputting the key and a counter (of the current block) into the cipher, and then XOR with the plaintext at that block to obtain the final ciphertext.
The counter mode has independent ciphertext blocks, so both encryption and decryption are parallelizable.
For any $L > 0$, if F is a secure PRF over (K, X, X) then E_{CTR} is a sematically secure cipher over (K, X^L, X^L). In particular, for any efficient adversary A targetting E_{CTR}, there exists an efficient PRF adversary B such that

$$Adv_{ss}[A, E_{CTR}] = 2Adv_{PRF}[B, F]$$

Since $Adv_{PRF}[B, F]$ is negligible, $Adv_{ss}[A, E_{CTR}]$ will be negligible too.
For a q-query adversary A (in chosen-plaintext attacks) targetting E_{CTR}, there exists an efficient PRF adversary B such that

$$Adv_{CPA}[A, E_{CTR}] \leq 2Adv_{PRF}[B, F] + \frac{2q^2L}{|X|}$$

CTR is only secure if $q^2L << |X|$, which is better in comparison to CBC.

9.3 One-time Key Security

The goal of this section is to build a secure encryption from a secure PRP. For an adversary, we consider an adversary that sees only one ciphertext (and thus only encryption from one key), and its goal is to learn about the plaintext from the ciphertext (breaching semantic security).
ECB is not semantically secure. The adversary sends the challenger two plaintexts where each are two blocks long. One with different blocks and one with the same block(ex. "HELLO HELLO", "HELLO WORLD"). The response will be either be a ciphertext that has different halves or the same half, which gives the adversary total advantage.

9.4 Many-time Key Security

In this context, the same key is used multiple times to encrypt multiple items. For example, using the same AES key to encrypt many files in a filesystem or packes in a network transmission. In this setting, an adversary can obtain encryptions of arbitrary messages of its choice, thus it is called a chosen plaintext attack. The adversary's goal is to still break semantic security.
For a CPA, the semantic security game is defined slightly differently. The game begins with the challenger committing to a key k. The adversary then queries adaptively q plaintexts and receives their encryptions. Afterwards, the adversary sends m_0, m_1 to the challenger and the guessing game ensues.
If a key is used multiple times and the output will be the same with the same plaintext, an adversary will be able to send one of the q queries again in the game round, thus giving the adversary total advantage
One way to account for this is to implement randomized encryption, where there is some randomness in the algorithm such that the same key and same plaintext on different runs provide different outputs. This is done usually with the CT being longer than the PT, the difference in bits are random bits.
Another way to account for this is to apply nonce-based encryption. A nonce is something that changes from message to message. There are different way to actualize this. One way is to choose a random number as the nonce each time, another another is to use some sort of counter as a nonce (for example, refer to CTR). In the game setting, each encryption that the challenger applies uses distinct nonces, but the nonces should be given by the adversary.

10 Data Encryption Standard

Data Encryption Standard (DES) uses 16 rounds of Feistel Network as its core. Before the feistel network is used, a permutation is applied to the data. The permutation is undone after using the network. The permutation has no cryptographic purpose other than facilitating loading sizes of blocks on older hardware. The input to DES is a 64-bit block of plaintext and 56-bit long key.

10.1 DES Round function

In the Feistel network, the function f_i used at each layer takes in a 48-bit key and a 32-bit sub-block. The 32-bit sub-block is first expanded to match the length of the key, then a bitwise XOR is carried out between the sub-block and the key. The result is split into 8 chunks (6 bits each), each chunk is mapped into 4 bits via a lookup table (S-Boxes) and joined together to form 32 bits. This 32 bit output undergoes another permutation (P-box) before being returned.

10.2 S-Boxes

S-Boxes are implemented as a look-up table, where the input 6-bits are split into the middle 4 bits and the outer 2 bits. Together, they form indices to index into the lookup table which has 4-bit entries.
The selection of values in the S-table is crucial. If the table is linear, then the feistel network becomes trivially decipherable. The notion of linearity is defined as the ability to represent the table as a linear system of equations associating the indices with the values of the table. (i.e. $S_i(\mathbf{x}) = A_i\mathbf{x} \pmod 2$) can be represented as a linear system of equations,

10.3 Exhaustive Search for Key

Suppose DES is an ideal cipher where there are 2^{56} random invertible functions, then for all messages and ciphertexts there is at most one key k such that $c = DES(k, m)$. The uniqueness of a key not exactly a good idea, as if we are guranteed that there is only one key, then finding simply 1 key that decrypts a ciphertext will decipher the rest of the future ciphertexts too.

10.4 Stacking DES

To counteract exhaustive search attacks in DES, one method is stacking DES together three times in the follow way:
Let $E : K \times M \to M$ be a block cipher, and we define $\mathbf{3E} : K^3 \times M \to M$ as:

$$\mathbf{3E}((k_1, k_2, k_3), m) = E(k_1, D(k_2, E(k_3, m)))$$

Effectively, 3DES encrypts a message m with k_3, decrypts with k_2, and then encrypts with k_1. One interesting note to make is that when $k_1 = k_2 = k_3$, we are back to a regular DES. This means that we need 3 time the length of the original key, which is now $3 * 56 = 168$ bits long.
However, two rounds of DES $2DES((k_1, k_2), m) = DES(k_2, DES(k_1, m))$ would be insecure. 2DES has a key length of $2 * 56 = 112$ bits long. A naive approach would be to brute force every possible key, so in 2^{112}

time. However, with a slight exploit we can see that if we query m and receive c, then we can approach breaking the cipher in both directions: exhaustively computing $DES(k_1, m)$ and exhaustively computing $DES^{-1}(k_2, c)$. Each of which takes 2^{56} time. However, there will be a match in decrypting and encrypting, which would validate the (k_1, k_2) pair. This kind of attack is called a **Meet In The Middle (MITM)** attack. A more elaborate description of this type of attack follows:

11 Attacks

11.1 Meet In The Middle Attack

For 2DES, an Adversary exploits the property that given $c = DES(k_2, DES(k_1, m))$, then $DES^{-1}(k_2, c) = DES(k_1, m)$. This means all we need is to find a (k_1, k_2) pair that creates this match.
The Adversary proceeds by collecting every possible decryption of c and stores such in a $2^{56} \times 2$ table where row contains an entry of the key used and the decrypted value. This table will need to be sorted by value, which as we know happens at a fastest time complexity of $O(nlogn)$. Thus generating the sorted table of values and corresponding keys take $2^{56}log2^{56}$.
After this table is generated, the Adversary tries every possible encryption of m and looks up the table for a matching value. As there are 2^{56} possible encryptions to try and binary search takes $O(logn)$ time, we have another runtime of $2^{56}log2^{56}$. Combined, the runtime is lesser than 2^{63} which is way lesser than a naive 2^{112}.
3DES is susceptible to this kind of attack as well, however attaching 3DES with a MITM attack will have a runtime of 2^{118}.

11.2 Side Channel attack

Side channel attacksare defined by attacks that do not directly target the software system. Often this is due to hardware implications of the code. For ciphers, the time and power used in a piece of hardware can reveal key information about the process. The two examples of this are:

- the time to execute different instructions are different
- for validating array of input, the longer it takes, the more correct the first items are. This is because cmp is often implemented by a loop that breaks out as soon as a mismatch is found.

11.3 Fault Attack

A fault attack attemptes to create computing errors in the system to reveal secret internal states such as keys.

11.4 Linear Cryptanalysis

Linear Cryptanalysis is a known plaintext attack. Linear relationships between keys, plaintext, and ciphertexts are studied and explioted

11.5 Differential Cryptanalysis

In the context for block ciphers, a differential cryptanalysis is a set of techniques for tracing differences where the cipher exhibits what is know as non-random behaviour and exploiting such details to recover the secret key.

11.6 Quantum Attacks

For a classical computer, the best generic search algorithm time is $O(n)$ for n inputs. However, for a quantum computer (whether it is possible to built is unknown but not the focus of this text), it can be done in $O(\sqrt(n))$ with Grover's algorithm. This means a previously 2^{64} time attack on a classical computer only needs 2^{32} with a quantum computer. However, an easy way to account for this issue is to simply increase key-lengths.
For collison attacks, the Grover's algorithm can complete an attack in $O(|T|^{\frac{1}{3}})$ time compared to a generic attack on a classical computer running in $O(|T|^{\frac{1}{2}})$.

12 Advanced Encryption Standard

Advance Encryption Standard (AES) is also known as a Rijndael cipher. It is a classic example of a substitution-permutation network and has a blocksize of 128 bits. The input of AES is a 4×4 matrix of one byte in each entry, and the key size is either one of 128, 192, or 256.
Depending on the key size, the rounds in AES are different as well. Namely: AES-128 has 10 rounds, AES-192 has 12 rounds, and AES-256 has 14 rounds.
Before each round, a piece of the expanded key is XOR'd with the input block (called AddRoundKey). AddRoundKey is applied to the plaintext block before applying the first round.

12.1 AES Round Function

The AES Round function has 4 steps:

1. SubBytes: a non-linear substitution with an S-Box (similar idea to S-Boxes in DES)
2. ShiftRows: a transposition step where the last three rows of the input are shifted cyclically a certain number of steps
3. MixColumns: A linear mixing operation which operates on the columns of the input
4. AddRoundKey

12.2 Properties

There are two major properties in AES and substitution-permutation networks in general:
Confusion: every bit of ciphertext depends on several parts of the key. Ideally, each bit of ciphertext depends on the whole key.
Diffusion: a change in one bit in the plaintext would reflect a large change (on average half the bits) in the ciphertext.

13 Message Authentication Code

A message authentication code is useful where message integrity is to be maintained. Integrity is defined as the inability of an attacker to modify a ciphertext in such a way that the decrypted plaintext changes meanings. For example, message integrity is needed to protect public binaries on disk, or protect banner ads on web pages.
A MAC $I = (S, V)$ defined over (K, M, T) is a pair of algorithms for signing $S : K \times M \to T$ (which generates a t tag) and verifying $V : K \times M \times T \to \{0, 1\}$ the encryption (by outputting 0 or 1).
One simple example of a MAC is a Cyclic Redundancy Check (CRC). It is designed to detect random errors in the encryption and not malicious errors. An attacker can easily modify the message and recompute the CRC.

13.1 Constructing a MAC from a PRF

A simple construction is to let $S(k, m) := F(k, m)$, where F is a secure PRF. $V(k, m, t)$ will output 1 if $t = F(k, m)$ and 0 otherwise. As long as the PRF is secure and the PRF output is long enough, the resulting MAC will be secure as well.

13.2 MAC Security

For an attacker, the attacker is capable of giving a chosen message attack, where for a certain amount of queries $m_1, m_2, ..., m_q$ the attacker can obtain all corresonding tag t_i's. The goal of an attack is existential forgery, where a new message-tag pair (that does not exist from message-tag pairs that the attacker has already seen) is generated that is verifiable by V.
The notion of a secure MAC is defined in a game sense as the following:

- Challenger commits to a key k
- Adversary queries q messages to the Challenger
- Challenger computes S and returns q corresponding tags
- Adversary sends newly forged (m, t) to Challenger
- Challenger computes V and outputs $b = 1$ if forgery is successful

A MAC scheme $I = (S, V)$ is said to be secure if for all efficient A, $Adv_{MAC}[A, I] = \mathbb{P}[b = 1] \leq \epsilon$ for a negligible *epsilon*.
If $F : K \times X \to Y$ is a secure PRF and $|Y|^{-1}$ is negligible, then the MAC I_F constructed from it is also secure. In particular, for every efficient MAC adversary A, there exists an efficient PRF adversary B such that

$$Adv_{MAC}[A, I_F] \leq Adv_{PRF}[B, F] + \frac{1}{|Y|}$$

As such, I_F is secure as long as $|Y|$ is large.

13.3 MAC Expansion/Truncation

Suppose we use a PRF to construct a MAC, we need some sort of method to generate a tag for an entire payload, not just a block. Thus questions regarding expansion and truncation of MAC outputs arise.

13.3.1 Truncation

MAC truncation can be easily achieved by taking the first w desired bits in the original MAC output. Security in truncated MAC outputs are preserved as long as 2^{-w} is a negligible amount.

13.3.2 Expansion

Often expansion of MAC outputs are computed with similar styles to how block ciphers operate, which will be discussed in the coming section.

13.4 MAC Expansion Constructions

13.4.1 CBC-MAC

Suppose we have a PRF $F : K \times X \to X$, we define a new PRF $F_{ECBC} : K^2 \times X^{\leq L} \to X$. For an input message m, each message block $m[i]$ is XOR'd with the outputs of the previous block before being fed into F for the result. The first block $m[0]$ is XOR'd with 0^n (a zero IV), and at the very end, the result is passed through F with another independently chosen key. If the final encryption is not computed, the function is called a raw CBC function.
For every efficient q-query PRF adversary A attacking F_{ECBC}, there exists an efficient adversary B such that

$$Adv_{PRF}[A, F_{ECBC}] \leq Adv_{PRF}[B, F] + \frac{2q^2}{|X|}$$

13.4.2 CMAC

A cipher-based MAC (CMAC) is similar to CBC-MAC but it does not apply the last encryption step. Instead, the MAC uses two keys. The first key is used in the underlying PRF, and the second key is XOR'd with the last block before XOR'd with the outputs of the previous block.

13.4.3 NMAC

Suppose we have a PRF $F : K \times X \to X$, we define a new PRF $F_{NMAC} : K^2 \times X^{\leq L} \to k$. For each block $m[i]$ in the message m, it is passed to F with the output of the previous block as the key. For the first block $m[0]$, the original key k is used as the key for F. After passing through all of the blocks, we obtain an output t. This process is called a cascade function. However, a cascade function is not a secure MAC. The next step involves padding the output of the cascade function with a fixed pad `fpad` and sending $t||$`fpad` to F with another independently chosen key to derive the final tag.
For every efficient q-query adversary A attacking F_{NMAC}, there exists an efficient adversary B such that

$$Adv_{PRF}[A, F_{NMAC}] \leq Adv_{PRF}[BF] + \frac{q}{2|K|}$$

13.4.4 PMAC

A parallel MAC is an incremental construction where each i^{th} block of the payload is XOR'd with an easy to compute function $P(k, i)$. Then every processed block except the last are each passed through a PRF $F(k_1, \cdot)$. All the blocks are XOR'd together before one last pass through of $F(k_1, \cdot)$ to provide the tag.
Suppose the F used in the PMAC is a PRP, we can update the tag when a block changes. Suppose that for a message m, the j^{th} block is updated ($m[j] \to m'[j]$), the update to the tag is done as follows:

$$\begin{aligned} t' = F(k_1, & \\ & F^{-1}(k_1, t) \\ & \oplus F(k_1, m[j] \oplus P(k, j)) \\ & \oplus F(k_1, m'[j] \oplus P(k, j))) \end{aligned}$$

Note that this only works for any j except for the last. However, similarly we can add new blocks and remove blocks an still update the tag.
For every efficient q-query adversary A attacking F_{PMAC}, there exists and efficient PRF adversary B such that

$$Adv_{PRF}[A, F_{PMAC}] \leq Adv_{PRF}[B, F] + \frac{2q^2L^2}{|X|}$$

13.4.5 Discussion

Pure CBC and cascade functions are not secure MAC's. This is because MAC's with this kind of construction can be forged with one chosen message query. For example, `cascade`(k, m) can be extended to `cascade`$(k, m||w)$ for any w.
For a raw CBC function, the adversary can approach in the following manner:

- Choose an arbitrary one-block messsage m
- Request a tag $t = F(k, m)$
- Output t as MAC forgery for the 2 block message $(m, t \oplus m)$

As a matter of fact:

$$\begin{aligned} & rawCBC(k, (m, t \oplus m)) \\ &= F(k, F(k, m) \oplus (t \oplus m)) \\ &= F(k, t \oplus (t \oplus m)) \\ &= t \end{aligned}$$

Suppose $F_{BIG} : K \times X \to Y$ be a PRF that has the extension property

$$F_{BIG}(k, x) = F_{BIG}(k, y) \to F_{BIG}(k, x||w) = F_{BIG}(k, y||w)$$

A generic attack could run in the following way:

- query $|Y|^{1/2}$ random messages in X and obtain their respective tags.
- find a collision $t_u = t_v$ for $u \neq v$. (There will be one that exists with high probability by the birthday paradox)
- choose some w and query $t := F_{BIG}(k, m_u||w)$.
- Output forgery $(m_v||w, t)$.

This attack works because $F_{BIG}(k, m_v||w)$ is the same as $F_{BIG}(k, m_u||w)$. The time that it takes to execute this attack is dependent on tag collision, which is probabilistic.

13.5 MAC Padding

As MAC's usually have specific blocksizes, for payloads that are not to a multiple of a blocksize, padding is needed. Zero padding is insecure. For security, padding must be invertible.
The ISO standard specifies to pad with 1 and then all 0's. A new dummy block if it is needed. The 1 would denote the beginning of the pad.

14 Hashing

A hash function is defined as $H : M \to T$ where $|M| >> |T|$. Due to the difference in size of M and T, we are bound to have different elements in M map to the same value in T. In the context of hash functions, we call this a collision.
More formally, a collision for a hash function H is defined as a pair $m_0, m_1 \in M$ such that $H(m_0) = H(m_1)$ and $m_0 \neq m_1$. A hash function is said to be collision resistant if for all efficient adversary A,

$$Adv_{CR}[A, H] = \mathbb{P}[\texttt{A outputs collision for H}] < \epsilon$$

for a negligible ϵ.

14.1 Birthday Attack

Suppose we have a hash function $H : M \to \{0,1\}^n$, a generic algorithm can find a collision in $O(2^{\frac{n}{2}})$ time, and is carried out in the following manner:

- Sample $2^{\frac{n}{2}}$ random messages in M
- For each of the messages sampled, compute and save its hash
- Look for a collision within the hashes computed
- If no collision has been found, start over

This attack works because of the birthday paradox. The paradox states that if there is a upper bound B, then when there are $1.2 \times B^{\frac{1}{2}}$ samples, then the probability that there is a collision is at least 50%. Because the odds are 50%, we expect two iterations of the attack to generate a collision.

14.2 Merkle-Damgard Paradigm

The Merkle-Damgard iterated construction (MD) is a way to provide secure hashing over many blocks of input. The hash of a block is provided by the output hash of the previous block (IV if no previous block) and itself given to a hash function.

The collision resistance of a MD construction depends on its underlying hashing function. Namely, for a MD hashing function H built upon h, if h is collision resistant, then so is H.