Summer of Science 2024

Midterm Report

Vivek Vardhan K

24 June 2024

SoS Mentor

Ankit Kumar Misra (Cryptography)

Contents

1 Introduction

1.1 Why do we need Cryptography?

In the modern world, everything is running on lots of data and the internet like texting someone on the internet, accessing websites on the internet, and downloading resources on the internet. If an attacker can modify this data and know the data, then this becomes a great threat to our security. Assume we want to make an online transaction and if the attacker can modify the data of the transaction he may divert it into some other bank account, maybe steal our bank account details and our money. Assume you have a disk or a DVD and you want the content in it to be protected,so that no one else other than the owner can see it.

If you just give a normal disk in which the files are present, anyone with the hard disk can see it. Assume you are requesting some data on the server and the server provides only data to some authorized people, how does the server know that you are the right person and not some attacker who is trying to steal data? Assume you are chatting with someone on the internet and if some attacker eavesdrops on your chats then it is a great threat to our privacy. To provide a solution to all these problems, Cryptography is one of the most widely used efficient techniques.

1.2 Applications of Cryptography

Cryptography is used in almost everything that is related to the internet where there is a need of security , data confidentiality or integrity . It's major applications are in secure communication over internet including web traffic(HTTPS) and wireless traffic like GSM , Bluetooth.. , Encryption of files on any Data Storage devices (Hard drives(EFS) , DVDs(CSS) , Blu-Ray(AACS) , cloud storages) and in user authentication over web (SSL,TLS) and a lot more.

1.2.1 Anonymous communication

Consider a user who wants to access the internet but he doesn't want to reveal his identity to the internet . This problem can be solved by **Mix Networks** . These Networks use proxy relaying servers and encryption to ensure anonymity of senders and receivers .Another solution for this is **Tor Network** which encrypts data packets in layers and uses Tor Nodes to hide the routing process before the packets reach the public internet.

1.2.2 Anonymous digital cash

We can generally buy books when we have a dollar without the merchant not knowing the customer. Now consider this case in digital cash too , Can we create a digital dollar coin and can we make sure that when a dollar coin is spent when the merchant should have no idea who has taken the book . Now one problem over here is , there is no way to stop the customer if the customer tries to create copies of the digital dollar coins that we created . If the customer can create copies of coins , then he can doublespend it resulting in a lot of chaos. So this focuses on __ aspect of cryptography . One solution to this could be if digital cash is spent only once , her identity is not known to anyone but if she uses it more than once all of a sudden her identity will be exposed online.

1.2.3 Election / Private auction (SMPC problem)

Assume elections are being held in a city . The goal now is the voters would like to compute the majority of the votes , but do it in such a way that nothing except the majority of the votes is revealed . A similar can also be seen in a Private Auction . Suppose every bidder has his own bid and assume we are conducting a Vickrey Auction(This is a type of auction where the highest bid wins but pays the second highest bid). Now our goal is to not reveal any of the bids but just reveal who is the final bid winner and what is the amount of the second highest bid? . Our general intuitive solution would be to introduce a trusted authority and make it collect the inputs of the individuals and it only reveals the things that we want to make them public . But , Now the problem arises when the trusted authority / association that we introduce isn't trustworthy . A basic rule of cryptography is that **Any function that you could do with a trusted authority can also be done without a trusted authority** . A lot of research is still going on here. The solutions that are available till now are mainly based on cryptographic primitives like secret sharing , zero-knowledge proofs etc..

1.3 History of Cryptography

Cryptography is derived from ancient greek where "kryptos" means "hidden / secret" and "graphein" means "to write / study" . So the literal meaning of cryptography is to study hidden/secret text and understand their meaning. Cryptography started its origin about 4000 years ago especially when kings used to send their secret messages to his army.Even in the much recent times , America broke some of the ciphers of Germany(Enigma machines) and gained advantage over Germany in the second world war .

1.4 Some basic rules in Cryptography

A cryptography mechanism is defined by mainly two algorithms and the inputs and outputs involved in it. Ciphertext is the encrypted text , generally after encryption the attacker may able to see ciphertext but not the original plaintext which provides confidentiality for your message.

- Encryption Algorithm [*takes a key and message as an input and converts it into ciphertext*]

- Decryption Algorithm [*takes ciphertext and key as input and converts it into the message*]
- (K,M,C) `(Key Space,Message Space,Cipher Space)`

The secret key is the one that we should never reveal to anyone since the Encryption Algorithm is publicly known . So if the attacker is able to see a cipher then he can decrypt the cipher by knowing the secret key giving him abilities to read the plaintext of that ciphertext. Now there are two types of keys that we use in cryptography.

- One time Key [*Single use key i.e. One key is used to encrypt only one message*]
- Many time key [*Multi-use key i.e. same key is used to encrypt many files*]

Knowing all these , we can divide the process of implementing into two parts .

1. Secret Key Establishment [*Creating and implementing a shared key*]
2. Secure Communication
   - Encryption Keys [*provide confidentiality and integrity*]
   - Digital Signatures

Ofcourse Cryptography is a tremendous tool to provide security and is also a basis for many security mechanisms . But Since there are a lot of attacks that can be posed on cryptography mechanisms , We need to be careful while implementing a cryptographic mechanism and using it properly.

1.5 Types of Encryption

Broadly We can classify the types of encryption into two.

1.5.1 Symmetric Encryption

- Uses a single key for both encryption and decryption
- Main drawback is that both parties need to possess the same key to decrypt the data

1.5.2 Asymmetric Encryption

- Uses two different keys : a public key for encryption and a private key for encryption
- Provides enhanced security and flexibility
- Commonly used for secure communication and digital signatures

Whatever we are going to discuss from now on belongs to Symmetric Encryption

2 Classical Ciphers

A classical cipher is an encryption method from past that is no longer used in modern cryptography. All of these classical ciphers are badly broken and some of them created a very large impact like a nation's defeat in wars.

2.1 Substitution Cipher

2.1.1 Construction

This cipher takes input as English letters and outputs english letters . The encryption algorithm is like this you choose every alphabet in english and map it to some other letter and the mapping being one-one (creation of a key - table). Decryption algorithm would just be take the ciphertext lookup for each character in the key-table and map it to the root element in the key-table resulting in plaintext .

2.1.2 Breaking of Substitution Cipher

You can break the substitution cipher by an attack called word frequency , most classical ciphers fall for this attack . In English , we have enough letters to be repeated and the words are not formed any random but they have some specific structure and pattern. For example , the most repeated letter in english is ”e” (it is used 12.7% of time in standard english text) and then we have ”t”(9.1%) , ”a”(8.1%) and so on. We also have digrams(pairs of letters) some of the most common pair of letters(”he”,”an”,”in”,”th”). So, in your cipher the most common letter you found is more likely to bea an ”e” and the second most common letter is most likely going to be a ”t”.. You can deduce some of the cipher with these by guessing some letters and resulting in breaking of the cipher.

2.2 Caesar Cipher

2.2.1 Construction

This can be viewed as a special case of Substitution Cipher. In Substitution cipher we assign a letter to some other random letter irrespective of its position , But in the caesar cipher , we make letters shift by some fixed number (¡26) . Suppose if we want to encrypt the word ”hello” if we decide the number of the letters to shift it with is 13 , then it becomes ”uryyb”. The decryption is something similar to it , but instead of going forward in the alphabetical order we go in the reverse alphabetical order to find the plaintext by the ecided shift of letters.

2.2.2 Breaking of Caesar Cipher

This also falls for the word frequency attack . But this has a much simpler exhaustive attack , you check for each number of shift and until you find something meaningful you try to decrypt it again by changing the shift . The total number of shifts that could be possible are 25 (if you shift by 26 letters , essentially you would get the same message) which isn't large.

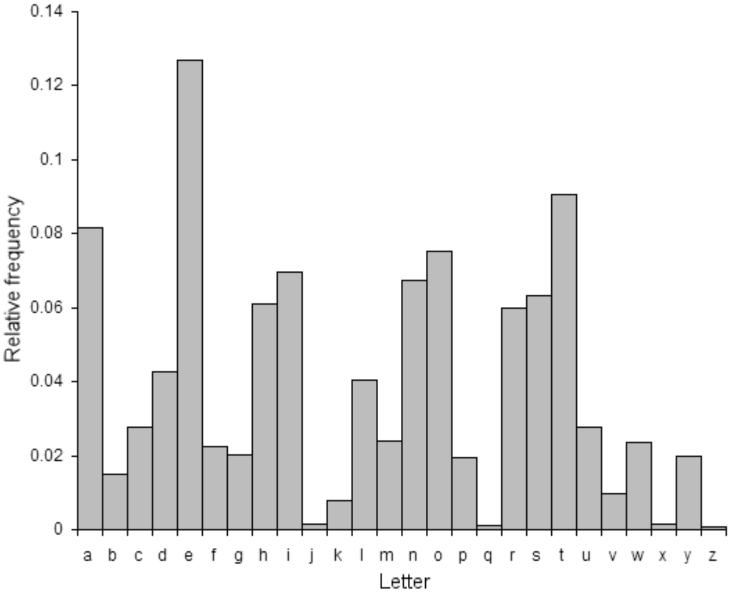
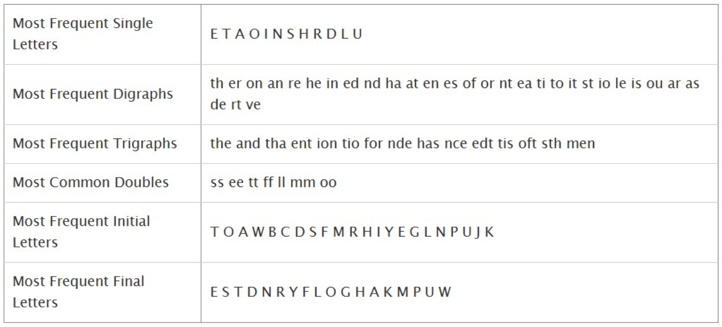


Figure 1: Letter frequency in English



Most Frequent Single Letters	E T A O I N S H R D L U
Most Frequent Digraphs	th er on an re he in ed nd ha at en es of or nt ea ti to it st io le is ou ar as de rt ve
Most Frequent Trigraphs	the and tha ent ion tio for nde has nce edt tis oft sth men
Most Common Doubles	ss ee tt ff ll mm oo
Most Frequent Initial Letters	T O A W B C D S F M R H I Y E G L N P U J K
Most Frequent Final Letters	E S T D N R Y F L O G H A K M P U W

Figure 2: Most Common groups of letters in English

2.3 Vigener Cipher

2.3.1 Construction

This cipher was used by Romans. Again the key space , message space and cipher space all belong to 26 english letters. To encrypt something with this mechanism , first you choose a key and if the key's length is message's length you will repeat it until you make key's length equal to message's length . Now we will choose each character of message and key one by one add them according to the lexicographic numbers and if it exceeds 26 we will do the `sum %(modulo) 26` and write the character at the obtained number's place in our ciphertext . Similarly to decrypt instead of addition you subtract the letters and find modulo 26 to obtain the plaintext.

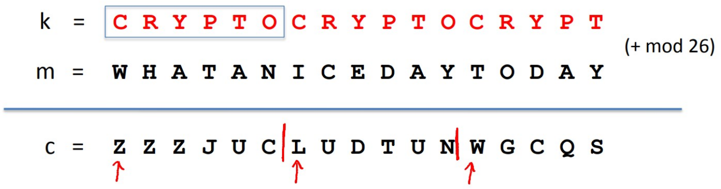

k = C R Y P T O C R Y P T O C R Y P T
(+ mod 26)
m = W H A T A N I C E D A Y T O D A Y
c = Z Z Z J U C L U D T U N W G C Q S

Figure 3: Encryption of Vigener Cipher

2.3.2 Breaking a Vigener

This was able to add some additional security compared to substitution cipher but still it fell for the word frequency attack . Assume you know the length of the key (`l`). Now the first letter in the ciphertext and the letter after `l` letters and the letter after another `l` letters and so on are decrypted using the same key . So if you observe that group of letters the most frequent letter should most likely be an "e" and similarly you perform the attack revealing the plaintext.

2.4 Rotor Machines

The idea of this rotor machines is we are using same key-table for every character , if we are able to use different key-table for each character's encoding then we may have better security. So there is a rotor machine in which our key-table(similar to substitution cipher) is present then you enter each character of your message , with every character of message the substitution cipher table will rotate by 1 letter resulting in a new-key table.

For more security , German improved it by placing more rotors which have around 3-5 rotors and this machine is called Enigma machine .

But all of these fell to the Word frequency

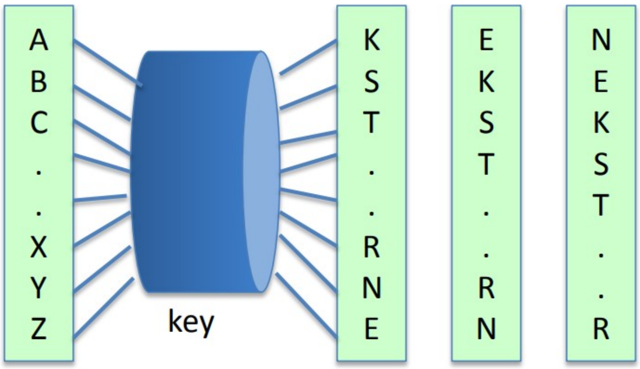


Figure 4: Rotor Machine

2.5 DES(Data Encryption Standard)

2.5.1 Construction

DES uses a key-space of length 2^{56} and it has a block size of 64 bits [8 bytes] . It is discussed in much detail about block ciphers.

2.5.2 Attacks on DES

These days it can be easily broken so it is not recommended using in the upcoming projects. Instead the developments of these AES(2001) , Salsa20(2008) .

3 Pseudo Randomness

3.1 How is Pseudo Randomness used in encryption?

4 Discrete Probability Theory

4.1 Probability Distribution

Let U be a finite set , Probability Distribution P over U is a function

$$P : U \to [0, 1] \mid \sum_{x \in U} P(x) = 1$$

Examples:

1. Uniform distribution: $\forall x \in U : P(x) = \frac{1}{|U|}$
2. Point distribution at x_0: $P(x_0) = 1$, $\forall x \neq x_0 : P(x) = 0$

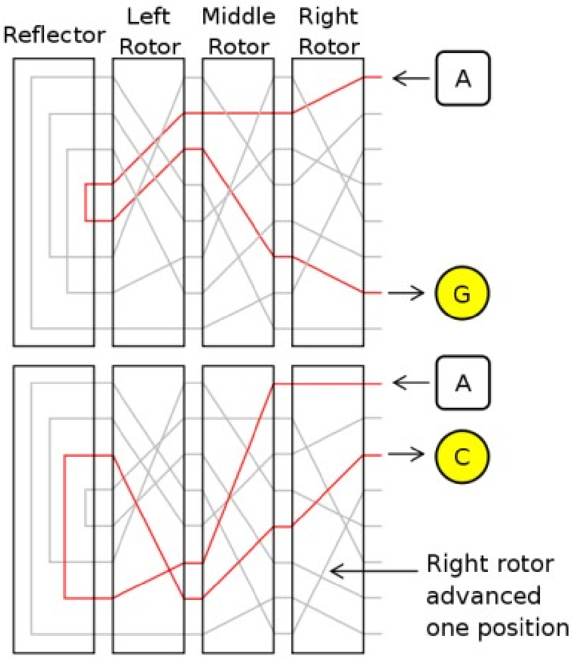


Figure 5: Enigma Machine

We can also define a **Probability Distribution Vector** called which represents the value of each probability of each value for better readability . For example if we want to create a probability distribution vector for 2-bit strings which will look something like this

$$(P(00), P(01), P(10), P(11)) \in R^8$$

4.2 Event

For a set A $\subseteq U$:

$$Pr[A] = \sum_{x \in A} P(x) \in [0, 1]$$

Here the set A is called an event . Note : Pr[U] = 1
Example: U = $\{0, 1\}^8$

- $A = \{\text{all } x \in U \text{ such that } lsb_2(x) = 11\} \subseteq U$ [lsb_2 meaning the least 2 significant bits] For the uniform distribution on U: $\Pr[A] = \frac{1}{4}$

Events follow the **Union Bound** property :

- For events A_1 and A_2
  $\Pr[A_1 \cup A_2] = \Pr[A_1] + \Pr[A_2]$ - $\Pr[A_1 \cap A_2]$

4.2.1 Independent Events

If two events A and B are said to be independent if Pr[A & B] = Pr[A] · Pr[B] .

4.3 Random Variables

A random variable X is a function $X : U \to V$.
Example: $X : \{0,1\}^n \to \{0,1\}$; $X(y) = \text{lsb}(y) \in \{0,1\}$.

For the uniform distribution on U:

$$\Pr[X = 0] = \frac{1}{2}, \quad \Pr[X = 1] = \frac{1}{2}$$

More generally, A random variable X induces a distribution on V implies

$$\Pr[X = v] = \Pr[X^{-1}(v)]$$

Uniform Random Variable.
Let U be some set, e.g. $U = \{0,1\}^n$.
We write $r \leftarrow U$ to denote a uniform random variable over U.
For all $a \in U$:

$$\Pr[r = a] = \frac{1}{|U|}$$

(Formally, r is the identity function: $r(x) = x$ for all $x \in U$.)

4.4 Randomized algorithms

To know about Randomized algorithms , we need to know about Deterministic Algorithms first . Deterministic algorthims are predictable and repeatable. They always output the same thing every time you give the same input . Randomised algorithms on the other side take in an element of randomness for execution resulting in different output for the same input . Theoretically ,

$$y \leftarrow A(m\ ;\ r) \quad \text{where} \quad r \xleftarrow{R} \{0,1\}^n$$

The output is a random variable.

4.5 XOR (Exclusive OR)

XOR of two strings in $\{0,1\}^n$ is defined as their bit-wise addition mod 2.

4.5.1 An important property of XOR:

Theorem: Let Y be a random variable over $\{0,1\}^n$, and X an independent uniform random variable on $\{0,1\}^n$. Then $Z := Y \oplus X$ is a uniform random variable on $\{0,1\}^n$.

Proof: (for $n = 1$)
Consider Y as a random variable over $\{0,1\}$ and X as an independent uniform random variable on $\{0,1\}$. We need to show that $Z = Y \oplus X$ is uniformly distributed over $\{0,1\}$.

Since X is uniform, we have:

$$\Pr[X = 0] = \Pr[X = 1] = \frac{1}{2}$$

Let $a \in \{0,1\}$. We need to show that $\Pr[Z = a] = \frac{1}{2}$. There are two cases to consider for Z:

1. **Case** $a = 0$:

$$\Pr[Z = 0] = \Pr[Y \oplus X = 0] = \Pr[Y = X]$$

Since Y and X are independent and X is uniform:

$$\Pr[Y = X] = \Pr[Y = 0, X = 0] + \Pr[Y = 1, X = 1]$$

Because Y is a random variable over $\{0,1\}$ and X is uniform:

$$\Pr[Y = 0, X = 0] = \Pr[Y = 0] \cdot \Pr[X = 0] = \Pr[Y = 0] \cdot \frac{1}{2}$$

$$\Pr[Y = 1, X = 1] = \Pr[Y = 1] \cdot \Pr[X = 1] = \Pr[Y = 1] \cdot \frac{1}{2}$$

Therefore:

$$\Pr[Y = X] = \Pr[Y = 0] \cdot \frac{1}{2} + \Pr[Y = 1] \cdot \frac{1}{2} = \frac{1}{2}(\Pr[Y = 0] + \Pr[Y = 1]) = \frac{1}{2}$$

2. **Case** $a = 1$:

$$\Pr[Z = 1] = \Pr[Y \oplus X = 1] = \Pr[Y \neq X]$$

Since Y and X are independent and X is uniform:

$$\Pr[Y \neq X] = \Pr[Y = 0, X = 1] + \Pr[Y = 1, X = 0]$$

Because Y is a random variable over $\{0,1\}$ and X is uniform:

$$\Pr[Y = 0, X = 1] = \Pr[Y = 0] \cdot \Pr[X = 1] = \Pr[Y = 0] \cdot \frac{1}{2}$$

$$\Pr[Y = 1, X = 0] = \Pr[Y = 1] \cdot \Pr[X = 0] = \Pr[Y = 1] \cdot \frac{1}{2}$$

Therefore:

$$\Pr[Y \neq X] = \Pr[Y = 0] \cdot \frac{1}{2} + \Pr[Y = 1] \cdot \frac{1}{2} = \frac{1}{2}(\Pr[Y = 0] + \Pr[Y = 1]) = \frac{1}{2}$$

Thus, $\Pr[Z = 0] = \Pr[Z = 1] = \frac{1}{2}$, meaning Z is uniformly distributed over $\{0,1\}$. This completes the proof for $n = 1$.Similarly we can extend this to other values of n.

4.5.2 Birthday Paradox

Let $r_1, \ldots, r_n \in U$ be independent identically distributed random variables.
Theorem: When $n = 1.2 \times |U|^{1/2}$, then $\Pr[\exists \text{ i} \neq j : r_i = r_j] \geq \frac{1}{2}$.
Example: Let $U = \{0,1\}^{128}$.

After sampling about 2^{64} random messages from U,
some two sampled messages will likely be the same. when the number of elements that we sample are about $|U|^{\frac{1}{2}}$ then it is more likely that two persons will have same value . but the important condition is that $r_1, r_2, r_3, r_4, r_5 \ldots$. $\in$ U be independent identically distributed random variables.

4.6 Statistical Test

A statistical test on $\{0,1\}^n$ is defined as an algorithm A such that A(x) outputs ”0” or ”1”.

4.7 Advantage

Let G:$K \leftarrow \{0,1\}^n$ be a PRG and A be a statistical test on $\{0,1\}^n$. We define advantage to be

$$\text{Adv}_{\text{PRG}}[A, G] := \left| \Pr_{K \xleftarrow{R} K}[A(G(K)) = 1] - \Pr_{r \xleftarrow{R} \{0,1\}^n}[A(r) = 1] \right| \in [0,1] \tag{1}$$

- Adv close to 1 $\Rightarrow$ A can distinguish G from random
- Adv close to 0 $\Rightarrow$ A cannot distinguish G from random

Essentially advantage gives an idea of how close to random functions it is .

5 One Time Pad

5.0.1 Construction

This is a secure cipher invented by Vernam in 1917 . The message space(M) and cipher space(C) is $\{0,1\}^n$ where $n \in Z$ and the key space(K) also belongs to $\{0,1\}^n$. The key is a random string as long as message to be encrypted.

$$C = E(K, M) = K \oplus M$$

$$D(K, C) = K \oplus C$$

Any cipher that is treated to be consistent must satisfy the consistency equation.

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

We can prove this for the One-Time Pad since $D(k, E(k, m)) = k \oplus k \oplus m = m$ $[\because x \oplus x = 0 \,\text{and}\, 0 \oplus k = k]$
You can also obtain the key if you have the plaintext and ciphertext by using the relation $k = m \oplus c$

5.0.2 Problem with One Time Pad

The problem in One Time Pad is that the keys are as long as messages . Imagine if I want to send a file which has around 1GB of data in it . Now if I want to encrypt it with One Time Pad then I and the receiver must share a secret key confidentially which is about 1GB and if there is some mechanism to share that 1GB key securely , I would rather use that mechanism to send my 1GB file to the receiver instead of One Time Pad . But other than that One Time Pad has a lot of advantages and security features in it.

5.0.3 Advantages with One Time Pad

One time pad is very fast as it just involves only XOR operations but the only caveat is that it generates long ciphertext. To explain the security of this cipher we need to explain what does it mean for a cipher to be secure. Claude Shannon [Father of Information Theory] proposes his idea that **CT(Ciphertext) should reveal no info about PT(Plaintext)**

INFORMATION THEORETIC SECURITY - SHANNON

Perfect Secrecy
A Cipher (**E,D**) over ($\mathcal{M}$,$\mathcal{K}$,$\mathcal{C}$) has perfect secrecy if $\forall m_0, m_1 \in \mathcal{M}$ ($|m_0| = |m_1|$) and ∀ c ∈ $\mathcal{C}$

$$Pr[E(k, m_0) = c] = Pr[E(k, m_1) = c] \text{ where } k \xleftarrow{R} K$$

So if some cipher has perfect secrecy than we can't tell which message it has been encrypted from by seeing the cipher because all messages are equally likely to be encrypted. So there is no Ciphertext only attack (Deducing plaintext by just knowing one cipher).

5.0.4 One Time Pad's perfect secrecy

Theorem: The One-Time Pad (OTP) encryption scheme has perfect secrecy.

Proof: Consider the OTP encryption process:

$$C = E(K, M) = K \oplus M$$

where K is a random key of the same length as M, and $\oplus$ denotes bitwise XOR.

To prove perfect secrecy, we need to show that:

$$\Pr[E(K, m_0) = c] = \Pr[E(K, m_1) = c] \text{ for all } m_0, m_1 \in \mathcal{M} \text{ and } c \in \mathcal{C}$$

Since $C = K \oplus M$, for any given $M = m$ and ciphertext $C = c$, there exists a unique key $K = c \oplus m$. Given that the key K is uniformly chosen from the key space, we have:

$$\Pr[K = k] = \frac{1}{|\mathcal{K}|} \text{ for all } k \in \mathcal{K}$$

Therefore, for any $m_0, m_1 \in \mathcal{M}$,

$$\Pr[E(K, m_0) = c] = \Pr[K = c \oplus m_0] = \frac{1}{|\mathcal{K}|}$$

and similarly,

$$\Pr[E(K, m_1) = c] = \Pr[K = c \oplus m_1] = \frac{1}{|\mathcal{K}|}$$

Since these probabilities are equal, we have:

$$\Pr[E(K, m_0) = c] = \Pr[E(K, m_1) = c] \text{ for all } m_0, m_1 \in \mathcal{M} \text{ and } c \in \mathcal{C}$$

Thus, the One-Time Pad encryption scheme has perfect secrecy.

5.1 Shannon's Theorem : Key Size Requirement

Theorem: If a cipher has perfect secrecy, then the size of the key space $|\mathcal{K}|$ must be at least the size of the message space $|\mathcal{M}|$.

Proof by Contradiction: Assume for contradiction that $|\mathcal{K}| < |\mathcal{M}|$.

Since $|\mathcal{K}|$ is the number of possible keys and $|\mathcal{M}|$ is the number of possible messages, each key $k \in \mathcal{K}$ can only encrypt to at most $|\mathcal{C}|$ ciphertexts. Given perfect secrecy, each plaintext $m \in \mathcal{M}$ must be equally likely to produce any ciphertext $c \in \mathcal{C}$. Therefore, the total number of unique ciphertexts must be at least the number of plaintexts, implying that $|\mathcal{C}| \geq |\mathcal{M}|$.

However, if $|\mathcal{K}| < |\mathcal{M}|$, there are not enough unique keys to cover all possible messages with perfect secrecy. Specifically, there would be some messages $m_0, m_1 \in \mathcal{M}$ that map to the same ciphertext c for a given key k.

This would violate the condition for perfect secrecy:

$$\Pr[E(K, m_0) = c] \neq \Pr[E(K, m_1) = c]$$

Thus, our assumption that $|\mathcal{K}| < |\mathcal{M}|$ must be false. Therefore,

$$|\mathcal{K}| \geq |\mathcal{M}|$$

Since the One-Time Pad uses a key that is the same length as the message, it achieves perfect secrecy in the most optimal manner, making it the optimal cipher for perfect secrecy.

6 Stream Ciphers

6.1 Pseudo Random Generator(PRGs)

Pseudo Random Generator is a function $G : \{0, 1\}^s \to \{0, 1\}^n \quad n >> s$
This function is efficiently calculated by a deterministic algorithm. This takes in a small input called seed and generates a large string which we can use for

key. But through Shannon's theorem regarding key-length we can't attain perfect security for this . So stream ciphers avoid the problem of long keys , but they do not have perfect secrecy as defined by Shannon .One important thing is **PRGs must be unpredictable** . First let us see what does it mean for a PRG to be predictable. Generally speaking if we could even predict one bit of PRG then it is predictable . Because it would eventually lead to the entire output being predicted.

Claim: If any bit of the PRG output can be predicted, then the entire output can be predicted.

Proof:

Assume for contradiction that there exists a PRG whose output is predictable, meaning that there is some efficient algorithm A that can predict at least one bit of the PRG output with non-negligible probability. Specifically, let $G(s)$ be the output of the PRG when given the seed s. Suppose A can predict the i-th bit of $G(s)$, denoted $G(s)_i$.

If A can predict $G(s)_i$ with non-negligible probability, then we can use A to predict the next bits of the PRG output. This is because PRGs are deterministic, meaning that if we know part of the output and the internal state, we can compute the next part of the output.

Formally, let $G(s) = b_1 b_2 \dots b_n$ be the bit string output by the PRG, where each b_j is a bit. Suppose A can predict b_i given the previous bits $b_1, b_2, \dots, b_{i-1}$. If A can predict b_i, then we can construct a new algorithm A' that predicts b_{i+1}, and so on, until the entire output $G(s)$ is predicted.

This leads to a contradiction because if the entire output $G(s)$ can be predicted, then $G(s)$ is not indistinguishable from a truly random string. Therefore, no efficient algorithm A should be able to predict any bit of the PRG output with non-negligible probability.

Thus, a PRG must be unpredictable to ensure that its output remains indistinguishable from a truly random string, preserving its security.

Definition: There exists an "efficient" algorithm A and $1 \leq i \leq n-1$ such that

$$\Pr[A(G(k))|_{1,2,\dots,i} = G(k)|_{i+1}] > \frac{1}{2} + \epsilon$$

where ϵ is non-negligible (e.g., $\epsilon \geq \frac{1}{2^{30}}$). For our analysis , we can consider ϵ to be negligible if $\epsilon \leq \frac{1}{2^{80}}$

In theory, ϵ is a function $\epsilon : \mathbb{Z}_{\geq 0} \to \mathbb{R}_{\geq 0}$ and

- ϵ is non-negative: $\exists d$ such that $\epsilon(\lambda) \geq \frac{1}{\lambda^d}$ infinitely often (i.e., $\epsilon \geq \frac{1}{\text{poly}}$, for many λ)
- ϵ is negligible: $\forall d, \lambda \geq \lambda_d$: $\epsilon(\lambda) \leq \frac{1}{\lambda^d}$ (i.e., $\epsilon \leq \frac{1}{\text{poly}}$, for large λ)

6.1.1 Some weak PRGs

Linear Congruential Generator takes parameters a,b,p where r[0] is assumed to be seed . r[i] = (a*r[i-1] + b) modulo p . It outputs few bits of r[i] and increments i. Likewise required number of bits are taken to form a key.

glibc random() is part of the GNU C Library (glibc) and is used to generate pseudo-random numbers. This function is commonly used in C programs and provides a sequence of numbers that appear to be random. However, it is important to understand both its functionality and its limitations, particularly its weaknesses as a pseudo-random generator (PRG).The `glibc()` random function uses a combination of a linear congruential generator (LCG) and a secondary algorithm to produce pseudo-random numbers. The specific algorithm used by `glibc()` is as follows:

$$r[i] \leftarrow (r[i-3] + r[i-31]) \mod 2^{32}$$

$$\text{output } r[i] \gg 1$$

Here, r is an array of state values, and the function combines values from two different points in the state array to generate the next value. The result is then right-shifted by one bit to produce the output.

- **Predictability**: Since `glibc()` random is a deterministic algorithm, if an attacker knows the seed or can deduce it, they can predict the entire sequence of generated numbers.
- **Short Period**: The period of the generator is relatively short compared to more robust PRGs, meaning that the sequence of numbers will eventually repeat.
- **Linear Congruential Generator**: The core of the `glibc()` random function is an LCG, which is known to have poor randomness properties and can be easily broken if we have enough output information.

So it is advised not to use any of these in the upcoming cryptographic protocols.

6.2 Two time pad attack on Stream Ciphers

If two messages are encrypted using the same key and if the hacker can see both of the ciphers which are encrypted using the same key then the stream cipher becomes insecure and the plaintext is revealed . Assume

$$m_1 \oplus PRG(k) \rightarrow C_1$$

$$m_2 \oplus PRG(k) \rightarrow C_2$$

If we observe the value of $C_1 \oplus C_2$ then it becomes $m_1 \oplus m_2$. and if we have enough redundancy in English and ASCII encoding that if we know $m_1 \oplus m_2$ then we can decode m_1 & m_2 . So the key learning from this is that you should never use the one time pad key more than once .

- **Project Venona** (1941-1946): This was a project by the US intelligence to read the plaintexts from the ciphertexts of Russia during World War II. Russia was using a stream cipher, and they initially generated the keys manually by throwing dice, thinking it would create randomness. However, finding this method laborious, they began reusing the same keys. This reuse led to the vulnerability exploited by the US intelligence, known as the two-time pad attack. As a result, US intelligence was able to read about 3000 plaintexts and understand the majority of their messages.
- **MS-PPTP** (Microsoft Point-to-Point Tunneling protocol) is a protocol for users accessing a website . The user would request for something , so a request is sent from user to server (which is encrypted using one time pad) and server sends back the data that the user requested . The user and the server will have a shared secret key everytime with which the messages are encrypted . Now if some attacker can focus on one certain user and observe the ciphertexts from him and he could essentially do a two time pad attack because every message that is being sent is encrypted using the same key.
- **802.11b WEP**(Wired Equivalent Privacy) is used between a client and an access point . It uses a Pseudo-Random Generator (PRG) with IV concatenated with key (IV —— key).The IV is 24 bits long. The key changes after every frame, with repeated IV after 2^{24} (approximately 16 million frames).This repetition of the pad essentially creates a scenario similar to a two-time pad attack. Another mistake in WEP is that the key was not randomized properly, such as using $0||k$ and $1||k$. The PRG key is 104 bits (RC4), and listening to 106 frames can help recover the secret key (40,000 frames are also efficient). On some 802.11 cards, IV resets to 0 after a power cycle leading to encryption with repeated keys.We can make WEP better by joining messages to reduce the number of frames , use a PRG to generate a large Pseudo random key and can divide this large key into multiple parts where each frame uses a different pseudorandom key.

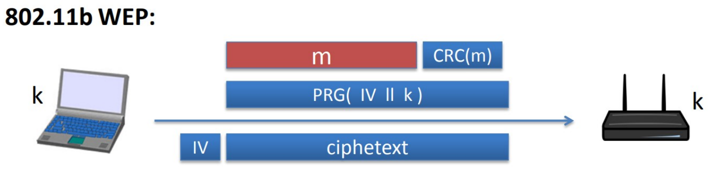


Figure 6: Working of WEP

Disk Encryption - One Time pad

One of the major issues with using stream ciphers for disk encryption is that if a single bit or byte in the encrypted data is altered, it can be immediately identified and located.An attacker can modify the encrypted data and observe the changes in the decrypted data, potentially leading to the discovery of patterns or even the decryption key.Stream ciphers do not provide built-in mechanisms to ensure the integrity of the data. Any modification, intentional or accidental, can corrupt the data without being detected until it is too late. So the key takeaway here is that One Time Pad has no integrity meaning **OTP is malleable**. Modifications of ciphertext are undetected and have predictable impact on plaintext.

6.2.1 RC4

RC4, designed in 1987, is a widely-used stream cipher known for its simplicity and speed. The RC4 algorithm has a variable size key, typically ranging from 40 to 256 bits, and operates by generating a pseudorandom stream of bits (keystream) which is XORed with the plaintext bits to produce ciphertext.

Functionality

- **Seed:** RC4 starts with a key scheduling algorithm (KSA) that initializes a permutation of all 256 possible byte values (a state array, S) using a variable-sized key.
- **Keystream Generation:** The state array is then used to generate a pseudorandom stream of bits which form the keystream. For each byte of plaintext, one byte of the keystream is XORed with it to produce the ciphertext byte.
- **Bytes per Round:** Each round of the algorithm processes one byte of the plaintext.

Usage

RC4 has been used in various protocols and standards, most notably in HTTPS and WEP (Wired Equivalent Privacy).

Weaknesses

Despite its popularity, RC4 has several weaknesses:

1. **Bias in Initial Output:** The initial bytes of the keystream generated by RC4 are not uniformly random. For instance, the probability that the second byte of the keystream is zero is given by:

   $$\Pr[\text{2nd byte} = 0] = \frac{2}{256}$$

   This means that the second byte is more likely to be zero than any other value.

2. **Probability of (0,0):** The probability of encountering a pair of zeroes in the initial keystream is higher than expected:

$$\Pr[(0,0)] = \frac{1}{256^2} + \frac{1}{256^3}$$

3. **Related Key Attacks:** RC4 is susceptible to related key attacks, where the attacker can deduce information about the plaintext or the key by analyzing patterns in the ciphertexts encrypted with related keys.

4. **Bias in First 256 Bytes:** The first 256 bytes of the RC4 keystream are biased and thus generally advised to be discarded to improve security.

Improvements

To mitigate these weaknesses, it is often recommended to discard the initial 256 bytes of the keystream. This practice is known as RC4-dropN, where N indicates the number of initial keystream bytes to discard.

6.2.2 CSS (Content Scrambling System)

CSS uses Linear Feedback Shift Registers which are more compatible to hardware elements . A LFSR is a shift register where some positions, known as taps, are XORed together to generate a new bit. The first bit becomes the XOR of all these taps, and the last bit is shifted out. The seed represents the initial state of the LFSR.

- **DVD Encryption (CSS)**: Utilizes 2 LFSRs.
- **GSM Encryption (A5/1, A5/2)**: Utilizes 3 LFSRs.
- **Bluetooth Encryption (E0)**: Utilizes 4 LFSRs.

It was originally designed for DVD encryption and uses a seed of 5 bytes (40 bits) . During then , the government laid some rules on cryptographic protocols that are to be exported (since a lot of DVDs were exported) and the developers of CSS should limit the seed of CSS to a very small seed space . CSS essentially uses two LFSRs (17-bit LFSR & 25-bit LFSR)

Attack on CSS

DVDs typically use MPEG files, which means a small prefix of the file is often known in advance, usually around 20 bits. By knowing this prefix, we can try all 2^{17} possible initial values of the 17-bit LFSR to match the first 20 bits of the output.This approach significantly reduces the complexity of breaking the encryption.

6.2.3 eStream : Salsa20

To avoid the problem of repeated keys , we introduce a new element called Nonce. A nonce is a non-repeating value for a given key. It ensures that the same key can be safely reused for multiple encryptions. So if we assume the key to be k and r , the pair (k, r) should never be reused to maintain security implying that nonce space should be large enough so that while choosing a random value you should never choose the same nonce again. Salsa20 is a specific eStream cipher designed for efficiency in both software and hardware implementations.

$$\text{Salsa20} : \{0,1\}^{128 \text{ or } 256} \times \{0,1\}^{64} \to \{0,1\}^{n}$$

Where:

- Key size: 128 or 256 bits
- Nonce size: 64 bits
- Maximum output size: $n = 2^{73}$ bits

The core of Salsa20 is defined as:

$$\text{Salsa20}(k; r) := H(k, (r, 0)) \| H(k, (r, 1)) \| \ldots$$

Where H is an invertible function designed to be fast on x86 architectures using SSE2 instructions.

6.2.4 Performance Analysis of Stream Ciphers

The following table presents performance data for various stream ciphers, measured on an AMD Opteron processor running at 2.2 GHz under Linux. The data is from Crypto++ 5.6.0, as reported by Wei Dai.

PRG	Speed (MB/sec)
RC4	126
Salsa20/12	643
Sosemanuk	727

Table 1: Performance comparison of stream ciphers

6.3 Secure PRGs

We say that G:K $\leftarrow \{0,1\}^n$ is a secure PRG if $\forall$ efficient statistical tests A $\text{Adv}_{\text{PRG}}[A, G]$ is negligible. Generally speaking this means PRG is unpredictable.

Proof: If a PRG is Predictable, then it is Insecure

We will show that if a PRG is predictable, there exists a statistical test with a non-negligible advantage, thereby proving the insecurity of the PRG.
Define a statistical test B as follows:

$$B(X) = \begin{cases} 1 & \text{if } A(X_1, \ldots, X_i) = X_{i+1} \\ 0 & \text{otherwise} \end{cases}$$

Where A is an algorithm that tries to predict the next bit of the PRG output.

Let $r \xleftarrow{R} \{0,1\}^n$ be a truly random string, and let $k \xleftarrow{R} K$ be a random seed for the PRG.

$$\Pr[B(r) = 1] = \frac{1}{2}$$

$$\Pr[B(G(k)) = 1] = \frac{1}{2} + \epsilon$$

where ϵ is non-negligible.

From this, we derive that the advantage of the PRG against the statistical test B is greater than ϵ:

$$\text{Adv}_{\text{PRG}}[B, G] > \epsilon$$

Yao proved the theorem **an unpredictable PRG is secure** which is the converse of the theorem. It states if ∀ i ∈ 0,1,...,n-1 PRG is unpredictable at position i then it is a secure PRG . Essentially it says that if next bit-predictors cannot distinguish G from random then no statistical test can.

Computationally Indistinguishable

Let P_1 and P_2 be two distributions over $\{0,1\}^n$.

Definition: We say that P_1 and P_2 are **computationally indistinguishable** (denoted $P_1 \approx_P P_2$) if

$$\forall \text{ efficient statistical tests } A, \left| \Pr_{x \leftarrow P_1}[A(x) = 1] - \Pr_{x \leftarrow P_2}[A(x) = 1] \right| < \text{negligible}$$

A PRG is secure if it is computationally indistinguishable.

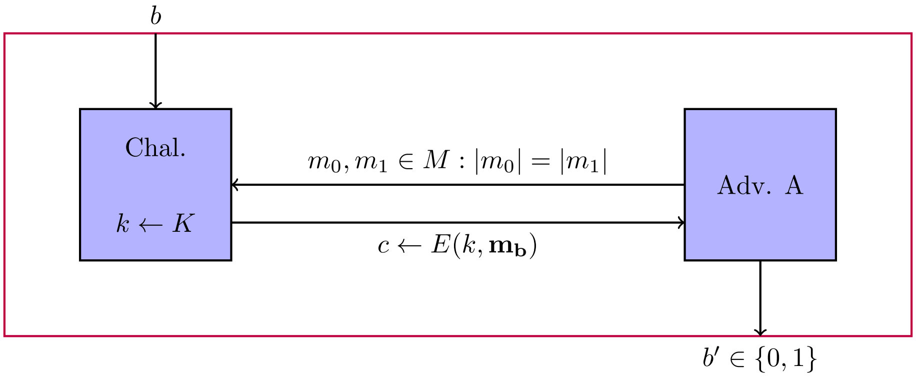
6.4 Semantic Security

Since perfect secrecy isn't possible for a stream cipher according to Shannon we slightly modify the definition of perfect secrecy .
(E, D) has perfect secrecy if $\forall\ m_0,\ m_1 \in M$ ($|m_0| = |m_1|$)

$$\{E(k, m_0)\} \approx_p \{E(k, m_1)\}$$

where $k \leftarrow K$.

Now to test Semantic Security we test our cipher through an experiment. For $b = 0, 1$ define experiments EXP(0) and EXP(1) as:


b
Chal.
$k \leftarrow K$
$m_0, m_1 \in M : |m_0| = |m_1|$
$c \leftarrow E(k, \mathbf{m_b})$
Adv. A
$b' \in \{0,1\}$

for $b = 0, 1$: $W_b :=$ [event that EXP(b) = 1]

$$\text{Adv}_{\text{SS}}[A, E] := |\Pr[W_0] - \Pr[W_1]| \in [0, 1]$$

Here we allow the adversary to send us two messages and we will encrypt them and send him back . The adversary should tell us which message we have sent from .

Semantically secure

Definition : E is **semantically secure** if for all efficient adversaries "A" $Adv_{SS}[A, E]$ is negligible.If advantage is negligible then it essentially means then we can say that the attacker can't distinguish messages.

- OTP is semantically secure infact it is more secure than semantic security i.e. perfect secrecy. Since the ciphers have equal probability of being encrypted from both the messages , the advantage is essentially zero.

6.4.1 Stream ciphers are semantically secure

Theorem: Secure PRG implies a stream cipher derived from the secure PRG is semantically secure.
Proof: Let $G : K \to \{0, 1\}^n$ be a secure pseudorandom generator (PRG). Then, the stream cipher E derived from G is semantically secure. Let us consider a stream cipher E derived from a pseudorandom generator $G : K \longrightarrow \{0, 1\}^n$. We want to prove that if G is secure then E is semantically secure (sem. sec.).

We use an intuitive approach based on indistinguishability as follows:

Assume we have two scenarios:

1. An adversary *Adv. A* chooses between two messages m_0 and m_1. A key k is chosen uniformly at random from key space K. The ciphertext c is then generated by XORing message m_0 with the output of PRG function $G(k)$. The adversary tries to guess which message was encrypted.

2. In an analogous setup, instead of using $G(k)$, a truly random string r from $\{0,1\}^n$ is used to XOR with message m_0, resulting in ciphertext c. The adversary again tries to guess which message was encrypted.

Both halves aim to show that if $G : K \longrightarrow \{0,1\}^n$ is a secure PRG, then an adversary cannot distinguish whether c was created using $G(k)$ or r with probability significantly better than 1/2. This implies that stream cipher E derived from G would be semantically secure because it behaves similarly to one using truly random strings.

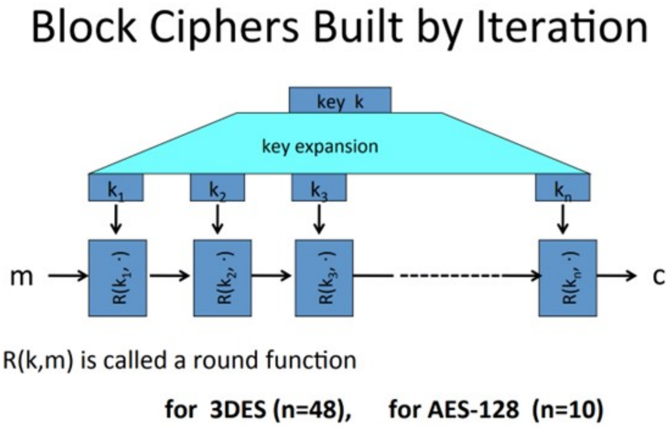
Formally, for every semantically secure adversary A, there exists a PRG adversary B such that:

$$\mathrm{Adv}^{A}_{\mathrm{SS}}(E) \leq 2 \cdot \mathrm{Adv}^{B}_{\mathrm{PRG}}(G)$$

7 Block Ciphers

Block ciphers take in ”n” bits of plaintext block and outputs ”n” bits of ciphertext block with a key of length ”n” . Some of the examples of Block Ciphers are

- **3DES** : n = 64 bits , k = 168 bits
- **AES** : n = 128 bits , k = 128,192,256 bits


Block Ciphers Built by Iteration

R(k,m) is called a round function

for 3DES (n=48), for AES-128 (n=10)

Figure 7: Block Cipher built by Iteration

Block ciphers are built by iteration . Generate a random key and expand the key into ”n” blocks(for 3DES (n=48) , AES-128 (n=10)). The generated key blocks are called round keys . Now pass the message through each of these keys along with a round function R(k_1,·) and the obtained message is passed through the next round function and so on as shown in the image. So to define a block cipher we need to specify two things **Key Expansion Algorithm , Round function**

Pseudo Random Functions & Permutations

Pseudo-Random Function (PRF)
A pseudo-random function (PRF) is a function $F : K \times X \rightarrow Y$ such that:

- **Efficient evaluation:** There exists an efficient algorithm to evaluate $F(k, x)$ for any key $k \in K$ and input $x \in X$.

Pseudo-Random Permutation (PRP)
A pseudo-random permutation (PRP) is a function $E : K \times X \rightarrow X$ such that:

- **Efficient evaluation:** There exists an efficient algorithm to evaluate $E(k, x)$ for any key $k \in K$ and input $x \in X$.
- **Injectivity:** The function $E(k, \cdot)$ is one-to-one for any key $k \in K$. In other words, for any $k \in K$ and distinct inputs $x_1, x_2 \in X$, we have $E(k, x_1) \neq E(k, x_2)$.
- **Efficient inversion:** There exists an efficient algorithm D such that for any key $k \in K$ and output $y \in X$, $D(k, y)$ returns the unique preimage $x \in X$ such that $E(k, x) = y$.

Functionally , any PRP is also a PRF. A PRP is a PRF where X=Y and is efficiently invertible.

Secure Pseudo-Random Function (PRF)
Let $F : K \times X \rightarrow Y$ be a function, where K is the key space, X is the input space, and Y is the output space. We denote by $\text{Funs}[X, Y]$ the set of all functions from X to Y. Let $\mathcal{S}_{\mathcal{F}} = \{F(k, \cdot) \mid k \in K\} \subseteq \text{Funs}[X, Y]$ be the set containing all functions obtainable by evaluating F with a key k from K and leaving the input slot empty.

A PRF F is considered secure if it is indistinguishable from a random function in $\text{Funs}[X, Y]$. Intuitively, this means that an adversary who can only query the function (without knowing the secret key) cannot tell the difference between interacting with the actual PRF F and interacting with a truly random function that maps elements of X to elements of Y.

Secure Pseudo-Random Permutation (PRP)
Let $E : K \times X \rightarrow X$ be a PRP, where K is the key space and X is the input and output space. We denote by $\text{Perms}[X]$ the set of all one-to-one functions from X to itself. Let $\mathcal{S}_{\mathcal{F}} = \{E(k, \cdot) \mid k \in K\} \subseteq \text{Perms}[X]$ be the set containing all functions obtainable by evaluating E with a key k from K and leaving the input slot empty.

A PRP E is considered secure if it is indistinguishable from a random permutation in $\text{Perms}[X]$. Intuitively, this means that an adversary who can only query the function (without knowing the secret key) cannot tell the difference between interacting with the actual PRP E and interacting with a truly random permutation of elements in X.

PRF $\implies$ PRG
Let $F : K \times \{0, 1\}^n \rightarrow \{0, 1\}^n$ be a secure PRF. Then the following $G : K \rightarrow$

$\{0,1\}^{nt}$ is a secure PRG:

$$G(k) = F(k,0) \parallel F(k,1) \parallel \cdots \parallel F(k,t-1)$$

The wonderful thing about this is that it is parallelizable which means it can be run on multiple cores making the process of encryption much faster. $F(k,\cdot)$ indistinguishable from random function $f(\cdot)$. This PRG construction is called "counter mode".

7.1 DES(Data Encryption Standard)

This is a block cipher that was widely used by banking(ACH-Automated Clearing House) and commerce in 1980s - 90s .

7.1.1 Construction

The core idea of DES were Feistel Networks. Given functions $f_1, ..., f_d : \{0,1\}^n \rightarrow \{0,1\}^n$

Feistel Network is an invertible function with the inverted circuit being similar

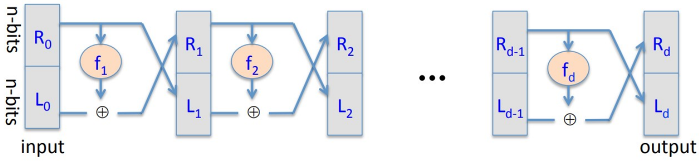


Figure 8: Feistel Network

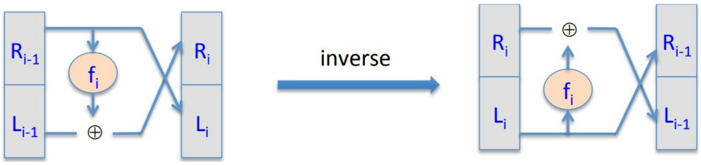


Figure 9: Inverse Feistel Network

to the encryption circuit but just the round keys would be applied in reverse order. This feistel network is used in many block ciphers but not AES.

A secure PRP from a 3 round Feistel

Luby Rackoff Theorem proves that we can deduce a secure PRP from a 3-round Feistel Network .

Theorem: Let $f : K \times \{0,1\}^n \longrightarrow \{0,1\}^n$ be a secure PRF. Then, there exists a 3-round Feistel network $F : K^3 \times \{0,1\}^{2n} \longrightarrow \{0,1\}^{2n}$ that is a secure PRP.

DES is a 16-round Feistel Network which has a key length of 56 bits and block length of 64 bits where

$$f_1, \ldots, f_{16} : \{0,1\}^{32} \longrightarrow \{0,1\}^{32}, \quad f_i(x) = F(k_i, x)$$

First we do an Initial Permutation and finally we do a Final Permutation . This is not for security but it is a standard. In the Feistel function we have specific boxes S-boxes and P-boxes which inturn due to some linearity caused DES to be broken.

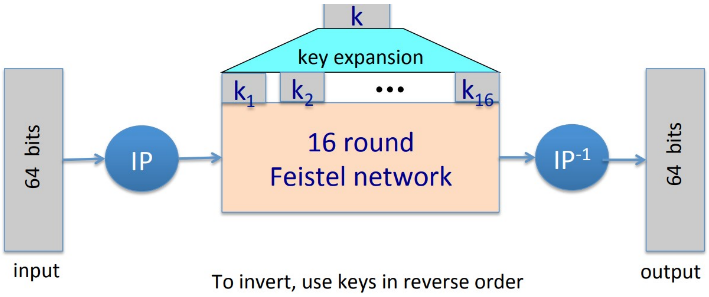


Figure 10: Implementation of DES

S-Box Role in Feistel Function

- During each of the 16 rounds of DES, the right half of the data is expanded from 32 bits to 48 bits using the expansion function.
- This 48-bit data is then XORed with the 48-bit round key.
- The result is divided into eight 6-bit blocks, each fed into one of the eight S-boxes.
- Each S-box produces a 4-bit output, resulting in a total of 32 bits.

P-Box Role in Feistel Function

- The 32-bit output from the S-boxes is then permuted using the P-box permutation table.
- This permuted result is then XORed with the left half of the data from the previous round.

If the S-box would have been just a fixed binary matrix it would create a lot of predictability in the ciphers and DES would be eventually broken. So there were some rules which the developers considered while making the S-boxes like No output bit should be close to a linear function of the input bits , S-boxes are 4-to-1 maps.

7.1.2 Attacks on DES

Exhaustive Search for Block Cipher Key

Goal: Given a few input-output pairs $(m_i, c_i = E(k, m_i))$ for $i = 1, \ldots, 3$, find the key k.

Lemma: Suppose DES is an *ideal cipher*

$$(2^{56} \text{ random invertible functions } \{0,1\}^{64} \to \{0,1\}^{64})$$

Then $\forall m, c$ there is at most <u>one</u> key k such that $c = \text{DES}(k, m)$.

Proof:

$$\begin{aligned}&\Pr[\exists k' \neq k : c = \text{DES}(k,m) = \text{DES}(k',m)] \leq \\ &\leq \sum_{k' \in \{0,1\}^{56}} \Pr[\text{DES}(k,m) = \text{DES}(k',m)] = 2^{56} \cdot \frac{1}{2^{64}} = \frac{1}{2^8} = \frac{1}{256}\end{aligned}$$

Proof for 99.5% probability:

$$\begin{aligned}\Pr[\text{unique key}] &= 1 - \Pr[\text{not unique key}] \\ &\geq 1 - \frac{1}{256} \\ &= \frac{255}{256} \\ &\approx 0.99609375 \\ &\approx 99.5\%\end{aligned}$$

If we check the probabilities if two messages are encrypted by using the same key, the unicity probabilities are as follows

- In DES $\approx 1 - \frac{1}{2^{71}}$
- In AES $\approx 1 - \frac{1}{2^{128}}$

showing us there will be atleast one pair and just two input/output pairs are enough for exhaustive key search. So the main reason that this cipher is broken is that , it has a very small key-size(2^{56}) and at a point of time in 2006 any message encrypted just got broken in 7 days.

7.1.3 Building up on DES

Method 1: Triple-DES

- Let $E : K \times M \to M$ be a block cipher.
- Define $3E : K^3 \times M \to M$ as:

$$3E((k_1, k_2, k_3), m) = E(k_1, D(k_2, E(k_3, m)))$$

- For 3DES: key size $= 3 \times 56 = 168$ bits. *(Increased key size enhances security.)*
- 3DES is three times slower than DES. *(Due to three encryption steps.)*
- Simple attack in time $\approx 2^{118}$. *(Still significantly better than 2^{56} for single DES.)*

Why not Double-DES?

- Define $2E((k_1, k_2), m) = E(k_1, E(k_2, m))$.
- Vulnerable to meet-in-the-middle attack.
- Attack: $M = (m_1, \ldots, m_{10})$, $C = (c_1, \ldots, c_{10})$.
- Step 1: Build table and sort on 2nd column to reduce search complexity.

m	$E(k_2, \cdot)$	$E(k_1, \cdot)$	c
$k_0 = 00\ldots00$	$E(k_0, M)$		
$k_1 = 00\ldots01$	$E(k_1, M)$		
$k_2 = 00\ldots10$	$E(k_2, M)$		
$\vdots$	$\vdots$		
$k_N = 11\ldots11$	$E(k_N, M)$		

- 2^{56} entries. *(Each key pair produces a unique output.)*

Meet-in-the-middle Attack

- Attack: $M = (m_1, \ldots, m_{10})$, $C = (c_1, \ldots, c_{10})$.
- Step 1: Build table of possible encryptions.
- Step 2: For all $k \in \{0, 1\}^{56}$ do:
  - Test if $D(k, C)$ is in the 2nd column.
  - If so, then $E(k_i, M) = D(k, C) \Rightarrow (k_i, k) = (k_2, k_1)$. *(Finding a match reveals key pair.)*

m	$E(k_2, \cdot)$	$E(k_1, \cdot)$	c
$k_0 = 00\ldots00$	$E(k_0, M)$		
$k_1 = 00\ldots01$	$E(k_1, M)$		
$k_2 = 00\ldots10$	$E(k_2, M)$		
$\vdots$	$\vdots$		
$k_N = 11\ldots11$	$E(k_N, M)$		

- Time $= 2^{56}\log(2^{56}) + 2^{56}\log(2^{56}) < 2^{63} \ll 2^{112}$, space $\approx 2^{56}$. *(Attack complexity much lower than double DES key size.)*
- Same attack on 3DES: Time $= 2^{118}$, space $\approx 2^{56}$. *(Higher security but more resource-intensive.)*

Method 2: DESX

- Let $E : K \times \{0,1\}^n \to \{0,1\}^n$ be a block cipher.
- Define E_X as:

$$E_X((k_1, k_2, k_3), m) = k_1 \oplus E(k_2, m \oplus k_3)$$

- For DESX: key length $= 64 + 56 + 64 = 184$ bits. *(Enhanced security with minimal overhead.)*
- ... but easy attack in time $\approx 2^{64+56} = 2^{120}$. *(Still vulnerable to certain attacks)*

Implementation Attacks

- **Side-channel attacks:**
  - **Timing Analysis:** Measures the time taken for encryption/decryption operations. Variations in timing can reveal information about the cryptographic algorithm or the secret key.
  - **Power Analysis:** Monitors the power consumption during encryption/decryption processes. Different operations can cause characteristic power consumption patterns, which can be analyzed to infer information about the secret key or the data being processed.
- **Fault attacks:**
  - Computing errors in the last round can expose the secret key k.
  - *Recommendation: Avoid implementing cryptographic primitives yourself.*

Linear and Differential Attacks

- **Linear Cryptanalysis:**
  - Given many input/output pairs, the key can be recovered in time less than 2^{56}.
  - Suppose for random k, m:

    $$\Pr[m[i_1] \oplus \cdots \oplus m[i_r] \oplus c[j_1] \oplus \cdots \oplus c[j_v] = k[l_1] \oplus \cdots \oplus k[l_u]] = \frac{1}{2} + \epsilon$$

  - For DES, $\epsilon = \frac{1}{2^{21}} \approx 0.0000000477$.
- **Attack Method:**

  - Given $\frac{1}{\epsilon^2}$ random $(m, c = \text{DES}(k, m))$ pairs, then:

    $$k[l_1, \dots, l_u] = \text{MAJ}[m[i_1, \dots, i_r] \oplus c[j_1, \dots, j_v]]$$

    with probability $\geq 97.7\%$.
  - For DES, $\epsilon = \frac{1}{2^{21}} \Rightarrow$ with 2^{42} input/output pairs, can find $k[l_1, \dots, l_u]$ in time 2^{42}.
  - Can find 14 key bits this way in time 2^{42}.
  - Brute force remaining $56 - 14 = 42$ bits in time 2^{42}.
  - Total attack time $\approx 2^{43}$, which is much less than 2^{56}, using 2^{42} random input/output pairs.

- A tiny bit of linearity in S_5 can lead to a 2^{42} time attack.

Quantum Attacks

- **Generic Search Problem:** Let $f : X \longrightarrow \{0, 1\}$ be a function. The goal is to find $x \in X$ such that $f(x) = 1$.
- **Classical Computer:** Best generic algorithm time complexity is $O(|X|)$.
- **Quantum Computer [Grover '96]:** Generic search time complexity is $O(\sqrt{|X|})$.
- **Feasibility of Quantum Computers:** Whether quantum computers can be practically built remains unknown.

Quantum Exhaustive Search

- Given $m, c = E(k, m)$, define Grover's algorithm: a quantum computer can find k in time $O(\sqrt{|K|})$.
- For DES: time complexity is approximately 2^{28}, for AES-128: time complexity is approximately 2^{64}.

A quantum computer could potentially break 256-bit key ciphers (e.g., AES-256) but it would need a lot of time to do it . So using long keys can keep us away from quantum attacks.

7.2 AES (Advanced Encryption Standard)

AES was created as a successor to DES & 3DES . It can have key size of 128/192/256 bits and a block size of 128 bits.The larger the key-size the slower the encryption is but it provides higher security. AES is based on Subs-Perm and not of Feistel Network.

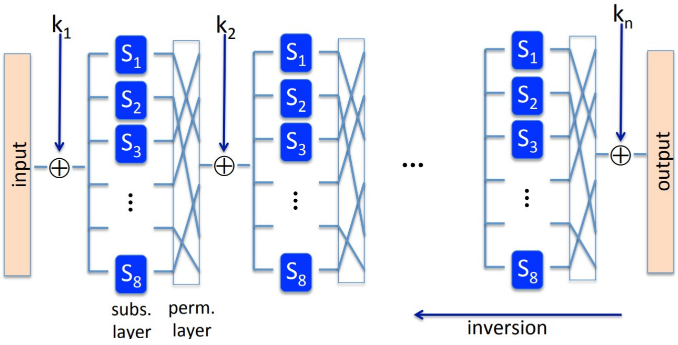


Figure 11: Subs-Perm Network

This change was necessary because in Feistel boxes some of the S-boxes were linear. AES has a similar structure of block ciphers that are built by iteration and the schematic for a 128 bit key sizee appears as follows.

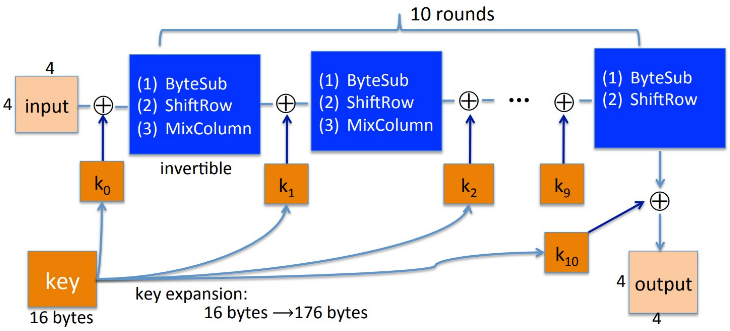


Figure 12: Implementation of AES

As we see in the image , the round function consists of 3 types of operations

- Byte Substitution - S-boxes are used here
- Shift Row - Rows are shifted randomly
- Mix Column - Columns are also mixed

Since AES has a lot of pre-computations to be done , if there are high-end servers then the pre-computations are done and stored resulting in a high usage of memory but the implementation will be very fast.

Usage of AES

An example of an AES implementation is a JavaScript library (approximately 6.4KB) that performs AES without pre-computed tables. This approach offers

Table 2: Code Size/Performance Tradeoff

Code	**Size/Performance**	**Implementation**
Pre-computed round functions (24KB or 4KB)	Largest	Fastest: table lookups and xors
Pre-computed S-box only (256 bytes)	Smaller	Slower
No pre-computation	Smallest	Slowest

flexibility but it is slower than hardware-accelerated options.For improved performance in web browsers, some implementations pre-compute the AES tables before encryption. This reduces the computational cost during the encryption process itself. Some processors are specifically designed for AES like Intel WestMere Processors which encrypts much faster(nearly 14x) than the software implementations on OpenSSL.

7.2.1 Attacks on AES

There are some attacks proved in certain documents showing if we have four related key input/output pairs we can do four times better then exhaustive search (which is around 2^{126} time , which is very large and it is impractical to do it with classical computers). Another related key attack , targets AES-256 with 2^{99} input-output pairs then we could crack the key within a time of roughly 2^{99} (which is still very large)

Constructing PRFs from PRGs

Building a 1-bit PRF

We can build a pseudorandom function (PRF) from a pseudorandom generator (PRG). Let's consider a secure PRG $G : K \to K^2$. We can define a 1-bit PRF $F : K \times \{0,1\} \to K$ as follows:

$$F(k, x \in \{0,1\}) = G(k)[x]$$

Here, $G(k)[x]$ denotes extracting the x-th bit from the output of $G(k)$. We know that that if G is a secure PRG, then F is a secure PRF. So a secure PRF is generated.

Extending the PRF Domain

We can extend the PRF to have a larger domain. Let's revisit the PRG $G : K \to K^2$. We can define a new function $G_1 : K \to K^4$ as:

$$G_1(k) = G(G(k)[0]) \parallel G(G(k)[1])$$

This essentially combines the outputs of two calls to G based on the first bit of the initial key. This construction allows us to define a 2-bit PRF:

$$F(k, x \in \{0,1\}^2) = G_1(k)[x]$$

Here, x is a 2-bit input, and $G_1(k)[x]$ extracts the appropriate bit based on the binary representation of x.

We can show that G_1 can also be considered a secure PRG based on the security of G. The intuition behind this is that distinguishing the output of G_1 from a random string in K^4 is computationally difficult because it would require breaking the security of G twice.

Extending to Arbitrary Lengths

The same concept can be generalized to construct PRFs with even larger domains. For a general PRG $G : K \to K^2$, we can define a function $G_n : K \to K^{2^n}$ as:

$$G_n(k) = G(G_{n-1}(k)[0]) \parallel G(G_{n-1}(k)[1])$$

This recursively builds a sequence of functions $G_1, G_2, \ldots, G_n$, where G_n outputs 2^n bits. Consequently, we can define an n-bit PRF:

$$F(k, x \in \{0,1\}^n) = G_n(k)[x]$$

Here, x is an n-bit input, and $G_n(k)[x]$ extracts the bit corresponding to the binary representation of x.

The security under the assumption of a secure G, each G_i in the sequence can be considered a secure PRG. The security relies on the difficulty of breaking the underlying PRG multiple times.

The GGM PRF

The GGM PRF (named after Goldreich, Goldwasser, and Micali) is a specific instantiation of this construction. It defines a PRF $F : K \times \{0,1\}^n \to K$ as:

$$F(k, x = x_0 x_1 \ldots x_{n-1} \in \{0,1\}^n) = G(k)[x_0] \parallel G(k_1)[x_1] \parallel \ldots \parallel G(k_{n-1})[x_{n-1}]$$

Here, the key k is expanded into n subkeys $k, k_1, \ldots, k_{n-1}$, and each subkey is used to evaluate G on the corresponding bit of the input x.

The GGM PRF is secure assuming the underlying PRG G is secure. However, it is generally not used in practice due to its slow performance, as it requires multiple evaluations of the PRG for each function evaluation.

7.3 Modes of usage of block ciphers

7.3.1 PRF Switching Lemma

The PRF Switching Lemma states that a secure pseudo-random permutation (PRP) can be used as a secure pseudo-random function (PRF) if the input space is large enough.

Let E be a PRP over (K, X). For any adversary A making at most q queries:

$$|\text{AdvPRF}[A, E] - \text{AdvPRP}[A, E]| < \frac{q^2}{2|X|}$$

where:

- AdvPRF$[A, E]$: Advantage of A in distinguishing E from a random PRF.
- AdvPRP$[A, E]$: Advantage of A in distinguishing E from a random permutation.

If $\frac{q^2}{2|X|}$ is negligible, and $Adv_{PRP}[A, E]$ is also negligible for a secure PRP, then $Adv_{PRF}[A, E]$ is negligible. Thus, an adversary cannot efficiently distinguish the PRP from a random PRF when the input space is sufficiently large.

7.3.2 One-Time Key

Using a new key for every message (e.g., encrypted email) ensures that an adversary sees only one ciphertext per key, aiming for semantic security.For this we need to build secure encryption from a secure PRP (e.g., AES).

Incorrect use of PRP : Electronic Code Book(ECB)

If $m_1 = m_2$, then $c_1 = c_2$. This makes ECB mode insecure for messages with more than one block.This reveals data about what was encrypted and this becomes crucial whenever we are encrypting images or files . An image can have same elements spreaded over some place even after encryption they all look similar telling us that all these are common.

Semantic Security (One-Time Key)

For one-time keys, semantic security is defined by:

$$Adv_{SS}[A, \text{OTP}] = |\Pr[\text{EXP}(0) = 1] - \Pr[\text{EXP}(1) = 1]|$$

This should be negligible, meaning the adversary cannot distinguish between encryptions of two messages of equal length.

ECB mode fails semantic security for multi-block messages. Example: If m_0 = "Hello World" and m_1 = "Hello Hello", and if $c_1 = c_2$, then the adversary can distinguish the messages with advantage 1.

7.3.3 Deterministic Counter Mode

Deterministic counter mode (e.g., AES, 3DES) is constructed from a PRF:

$$\text{EDETCTR}(k, m) = m[0] \oplus F(k, 0) \parallel m[1] \oplus F(k, 1) \parallel \cdots \parallel m[L] \oplus F(k, L)$$

Deterministic Counter-Mode Security

Theorem: For any $L > 0$, if F is a secure PRF, then EDETCTR is a semantically secure cipher. The semantic security advantage:

$$Adv_{SS}[A, \text{EDETCTR}] = 2 \cdot Adv_{PRF}[B, F]$$

Since F is a secure PRF, $Adv_{PRF}[B, F]$ is negligible, making $Adv_{SS}[A, \text{EDETCTR}]$ negligible.

7.3.4 Many Time-key

This is mainly used File systems where same AES key used to encrypt many files and IPsec where Same AES key used to encrypt many packets.

Semantic Security for Many-Time Key

When a key is used more than once, the adversary sees many ciphertexts with the same key. This leads to a chosen-plaintext attack (CPA) scenario where the adversary can obtain the encryption of arbitrary messages of their choice.

Formal Definition of CPA Security

A cipher $E = (E, D)$ defined over (K, M, C) is semantically secure under CPA if for all efficient adversaries A:

$$\text{AdvCPA}[A, E] = |\Pr[\text{EXP}(0) = 1] - \Pr[\text{EXP}(1) = 1]| \text{ is negligible.}$$

If $E(k, m)$ always outputs the same ciphertext for the same message m, an attacker can identify that two encrypted files or packets are the same. This is particularly problematic when the message space M is small.

7.3.5 Randomized Encryption

To prevent attacks due to repeated ciphertexts, use randomized encryption where $E(k, m)$ is a randomized algorithm. Encrypting the same message twice will yield different ciphertexts with high probability. However, this requires the ciphertext to be longer than the plaintext.

$$\text{CT-size} = \text{PT-size} + \texttt{\# random bits}$$

7.3.6 Nonce-based Encryption

Nonce (n): A value that changes with each message. The pair (k, n) is never reused.

Method 1: Counter Nonce - Nonce is a counter (e.g., packet counter). - Used when the encryptor maintains state across messages. - If the decryptor has the same state, the nonce need not be sent with the ciphertext.

Method 2: Random Nonce - Encryptor chooses a random nonce $n \leftarrow N$.

$$\begin{aligned}\text{Alice:} \quad & E(k, m, n) = c \\ \text{Bob:} \quad & D(k, c, n) = m\end{aligned}$$

The system must be secure even when nonces are chosen adversarially.

Definition: A nonce-based encryption scheme E is semantically secure under CPA if for all efficient adversaries A:

$$\text{Adv}_{\text{nCPA}}[A, E] = |\Pr[\text{EXP}(0) = 1] - \Pr[\text{EXP}(1) = 1]| \text{ is negligible.}$$

$$\begin{aligned}\text{Challenge:} \quad & k \leftarrow K,\ n_i,\ m_{i,0},\ m_{i,1} : |m_{i,0}| = |m_{i,1}| \\ \text{Adversary:} \quad & c \leftarrow E(k, m_{i,b}, n_i),\ b' \in \{0, 1\}\end{aligned}$$

All nonces $\{n_1, \ldots, n_q\}$ must be distinct for $i = 1, \ldots, q$.

7.3.7 CBC with Random IV

Construction 1: CBC with Random IV

Block Ciphers can be used with Cipher Block Chaining (CBC) with a random Initialization Vector (IV) . Let (E, D) be a Pseudorandom Permutation (PRP). The encryption process $E_{\text{CBC}}(k, m)$ proceeds as follows: - Choose a random IV $\in X$. - Perform encryption with CBC mode using the random IV. Each plaintext block is XORed with the previous ciphertext block before encryption.

$$\begin{aligned}c[0] &= E(k, IV \oplus m[0]) \\ c[i] &= E(k, c[i-1] \oplus m[i]) \quad \text{for } i = 1, 2, \ldots\end{aligned}$$

Decryption: To decrypt, each ciphertext block is decrypted and then XORed with the previous ciphertext block.

$$\begin{aligned}m[0] &= D(k, c[0]) \oplus IV \\ m[i] &= D(k, c[i]) \oplus c[i-1] \quad \text{for } i = 1, 2, \ldots\end{aligned}$$

This process ensures that identical plaintext blocks will result in different ciphertext blocks due to the chaining mechanism.

CPA Security Analysis of CBC

Theorem: For any $L > 0$, if E is a secure PRP over (K, X), then E_{CBC} is semantically secure under Chosen Plaintext Attack (CPA) over (K, X^L, X^{L+1}).

For a q-query adversary A attacking E_{CBC}, there exists a PRP adversary B such that:

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

Note: CBC is only secure as long as $q^2L^2 \ll |X|$.

Example: - For AES: $|X| = 2^{128} \Rightarrow qL < 2^{48}$ - For 3DES: $|X| = 2^{64} \Rightarrow qL < 2^{16}$

This indicates the maximum number of blocks that can be securely encrypted before needing to change the key.

Security Warning

CBC is not CPA-secure if the attacker can predict the IV. If an attacker can predict the IV, they can launch certain attacks that compromise the security of the encryption scheme.

Example: - A bug in SSL/TLS 1.0: The IV for record i is the last ciphertext block of record $i-1$. This predictability can be exploited by attackers.

Construction 1': Nonce-based CBC

To mitigate the issue with predictable IVs, we use a nonce-based CBC. In this construction, a unique nonce is used for each message, ensuring that the $(\text{key}, \text{nonce})$ pair is used only once.

$$E(k, m, \text{nonce}) = c$$

The nonce ensures that even if the same message is encrypted multiple times, it will produce different ciphertexts.

Caution in Crypto API (OpenSSL) The following function from the OpenSSL library illustrates how CBC encryption is performed . Here, the user supplies the IV, which could be a nonce if it is unique for each encryption. So while implementing with OpenSSL we should be careful if we are randomizing our IV or not.

CBC Technicality: Padding

In CBC mode, the message must be a multiple of the block size. Padding is added to the plaintext to ensure this requirement.

Example: For padding, if $n > 0$, add n byte pad as follows: $n, n, n, \ldots, n$. If no pad is needed, a dummy block is added to maintain consistency.

The padding is removed during decryption, ensuring the original message is recovered.

7.3.8 Counter Mode Encryption

Random Counter Mode (CTR)

Counter Mode (CTR) is a mode of operation for block ciphers that encrypts plaintext by XORing it with the output of a pseudorandom function (PRF) applied to a counter. This mode is parallelizable, unlike CBC.

Let $F : K \times \{0,1\}^n \to \{0,1\}^n$ be a secure PRF.

Encryption $E(k, m)$: choose a random $IV \in \{0,1\}^n$ and compute:

$$\begin{aligned} &\text{Encryption:} \\ &\quad c[i] = m[i] \oplus F(k, IV + i) \quad \text{for } i = 0, 1, \ldots, L \end{aligned}$$

Nonce-based Counter Mode

Nonce-based CTR mode uses a unique nonce for each message to ensure that the PRF input is never reused.

Nonce: 128 bits (64-bit nonce + 64-bit counter)

Ensure $F(k, x)$ is never reused by choosing the nonce to start at 0 for every message.

Security Analysis

Counter-mode Theorem: For any $L > 0$, if F is a secure PRF over (K, X, X), then E_{CTR} is semantically secure under CPA over (K, X^L, X^{L+1}).

For a q-query adversary A attacking E_{CTR}, there exists a PRF adversary B such that:

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

Note: CTR mode is secure as long as $q^2 L \ll |X|$, which is better than CBC.

Example: - For AES: $|X| = 2^{128} \Rightarrow qL < 2^{48}$

Comparison: CTR vs. CBC

Feature	CBC	CTR Mode
Uses	PRP	PRF
Parallel Processing	No	Yes
Security of Random Enc.	$q^2 L^2 \ll \lvert X \rvert$	$q^2 L \ll \lvert X \rvert$
Dummy Padding Block	Yes	No
1-byte Messages	16x Expansion (Nonce-based)	No Expansion

Neither mode ensures data integrity.

7.4 Performance of Block Ciphers in comparison to Stream Ciphers

This is a comparison between block ciphers and stream ciphers.

Cipher	Block/key size	Speed (MB/sec)
RC4	Stream	126
Salsa20/12	Stream	643
Sosemanuk	Stream	727
3DES	64/168	13
AES-128	128/128	109

Table 3: Performance comparison of cryptographic algorithms using Crypt++ version 5.6.0 [Wei Dai] on an AMD Opteron 2.2 GHz (Linux)

8 Data Integrity

We show integrity first without caring about the confidentiality. Examples where this data integrity becomes crucial is when protecting public system files on disk , protecting banner ads on web pages . It aims to not change the data whatever we are sending or requesting.

8.1 MAC (Message Authentication Codes)

Definition: A MAC $I = (S, V)$ over (K, M, T) consists of:

- $S(k, m)$ outputs $t \in T$
- $V(k, m, t)$ outputs 'yes' or 'no'

Take an example of Alice and Bob who are communicating to each other. For suppose Bob wants to verify that the message has come from Alice and not modified in between while reaching him . Bob could verify the tag and if it says yes indicating the message hasn't been modified.

Integrity Requires a Secret Key

If someone uses CRC(Cyclic Redundancy Check) and doesn't use some secret key then an attacker can easily modify m and re-compute CRC. CRC is designed to detect random, not malicious, errors.

8.1.1 Secure MACs

Attacker's Power: Chosen message attack

- For $m_1, m_2, \ldots, m_q$, attacker gets $t_i \leftarrow S(k, m_i)$.

Attacker's Goal: Existential forgery

- Produce a new valid message/tag pair (m, t) not in $\{(m_1, t_1), \ldots, (m_q, t_q)\}$.
- Cannot produce (m, t') for $t' \neq t$.

Definition: A MAC $I = (S, V)$ is secure if for all "efficient" attackers A:

$$\text{Adv}_{\text{MAC}}[A, I] = \Pr[\text{Chal. outputs 1}] \text{ is negligible.}$$

Example: Protecting System Files

- Virus infects and modifies system files.
- User reboots into a clean OS and supplies a password.
- With a secure MAC, all modified files will be detected.

At installation, the system computes:

$$\begin{array}{ll} F_1 & t_1 = S(k, F_1) \\ F_2 & t_2 = S(k, F_2) \\ \vdots & \\ F_n & t_n = S(k, F_n) \end{array}$$

Where k is derived from the user's password.

MACs Based on PRFs

Secure PRF $\Rightarrow$ Secure MAC

For a PRF $F : K \times X \to Y$, define a MAC $I_F = (S, V)$ as:

- $S(k, m) := F(k, m)$
- $V(k, m, t)$: outputs 'yes' if $t = F(k, m)$, 'no' otherwise.

Security

Theorem: If $F : K \times X \to Y$ is a secure PRF and $1/|Y|$ is negligible (i.e., $|Y|$ is large), then I_F is a secure MAC.

For every efficient MAC adversary A attacking I_F, there exists an efficient PRF adversary B attacking F such that:

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

Proof:

- Suppose $f : X \to Y$ is a truly random function.
- MAC adversary A wins if $t = f(m)$ and $m \notin \{m_1, \ldots, m_q\}$.
- Therefore, $\Pr[A \text{ wins}] = \frac{1}{|Y|}$.

Some examples

- AES: A MAC for 16-byte messages.
- Two main constructions used in practice:
  - CBC-MAC (banking – ANSI X9.9, X9.19, FIPS 186-3)
  - HMAC (Internet protocols: SSL, IPsec, SSH, ...)
- Both convert a small-PRF into a big-PRF.

8.1.2 Truncating MACs Based on PRFs

Lemma: Suppose $F : K \times X \to \{0,1\}^n$ is a secure PRF. Then $F_t(k,m) = F(k,m)[1 \ldots t]$ is also a secure PRF for all $1 \leq t \leq n$.

- If (S,V) is a MAC based on a secure PRF outputting n-bit tags, then the truncated MAC outputting w bits is secure as long as $1/2^w$ is negligible (e.g., $w \geq 64$).

8.2 CBC-MAC

8.2.1 ECBC-MAC

The construction is based on CBC mode itself except that a new random key is also taken for finally encrypting the message.

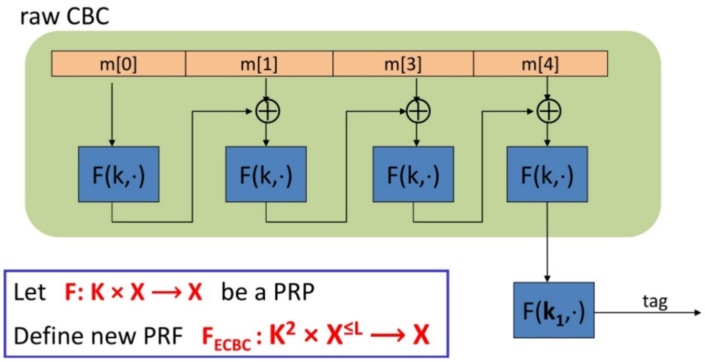

raw CBC
m[0]
m[1]
m[3]
m[4]
F(k,·)
F(k,·)
F(k,·)
F(k,·)
F(k₁,·)
tag
Let F: K × X ⟶ X be a PRP
Define new PRF $F_{ECBC}: K^2 \times X^{\leq L} \longrightarrow X$

Figure 13: Encrypted CBC-MAC

Why the Last Encryption Step in ECBC-MAC?

If we define a MAC $I_{\text{RAW}} = (S,V)$ where

$$S(k,m) = \text{rawCBC}(k,m)$$

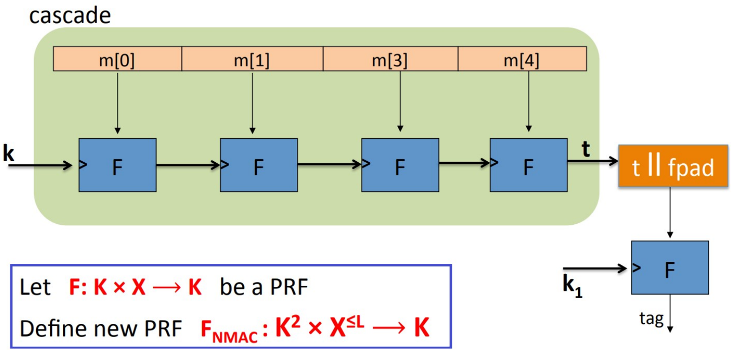
then I_{RAW} is easily broken using a single chosen message attack. The adversary works as follows:

- Choose an arbitrary one-block message $m \in X$.
- Request the tag for m. Obtain $t = F(k,m)$.
- Output t as a MAC forgery for the two-block message $(m, t \oplus m)$.

$$\text{rawCBC}(k,(m,t \oplus m)) = F(k, F(k,m) \oplus (t \oplus m)) = F(k, t \oplus (t \oplus m)) = t$$

8.2.2 NMAC

Here also as in the case of CBC-MAC , cascade alone doesn't provide security but the final random key adds the security.



Let $F: K \times X \longrightarrow K$ be a PRF

Define new PRF $F_{NMAC}: K^2 \times X^{\leq L} \longrightarrow K$

Figure 14: Implementation of NMAC

8.2.3 ECBC-MAC & NMAC Analysis

Theorem: For any $L > 0$:

- For every efficient q-query PRF adversary A attacking F_{ECBC} or F_{NMAC}, there exists an efficient adversary B such that:

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

$$\text{Adv}_{\text{PRF}}[A, F_{\text{NMAC}}] \leq q \cdot L \cdot \text{Adv}_{\text{PRF}}[B, F] + \frac{q^2}{2|K|}$$

- CBC-MAC is secure as long as $q \ll |X|^{1/2}$.
- NMAC is secure as long as $q \ll |K|^{1/2}$ (e.g., 2^{64} for AES-128).

8.2.4 Security Bounds

The security bounds for ECBC-MAC and NMAC are tight. After signing $|X|^{1/2}$ messages with ECBC-MAC or $|K|^{1/2}$ messages with NMAC, the MACs become insecure.

Extension Property

If the underlying PRF F is a PRP (e.g., AES), then both PRFs (ECBC and NMAC) have the following extension property:

$$\forall x, y, w : \quad F_{\text{BIG}}(k, x) = F_{\text{BIG}}(k, y) \Rightarrow F_{\text{BIG}}(k, x \parallel w) = F_{\text{BIG}}(k, y \parallel w)$$

Generic Attack on Derived MAC

Let $F_{\text{BIG}} : K \times X \rightarrow Y$ be a PRF with the extension property. The attack proceeds as follows:

1. Issue $|Y|^{1/2}$ message queries for random messages in X. Obtain (m_i, t_i) for $i = 1, \ldots, |Y|^{1/2}$.
2. Find a collision $t_u = t_v$ for $u \neq v$ (one exists with high probability by the birthday paradox).
3. Choose some w and query for $t := F_{\text{BIG}}(k, m_u \parallel w)$.
4. Output the forgery $(m_v \parallel w, t)$. Indeed, $t := F_{\text{BIG}}(k, m_v \parallel w)$.

Improved Security: Randomized Construction

Let $F : K \times X \rightarrow X$ be a PRF. Result: MAC with tags in X^2.

$$\text{Security:} \quad \text{Adv}_{\text{MAC}}[A, I_{\text{RCBC}}] \leq \text{Adv}_{\text{PRP}}[B, F] \cdot \left(1 + \frac{2q^2}{|X|}\right)$$

For 3DES: can sign $q = 2^{32}$ messages with one key.

Message m	Tag t
rawCBC	k
r (random $r \in X$)	rawCBC
2 blocks	k_1

Comparison

- ECBC-MAC is commonly used as an AES-based MAC:
  - CCM encryption mode (used in 802.11i).
  - NIST standard called CMAC.
- NMAC is not usually used with AES or 3DES due to the need to change the AES key on every block, requiring re-computation of AES key expansion.
- NMAC is the basis for a popular MAC called HMAC.

8.2.5 CBC-MAC Padding

Consider the case where the message is not a multiple of block size , then you need to add some bits to proceed with encryption . This is padding. For security, padding must be invertible:

$$m_0 \neq m_1 \Rightarrow \text{pad}(m_0) \neq \text{pad}(m_1)$$

ISO Standard: Pad with ”1000...00”. Add a dummy block if needed. The ”1” indicates the beginning of the pad.

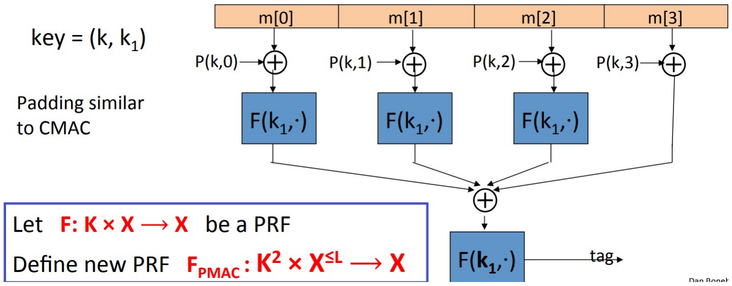
CMAC (NIST Standard)

CMAC is a variant of CBC-MAC with keys (k, k_1, k_2) . The keys k_1, k_2 are derived from k itself:

- No final encryption step (extension attack thwarted by last keyed XOR).
- No dummy block (ambiguity resolved by using k_1 or k_2).

8.3 PMAC

All of the MACs that we have seen till now are not parallelizable since each encryption requires a function of previous message . So a parallelizable MAC can be constructed with the following scheme



Let $\mathbf{F: K \times X \longrightarrow X}$ be a PRF

Define new PRF $\mathbf{F_{PMAC} : K^2 \times X^{\leq L} \longrightarrow X}$

Figure 15: Implementation of PMAC

8.3.1 PMAC Analysis

Theorem: For any $L > 0$, if F is a secure PRF over (K, X, X), then F_{PMAC} is a secure PRF over $(K, X \leq L, X)$.

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

PMAC is secure as long as $qL \ll |X|^{1/2}$.

8.3.2 One-Time MAC

A one-time MAC $I = (S, V)$ is secure if for all efficient adversaries A:

$$\text{Adv}_{\text{MAC}}[A, I] = \Pr[\text{Chal. outputs } 1] \text{ is negligible.}$$

Chal. Adv. $k \leftarrow K$ (m, t) $m_1 \in M$ $t_1 \leftarrow S(k, m_1)$ $b = 1$ if $V(k, m, t) =$ ‘yes’ and $(m, t) \neq (m_1, t_1)$ $b =$

One-Time MAC Example

Secure against all adversaries and faster than PRF-based MACs. Let q be a large prime (e.g., $q = 2^{128} + 51$):

key $= (a, b) \in \{1, \ldots, q\}^2$ msg $= (m[1], \ldots, m[L])$ S(key, msg) $= P_{\text{msg}}(a)+b \mod q$

where $P_{\text{msg}}(x) = x^{L+1} + m[L] \cdot x^L + \ldots + m[1] \cdot x$.

One-Time MAC Security (Unconditional)

Theorem: The one-time MAC on the previous slide satisfies:

$$\forall m_1 \neq m_2, t_1, t_2 : \Pr_{a,b}[S((a,b), m_1) = t1 \mid S((a,b), m_2) = t2] \leq \frac{L}{q}$$

Proof:

$$\forall m_1 \neq m_2, t1, t2 :$$

$$\Pr_{a,b}[S((a,b), m_2) = t2] = \Pr_{a,b}[P_{m_2}(a) + b = t2] = \frac{1}{q}$$

$$\Pr_{a,b}[S((a,b), m_1) = t1 \text{ and } S((a,b), m_2) = t2] = \Pr_{a,b}[P_{m_1}(a) - P_{m_2}(a) = t1 - t2 \text{ and } P_{m_2}(a) + b = t2] \leq \frac{L}{q^2}$$

$$\Rightarrow \text{ given valid } (m_2, t2), \text{ adv. outputs } (m_1, t1) \text{ and is right with prob. } \leq \frac{L}{q}$$

8.3.3 One-Time MAC to Many-Time MAC

Let (S, V) be a secure one-time MAC over $(K_I, M, \{0,1\}^n)$. Let $F : K_F \times \{0,1\}^n \to \{0,1\}^n$ be a secure PRF. Carter-Wegman MAC:

$$\text{CW}(k1, k2, m) = (r, F(k1, r) \oplus S(k2, m)) \quad \text{for random} \quad r \leftarrow \{0,1\}^n$$

Theorem: If (S, V) is a secure one-time MAC and F is a secure PRF, then CW is a secure MAC outputting tags in $\{0,1\}^{2n}$.

8.4 Hash Functions

8.4.1 Collision Resistance

Let $H : M \to T$ be a hash function where $|M| \gg |T|$. A collision for H is a pair $(m_0, m_1) \in M$ such that:

$$H(m_0) = H(m_1) \quad \text{and} \quad m_0 \neq m_1$$

A function H is collision-resistant if for all (explicit) efficient algorithms A:

$$\text{Adv}_{\text{CR}}[A, H] = \Pr[A \text{ outputs a collision for } H] \text{ is negligible.}$$

Example: SHA-256 (outputs 256 bits).

8.4.2 MACs from Collision Resistance

Let $I = (S, V)$ be a MAC for short messages over (K, M, T) (e.g., AES). Let $H : M_{\text{big}} \to M$. Define $I_{\text{big}} = (S_{\text{big}}, V_{\text{big}})$ over (K, M_{big}, T) as:

$$S_{\text{big}}(k, m) = S(k, H(m)), \quad V_{\text{big}}(k, m, t) = V(k, H(m), t)$$

Theorem: If I is a secure MAC and H is collision-resistant, then I_{big} is a secure MAC. Example: $S(k, m) = \text{AES-2-block-CBC}(k, \text{SHA-256}(m))$ is a secure MAC.

8.4.3 Security Implications of Collision Resistance

Collision resistance is necessary for security:

- Suppose an adversary can find $m_0 \neq m_1$ such that $H(m_0) = H(m_1)$.
- Then S_{big} is insecure under a 1-chosen message attack:
  - Step 1: Adversary asks for $t \leftarrow S(k, m_0)$.
  - Step 2: Outputs (m_1, t) as forgery.

$$S_{\text{big}}(k, m) = S(k, H(m)), \quad V_{\text{big}}(k, m, t) = V(k, H(m), t)$$

8.4.4 Protecting File Integrity Using Collision-Resistant Hashes

When a user downloads a package, they can verify that the contents are valid.

If H is collision-resistant $\Rightarrow$ attacker cannot modify package without detection.

No key is needed (public verifiability), but it requires read-only space.

Hashing is done using a fast MAC. Collision resistance ensures that given a tag and a message, one cannot find another message with the same tag. If someone finds even one collision, it implies that the MAC is insecure due to a 1-chosen message attack.

Linux distributions show public hashes to check whether the software was installed correctly.

8.4.5 The Birthday Paradox

Let $r_1, \ldots, r_n \in \{1, \ldots, B\}$ be independent and identically distributed integers.

Theorem: When $n = 1.2 \times B^{1/2}$, then

$$\Pr\left[\exists i \neq j : r_i = r_j\right] \geq \frac{1}{2}.$$

Proof: (For uniform independent $r_1, \ldots, r_n$)

$$\Pr\left[\exists i \neq j : r_i = r_j\right] = 1 - \Pr\left[\forall i \neq j : r_i \neq r_j\right]$$

$$= 1 - \left(\frac{B-1}{B}\right)\left(\frac{B-2}{B}\right)\cdots\left(\frac{B-n+1}{B}\right)$$

$$= 1 - \prod_{i=1}^{n-1}\left(1 - \frac{i}{B}\right) \approx 1 - e^{-\sum_{i=1}^{n-1}\frac{i}{B}} = 1 - e^{-\frac{n^2}{2B}}$$

For $n = 1.2 \times B^{1/2}$:

$$1 - e^{-0.72} \approx 0.53 > \frac{1}{2}.$$

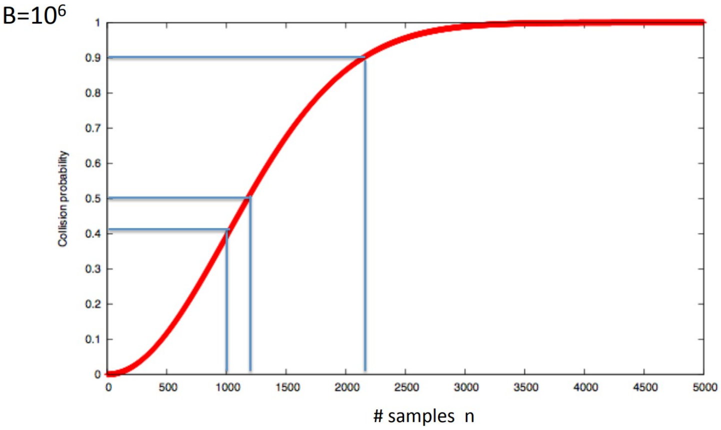


Figure 16: Graph for analyzing the Birthday Paradox

This Birthday Paradox came from trying to solve a problem to check how many people would have same birthdays if we take from B persons. If we take 365 people then within choosing 23 people you would have a good chance of two people having the same birthday. But we have a bias in real life in birthdays (not satisfying the condition independent) most people have their birthdays in September.

Generic Attack on Collision-Resistant Functions

Let $H : M \to \{0,1\}^n$ be a hash function with $|M| \gg 2^n$. A generic algorithm to find a collision in $O(2^{n/2})$ hashes is:

Algorithm:

1. Choose $2^{n/2}$ random messages in M: $m_1, \ldots, m_{2^{n/2}}$ (distinct with high probability).
2. For $i = 1, \ldots, 2^{n/2}$, compute $t_i = H(m_i) \in \{0,1\}^n$.
3. Look for a collision ($t_i = t_j$). If not found, go back to step 1.

Expected number of iterations ≈ 2. Running time: $O(2^{n/2})$ (space $O(2^{n/2})$).

Sample Collision-Resistant Hash Functions

Function	Digest Size (bits)	Speed (MB/sec)	Generic Attack Time
SHA-1	160	153	2^{80}
SHA-256	256	111	2^{128}
SHA-512	512	99	2^{256}
Whirlpool	512	57	2^{256}

NIST standards: The best-known collision finder for SHA-1 requires 2^{51} hash evaluations.

	Classical Algorithms	Quantum Algorithms
Block cipher $E : K \times X \to X$	Exhaustive search $O(\lvert K\rvert)$	$O(\lvert K\rvert^{1/2})$
Hash function $H : M \to T$	Collision finder $O(\lvert T\rvert^{1/2})$	$O(\lvert T\rvert^{1/3})$

8.4.6 Merkle-Damgard Iterated Construction

Given $h : T \times X \longrightarrow T$ (compression function), we obtain $H : X^{\leq L} \longrightarrow T$.

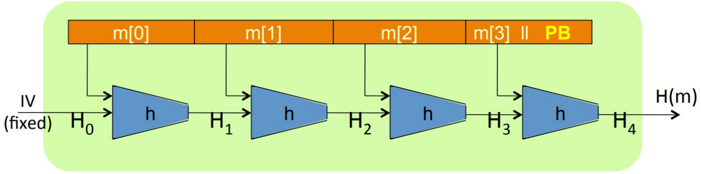


Figure 17: Merkley Demgard Iterated Construction

8.4.7 MD Collision Resistance

Theorem: If h is collision resistant then so is H . **Proof:** Collision on $H \Rightarrow$ Collision on h

Suppose $H(M) = H(M')$. We build a collision for h.

$$\begin{aligned} \text{IV} = H_0, \quad H_1, \quad \ldots, \quad H_t, \quad H_{t+1} = H(M) \\ \text{IV} = H'_0, \quad H'_1, \quad \ldots, \quad H'_r, \quad H'_{r+1} = H(M') \end{aligned}$$

$$h(H_t, M_t \parallel \text{PB}) = H_{t+1} = H'_{r+1} = h(H'_r, M'_r \parallel \text{PB}')$$

Case 1:

$$\text{If } \left(H_t \neq H'_r \quad \text{or} \quad M_t \neq M'_r \quad \text{or} \quad \text{PB} \neq \text{PB}'\right)$$

$$\Rightarrow \text{We have a collision on } h. \quad \text{STOP}$$

Otherwise,
Suppose $H_t = H'_r$ and $M_t = M'_r$ and $\text{PB} = \text{PB}'$ (i.e., $t = r$)

Then:

$$h(H_{t-1}, M_{t-1}) = H_t = H'_r = h(H'_{t-1}, M'_{t-1})$$

Case 2:

$$\text{If } \big(H_{t-1} \neq H'_{t-1} \quad \text{or} \quad M_{t-1} \neq M'_{t-1}\big)$$

$$\Rightarrow \text{We have a collision on } h. \quad \text{STOP}$$

Otherwise,

$$H_{t-1} = H'_{t-1} \quad \text{and} \quad M_{t-1} = M'_{t-1}$$

Iterate all the way to the beginning and either:

1. Find collision on h or
2. $\forall i : M_i = M'_i \Rightarrow M = M'$

This cannot happen because M and M' are a collision on H.
So, through this proof we can conclude that **To construct a collision resistant function , suffices to construct a compression function.**

8.4.8 Compression Function(h) from a Block Cipher

E: $K \times \{0,1\}^n \rightarrow \{0,1\}^n$ be a block cipher.

Davies-Meyer Compression Function

$$h(H, m) = E(m, H) \oplus H$$

Theorem: Suppose E is an ideal cipher (a collection of $|K|$ random permutations). Finding a collision $h(H, m) = h(H', m')$ takes $O(2^{n/2})$ evaluations of (E, D).

Miyaguchi-Preneel:

$$h(H, m) = E(m, H) \oplus H \oplus m \quad \text{(Whirlpool)}$$

$$h(H, m) = E(H \oplus m, m) \oplus m$$

Total of 12 variants like this. Other natural variants are insecure:

$$h(H, m) = E(m, H) \oplus m$$

8.4.9 SHA-256

SHA-256 has a 256-bit block size and 512-bit key size producing an output of 256-bit block.The block and key are put into a block cipher called SHACAL-2 which uses Merkle-Damgard function with Davies-Meyer compression functon .

Provable Compression Functions

These are functions that can provide excellent security as good as SHA-256 . We start by choosing a random 2000-bit prime p, and two random integers u and v such that $1 \leq u, v \leq p$.

For any $m, H \in \{0, \ldots, p-1\}$, define a hash function $h(H, m) = u^H \cdot v^m \bmod p$. Finding a collision for the hash function $h(\cdot, \cdot)$ is as computationally hard as solving the discrete logarithm problem modulo p.

Problem

The main issue with this approach is that computing $h(H, m)$ can be slow due to the complexity involved in modular exponentiation, especially with large primes like p.

8.5 HMAC

HMAC is one of the most widely used MAC on the internet. **H:** Hash function Example: SHA-256 (output is 256 bits)

$$S(k, m) = H((k \oplus \text{opad}) \parallel H((k \oplus \text{ipad}) \parallel m))$$

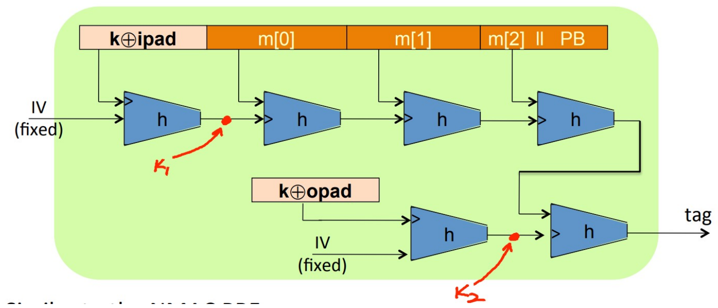


Figure 18: Implementation of HMAC

Similar to the NMAC PRF, main difference: the two keys k_1, k_2 are dependent.

HMAC Properties

Built from a black-box implementation of SHA-256.
HMAC is assumed to be a secure PRF:

- Can be proven under certain PRF assumptions about $h(.,.)$
- Security bounds similar to NMAC
- Need $q^2/|T|$ to be negligible ($q \ll |T|^{1/2}$)

In TLS: Must support HMAC-SHA1-96.

8.6 Verification Timing Attacks

Keyczar Crypto Library (Python)

```
def Verify(key, msg, sig_bytes):
    return HMAC(key, msg) == sig_bytes
```

The Problem: `==` implemented as a byte-by-byte comparison.

- Comparator returns false when the first inequality is found.

Timing Attack: To Compute Tag for Target Message m

1. Query server with a random tag.
2. Loop over all possible first bytes and query the server. Stop when verification takes a little longer than in step 1.
3. Check if the server returned the response in the same time as previously .
4. The observation here is that when we enter correct bits since it goes by a bit-by-bit check if the entered bit is correct it will go to the next bit for comparison and finally we can check if the entered bit is correct or not.
5. Repeat for all tag bytes until a valid tag is found.

Defense #1

Make string comparator always take the same time (Python):

```
def Verify(key, msg, sig_bytes):
    if len(sig_bytes) != expected_length:
        return False
    result = 0
    for x, y in zip(HMAC(key, msg), sig_bytes):
        result |= ord(x) ^ ord(y)
    return result == 0
```

- This approach can be difficult to ensure due to optimizing compilers.

Defense #2

Make string comparator always take the same time (Python):

```
def Verify(key, msg, sig_bytes):
    mac = HMAC(key, msg)
    return HMAC(key, mac) == HMAC(key, sig_bytes)
```

- Attacker doesn't know the values being compared.[1].

[1] Most of the images are taken from Dan Boneh's Slides of Cryptography 1 course.

9 Updated PoA

Week 5: *Hash Functions and Authenticated Encryption*

- Study about Digital Signatures and their applications
- Study hash functions , checksums for data and popular cryptographic hash functions such as SHA-256 and SHA-3
- Learn about authenticated encryption methods
- Understand how to ensure both confidentiality and integrity
- Learn about searching on encrypted data like homomorphic encryption and searchable encryption

Week 6: *Key Exchange and Computational Number Theory*

- Learn basic key exchange protocols and their working
- Study algorithms from computational number theory - Prime number generation for RSA , Digital Signature Algorithm
- Learn about public key infrastructure (PKI) and related algorithms
- Understand public key encryption and its applications

Week 7: *Advanced Public Key Cryptography*

- Study RSA and Diffie-Hellman protocols
- Learn about chosen ciphertext security (CCA security)
- Implementing some cryptographic algorithms by coding them
- Learning about cryptography that is secure from quantum computers

Week 8: *Secure Multiparty Computation and Real World Applications*

- Learn about secure multiparty computation (SMPC)
- Understanding practical implications and challenges
- Making Final report of what has been done