|  |  |  |  |
| --- | --- | --- | --- |
| **Course Name:** | **Information Security (116U01L602)** | **Semester:** | **VI** |
| **Date of Performance:** | **29 / 01 / 2025** | **DIV/ Batch No:** | **A-3** |
| **Student Name:** | **Kashish Mamania** | **Roll No:** | **16010122104** |

|  |
| --- |
| **Title: Application of RSA Algorithm for various security services like confidentiality, authentication, signature, non-repudiation and integrity** |

|  |
| --- |
| **Objectives:** |
| To write a program to convert plain text into cipher text using Caesar cipher and Transposition cipher |

|  |
| --- |
| **Expected Outcome of Experiment:** |
| **CO1,CO2** |

|  |
| --- |
| **Books/ Journals/ Websites referred:** |
| https:/[/www.openssl.org/](http://www.openssl.org/) https://github.com/openssl/openssl https://en.wikipedia.org/wiki/OpenSSL |

|  |
| --- |
| **Pre Lab/ Prior Concepts:** |
|  |

|  |
| --- |
| **New Concepts to be learned:** |
| OpenSSL |

|  |
| --- |
| **Abstract:** |
| SSL, or Secure Sockets Layer, is an encryption-based Internet security protocol. It was first developed by Netscape in 1995 for the purpose of ensuring privacy, authentication, and data integrity in Internet communications. SSL is the predecessor to the modern TLS encryption used today.  A website that implements SSL/TLS has "HTTPS" in its URL instead of "HTTP." Originally, data on the Web was transmitted in plaintext that anyone could read if they intercepted the message. For example, if a consumer visited a shopping website, placed an order, and entered their credit card number on the website, that credit card number would travel across the Internet unconcealed.  SSL was created to correct this problem and protect user privacy. By encrypting any data that goes between a user and a web server, SSL ensures that anyone who intercepts the data can only see a scrambled mess of characters. The consumer's credit card number is now safe, only visible to the shopping website where they entered it.  SSL also stops certain kinds of cyberattacks: It authenticates web servers, which is important because attackers will often try to set up fake websites to trick users and steal data. It also prevents attackers from tampering with data in transit, like a tamper-proof seal on a medicine container. |

|  |
| --- |
| **Related Theory:** |
| OpenSSL is a software library for applications that secure communications over computer networks against eavesdropping or need to identify the party at the other end. It is widely used by Internet servers, including the majority of HTTPS websites.  OpenSSL contains an open-source implementation of the SSL and TLS protocols. The core library, written in the C programming language, implements basic cryptographic functions and provides various utility functions. Wrappers allowing the use of the OpenSSL library in a variety of computer languages are available.  The OpenSSL Software Foundation (OSF) represents the OpenSSL project in most legal capacities including contributor license agreements, managing donations, and so on. OpenSSL Software Services (OSS) also represents the OpenSSL project, for Support Contracts.  OpenSSL is available for most Unix-like operating systems (including Linux, macOS, and BSD) and Microsoft Windows. |

|  |
| --- |
| **Implementation Details:** |
| **1. Enlist all the Steps followed and various options explored**   1. **Download and Install OpenSSL**    1. Download the Download Win64 OpenSSL v3.0.8 (EXE) from the website.      * 1. Install and update the Path variable in Environment Variables.  Generating RSA private /public key pair  1. Generate key pair using: openssl genrsa -out mykey1.key 1024 2. Extract public key from key pair using: openssl rsa -in mykey1.key - pubout -out mypublickey.key      Public Key Encryption Using the public key we can encrypt the text in plaintext.txt using: openssl pkeyutl - encrypt -in plaintext.txt -inkey mypublickey.key -pubin -out encrypted.txt     Hash Functions Create a file to be hashed.  Use the command: openssl dgst -sha256 hashfile.txt     Certificate Creation  * 1. Create a private key using:   openssl genrsa -des3 -out domain.key 2048 Enter a password     * 1. Create a certificate using the key using:   openssl req -key domain.key -new -out domain.csr Enter all the necessary information for the certificate     * 1. Create a self signed certificate used:   openssl x509 -signkey domain.key -in domain.csr -req -days 365 -out domain.crt     * 1. View the certificate using:   openssl x509 -text -noout -in domain.crt     Digital signature  * 1. Create private and public key using:   -openssl genrsa -aes128 -passout pass<phrase>: -out private.pem 4096  -openssl rsa -in private.pem -passin pass:<phrase> -pubout -out public.pem       * 1. Create a text file      * 1. Generate the signature of a file using:   -openssl dgst -sha256 -sign<private key> -out /tmp/sign.sha256<file>  -openssl base64 -in /tmp/sign.sha256 -out <signature>     * 1. Verify the signature using:   -openssl base64 -d -in <signature> -out /tmp/sign.sha256  -openssl dgst -sha256 -verify <pub-key> -signature /tmp/sign.sha256  <file> |

|  |
| --- |
| **Conclusion:** |
| Thus, in this experiment the concept of RSA algorithm for various security services like confidentiality, authentication, signature, non-repudiation and integrity was understood and applied by developing a website. |

|  |
| --- |
| **Post-Lab Questions:** |
| 2.1 In the RSA algorithm, p= 7, q=11 and e= 13, then what will be the value of d?  **Given:** p=7*p*=7, q=11*q*=11, e=13*e*=13.**Steps:**  **Compute *n*:**  n=p×q=7×11=77*n*=*p*×*q*=7×11=77  **Compute Euler’s totient ϕ(n)*ϕ*(*n*):**  *ϕ*(*n*)=(*p*−1)(*q*−1)=6×10=60  **Find d*d*:** Solve e⋅d≡1mod  ϕ(n)*e*⋅*d*≡1mod*ϕ*(*n*):13⋅d≡1mod  6013⋅*d*≡1mod60  Using the **Extended Euclidean Algorithm**, we get d=37*d*=37.  2.2 Discuss various cryptanalysis attacks possible to be carried out on RSA  **1. Factorization Attacks**   * **Principle:** Factorize n=p×q*n*=*p*×*q* to compute ϕ(n)*ϕ*(*n*) and derive d*d*. * **Methods:**   + **General Number Field Sieve (GNFS):** Fastest classical algorithm for large n*n*.   + **Shor’s Algorithm:** Quantum algorithm to factor n*n* in polynomial time, rendering RSA obsolete if quantum computers become practical. * **Mitigation:** Use n≥2048*n*≥2048-bit keys.   **2. Side-Channel Attacks**   * **Principle:** Exploit physical leaks (timing, power consumption, EM radiation) during decryption/signing. * **Examples:**   + **Timing Attacks:** Kocher’s 1995 attack measures decryption times.   + **Power Analysis:** Extract keys from power usage patterns. * **Mitigation:** Constant-time implementations and cryptographic blinding.   **3. Padding Oracle Attacks**   * **Principle:** Exploit error messages in padding validation (e.g., PKCS#1 v1.5). * **Example:** Bleichenbacher’s 1998 attack on TLS. * **Mitigation:** Use OAEP padding instead of PKCS#1 v1.5.   **4. Low Public Exponent Attacks**   * **Principle:** If e*e* is small (e.g., e=3*e*=3), recover plaintext via e*e*-th roots. * **Example:** e=3*e*=3 allows recovery of short messages without padding. * **Mitigation:** Use e=65537*e*=65537 and proper padding.   **5. Fault Attacks**   * **Principle:** Induce hardware errors during computation to leak keys. * **Example:** Glitching RSA-CRT computations to recover primes. * **Mitigation:** Validate signatures before output.   **6. Wiener’s Attack**   * **Principle:** Exploits small d*d* (d<13n1/4*d*<31​*n*1/4) via continued fractions. * **Mitigation:** Use large d*d* or balanced primes.   **7. Common Modulus Attack**   * **Principle:** If multiple users share n*n*, recover plaintext via GCD. * **Mitigation:** Unique n*n* per user.   2.3 Comment on drawbacks of RSA. Discuss solution(s) over the same.  **Drawbacks**   1. **Slow Performance**    * RSA is computationally intensive for large data (encryption/decryption is O(n3)*O*(*n*3)).    * Example: A 2048-bit RSA encrypts ~245 bytes max. 2. **Quantum Vulnerability**    * Shor’s algorithm breaks RSA on quantum computers. 3. **Key Management Overhead**    * Large keys require secure storage/distribution. 4. **Side-Channel Vulnerabilities**    * Leaks via timing/power/EM require hardware-level mitigations. 5. **Weak Parameter Choices**    * Small e*e*, close primes, or predictable RNGs weaken security.   **Solutions**   1. **Hybrid Encryption**    * Use RSA to encrypt symmetric keys (e.g., AES) for bulk data. 2. **Post-Quantum Cryptography**    * Transition to lattice-based (e.g., NTRU) or hash-based algorithms. 3. **Improved Implementations**    * Use constant-time code, OAEP padding, and strong RNGs. 4. **Key Length Updates**    * Adopt n≥3072*n*≥3072-bit keys for long-term security. 5. **Avoid RSA Misuse**    * Replace RSA with ECC (Elliptic Curve Cryptography) for faster operations. |