Final Project

Implementing Authentication and Confidentiality Services in OpenPGP

1 Tasks to be Performed

Two main services provided by OpenPGP are message authentication and confidentiality. In this project, you will implement message authentication and confidentiality services.

1.1 Authentication Service

Refer to the lecture notes, the authentication service involves the following steps (assuming Alice is the sender and Bob is the receiver):

- Generation of Authenticated Message
  - Alice creates a message digest of the outgoing message m, denoted by $h(m)$.
  - Using RSA, Alice signs the message digest $h(m)$ to produce a digital signature s of m, and the authenticated message is denoted by $m||s$ where $||$ denotes message concatenation.
- Verification of Authenticated Message
  - Upon receiving $m||s$, Bob computes the message digest of m.
  - Bob decrypts s using Alice's public key and performs the rest of verification steps.

1.2 Confidentiality Service

Refer to the lecture notes, the confidentiality service involves the following steps (assuming Alice is the sender and Bob is the receiver):

- Message Encryption
  - Alice randomly generates a session key k_s and encrypts her message m with the session key. The encrypted message is denoted by $E_{k_s}(m)$ where E is a symmetric key encryption scheme, such as AES.
  - Alice encrypts k_s using Bob's public key $(n, e)_b$. The encrypted key denoted by $E_{(n,e)_b}(k_s)$ where E is the RSA encryption function.
  - Alice sends $E_{k_s}(m)||E_{(n,e)_b}(k_s)$ to Bob
- Message Decryption
  - Upon receiving $E_{k_s}(m)||E_{(n,e)_b}(k_s)$, Bob decrypts the encrypted key with his private key to obtain k_s.
  - Using k_s, Bob decrypts the encrypted message to get m.

1.3 Implementation Requirements

Based on the previous descriptions, implement the authentication and confidentiality services of PGP:

- Programming language: C/C++ with GMP and OpenSSL libraries
- RSA: Use your own implementation from the previous project or OpenSSL
- AES: Use the appropriate functions from OpenSSL, with AES-128
- Message digest: You can adopt the "dgst" function in OpenSSL, with SHA-512

2 Expected Outcomes

When the main program is executed, here is the expected output:

1. Enter the name of the file that contains Alice's public-private key pair:
2. Enter the name of the file that contains Bob's public-private key pair:
3. Enter the name of the file that contains Alice's plaintext message:
4. Enter the output file name to store Alice's authenticated message:
5. Enter the output file name to store the verification steps performed by Bob:
6. Enter the output file name to store Alice's encrypted message:
7. Enter the output file name to store Bob's decryption of Alice's plaintext message:

2.1 Message formats

- Plaintext message: a string of characters saved in the first line of a file. An example file is provided; all should be UTF-8 Unix delimited.
- Authenticated message: the plaintext message is stored on the first line, and the signature is stored on the second line of a file.
- Encrypted message: the encrypted message is stored on the first line, and the encrypted session key is stored on the second line of a file.

3 Deliverables

- README: describe the purpose of your files and provide instructions on how to compile and execute your program.
- Well-documented source code.

3.1 Submission Instructions

Using the GnuPG software itself (https://gnupg.org/), generate a keypair for yourself, set it up with your email, and try it out with a friend. Then, encrypt and sign your email, source code, and public key to Dr. Taylor (public key provided) using one of the following methods. Note, you are not using your own program to encrypt anything, but GnuPG, which merely fyi defaults to RSA-2048/AES-128 for the latest version.

1. https://gnupg.org/software/frontends.html has a list of mail client plugins (which are more secure than web browser plugins if you are interested in using them later)
2. https://www.mailvelope.com/en is a nice browser plugin that works with most email services (gmail included).

Next, encrypt your source files using the provided public key, using either the command line GnuPG or a GUI program like KGPG, and upload the encrypted bundle to Canvas.