ΠΛΗ-591 Ασφάλεια Πληροφοριακών Συστημάτων

Ημερομηνία Παράδοσης 4/2/2017 στις 23:55 στο courses

Final Project

Part I: Building blocks

Create (or use from exercise 2) modules that allow users to encrypt files, decrypt files, hash files, sign files, verify signatures and manage digital certificates. More specifically:

- Encryption/decryption will use AES 128 in CBC mode and PKCS#5 padding
- Hashing will use SHA2-256
- Signing will use SHA2-256 to hash the file and RSA 2048 to sign it
- Digital certificates for public keys will be X.509 v3

The program will display a menu to the user listing these options. For each option, the input is as follows:

1. Encrypt: file name and 16 byte key
2. Decrypt: file name and 16 byte key
3. Hash: file name
4. Signature: the name of the file to be signed and the name of the file containing the RSA private key
5. Signature verification: the name of the signed file, the name of the file containing the signature and the name of the file containing the RSA public key
6. Certificate verification: the certificate will be verified based on expiration date and subject's name (have to be the same as the owner of the public key)

For the first 4 options, the result is written to a file.

For options 5 and 6, the program returns an indication as to whether verification was correct.

Part II

The following will familiarize you with TLS (SSL) implementations.

Create a client and a server that connect via python sockets. Only the server is authenticated (one-way authentication). After the client connects to the server, the client will prompt the user for a single line of input and send it to the server. The server will receive the text from the client and print it to the screen.

You may use self-signed certificates or may create a CA to sign the certificate (you do not need to get a certificate signed by a real CA).

Part III

Develop two processes (client/server) that initially authenticate each other with the use of digital certificates. Afterwards both processes use their key pairs (RSA-2048 bits) to exchange a symmetric key. The symmetric key is generated by the server and is securely sent to the client. Those two processes will finally start exchanging encrypted data (AES 128, CBC, PKCS#5 padding).

The client and server certificate can be signed by a Certificate Authority created manually using openSSL.

Deliverables

- Source code in a single .zip/.gz file (not tar files) that will be tested
- Report with details of your implementation.

Important Notes

1. You should structure your code in the minimum number of **sources** (e.g tools.py, partI.py, partII.py, etc). Use comments and prints for the requested outputs as described above.
2. The report should follow the **ACM small standard format**[1]. Any deviations will affect the evaluation of the report.
3. You are encouraged to use block diagrams/visualizations in the report provided that are all made with an illustration tool (not by hand).

   Code should run "out-of-the-box" (i.e in an intuitive manner) without any modification and display all the necessary output as defined in the description above.

[1] http://www.acm.org/publications/article-templates/acmsmall-word.zip