Exercises in Secure Mobile Systems

FH Hagenberg, WS 2022
FH-Prof. DI Dr. Erik Sonnleitner

Exercise 7: GPG, SSH & OpenSSL

UNIVERSITY OF APPLIED SCIENCES UPPER AUSTRIA

(1) GnuPG (GPG)

Terminology notes:

- **PGP**: *Pretty Good Privacy*, the original implementation by Phil Zimmermann, was among the first publicly available programs providing strom encryption for everyone. Doesn't exist anymore.
- **OpenPGP** Official standardization of the PGP message format, to allow interoperability between different implementations - OpenPGP is not a program by itself.
- **GPG (GnuPG)**: *GNU Privacy Guard* is the primary open-source implementation of OpenPGP and widely used. Provides a command-line client and programming library.

GnuPG Assignments

(1.1) Use the `gpg` command-line tool to create a new public/private key-pair for SMS. Depending on availability of entropy on your (virtual) system, this may take some time.

(1.4) Arbitrarily choose a data file, and cryptographically sign it with your newly created key-pair using `gpg`.

(1.5) Verify the signature on the command-line.

(1.6) Symmetrically encrypt any data file using AES-256.

(2) SSH: Prerequisites

OpenSSH is a free, open-source, massively used cryptographic command-line tool which implements an application-layer crypto-protocol. It is traditionally used for secure remote shell connections, but may transport arbitrary data streams. The OpenSSH suite ships with multiple tools including:

- **`ssh`** The SSH command-line client
- **`sshd`** The SSH daemon (server)
- **`ssh-keygen`** Tool for generating asymmetric key-pairs
- **`ssh-agent`** service holds key-pairs in memory for an entire login session.
- **`sshfs`** Maps SSH calls to a filesystem layer. With SSHFS, you can mount remote directory trees into a local mount point. The server only needs a running SSH server (often available via separate package)
- **`scp`** Secure remote file transmission tool

Notes: For exercises, you should ideally use 2 hosts (e.g. your host OS and a virtualized guest), but they can also be accomplished using only one Linux box (via localhost). The SSH service must be started before it can be used, e.g. via

- Classic SysV method: **`sudo /etc/init.d/ssh start`**
- Modern SystemD method: **`sudo systemctl start sshd`**

(2) SSH: Assignments

(2.1) **Public key authentication**: Use the OpenSSH tool `ssh-keygen` to generate a new asymmetric key-pair on the client machine, using the RSA version 2 algorithm. Assure to provide a passphrase for encrypting your private key. Once done and unless defined otherwise, the public key usually resides at `~/.ssh/id_rsa.pub`, while the private key is located at `~/.ssh/id_rsa`. Copy the public key to the server machine, and append its contents to the `authorized_keys` file on the server. Log in from client to server using public-key authentication.

- Describe the process in your protocol.
- How does public key authentication work?
- Why is it good practise to encrypt the private key?
- What actions have to be taken in order to force your server to *only* use public-key authentication, and forbid password-based authentication?

(2.2) **Proxying**: Create a transparent proxy using the `-D` option. Edit your Firefox' network configuration to use the specified port at your machine as SOCKS proxy. When surfing the net, Firefox will encrypt your data, send it to the SSH server, which in turn decrypts it and actually performs the HTTP(S) requests. Note: If you have access to a remote online SSH server, you can easily circumvent LAN firewall restrictions by binding the remote SSH server to any allowed port (e.g. 80, 443, etc).

(2) SSH: Assignments

(2.3) **Tunneling**: Create an encrypted TCP tunnel for arbitrary data transmissions using the `-L` option. On your client, bind port 1234 as local tunnel endpoint which routes incoming data to the server. Force the server to route incoming connections to a particular service (e.g. your e-mail IMAP server) and its corresponding port (e.g. 143). Try your secure connection by accessing **`localhost:1234`**, e.g. via Netcat. Provide a screenshot of the working tunnel access.

(2.4) **Key visualization**: Edit your SSH configuration file to always show visual host keys. Give a reason, why such an option exists. Should it be used?

(2.5) **SSHFS**: Mount a remote directory using **`sshfs`**. What are the advantages of using SSHFS, opposed to e.g. NFS or SMB (Microsoft local networking protocol) for network file transfers? What's the difference between SSHFS and the **`scp`** command? Note: You may need to install SSHFS on your machine.

(2.6) **Reverse tunneling**: What is *reverse tunneling*? Explain and provide an example.

(3) OpenSSL: Prerequisites

OpenSSL is a free, open-source, massively used crytographic library which consists of the command-line tool **`openssl`** and two sub-libraries:

- **`libcrypto`** Library of crypto-functions (ciphers, hashes, etc.)
- **`libssl`** Library for SSL/TLS protocol management

Implements, among other algorithms:

- Symmetric ciphers: AES, DES, 3DES, Blowfish, Camellia, etc.
- Block modes: CBC (default), CFB, ECB, OFB
- Asymmetric ciphers: RSA, DSA, DH, ECC-variants (e.g. ECDHE)
- Hash algorithms: MD2, MD4, MD5, SHA-1, SHA-256, SHA-384, SHA-512, RIPEMD 160, etc.
- Symmetric authentication: HMAC
- Certificates: X509, X509v3, SSLv3, TLSv1
- Input/output formats: asn1, bio, evp, pem, pkcs7, pkcs12

Using OpenSSL from the command-line:

- **`openssl help`** Display available commands, digests, ciphers.
- **`openssl enc help`** See help/options for **`enc`** command

(3) OpenSSL: Assignments

(3.1) Use the **`openssl`** tool to encrypt any data file of your choice with AES-256, using the OFB block mode, and ensure the result is exported with Base64 encoding (**`enc`** option).

(3.2) Generate a new 2048-bit RSA key pair with OpenSSL. The result is typically stored in a PEM-file - what information does this file contain?

(3.3) Extract the exact cryptographic math parameters of the newly created key pair (modulus, exponents, primes, coefficient).

(3.4) Encrypt the private key symmetrically (**`enc`** option).

(3.5) Export the public key to a new file.

(3.6) Create a certificate signing request (for a CA) from your previously generated keys.

(3.7) Use OpenSSL to create a HMAC for the file located at **`https://delta-xi.net/download/cat.gif`**, using the SHA-512 hashing algorithm and the key **`thiscatisnotgrumpy`**. What's the resulting MAC value?

(3) OpenSSL: Assignments

(3.8) Use OpenSSL to generate 16 random bytes, and output the result in Base64 encoding (e.g. useful for automated password generation).

(3.9) Use OpenSSL to establish a TLS connection to the HTTPS service at **`delta-xi.net`**. Which TLS protocol version and cipher is used by default? Deconstruct the cipher-string, and describe its components.

(3.10) How can OpenSSL deal with **Certificate Revocation Lists**, which keep track of invalidated certificates (e.g. stolen private key)?

(4) Minisign: A small, efficient tool for creating and verifying signatures

(4.1) Create a new keypair

(4.2) Sign any file on your system and verify its signature

(3) OpenSSL: Assignments

(5) Setting up a TLS web-server

Install a web-server on a Linux machine (preferably nginx or Apache), and register a free subdomain (e.g. at **`goip.de`**, **`freedns.afraid.org`**, or any other provider – if you already own a domain, you can use that one too).

Then create a certificate from **`https://letsencrypt.org`**. There are multiple ways to create and establish the certificate – look at the Let's Encrypt documentation and use whatever you like best. Typically, Let's Encrypt allows certificate creation using the **`certbot`** command-line tool.

Document all steps taken and submit a meaningful protocol. Configure the web-server to provide proper TLS encryption – describe the entire process, provide screenshots and command-lines.