MEIC-A
Group 08 of Alameda

Smartphone as a security token

78572, Telma Correia
telma.correia@tecnico.ulisboa.pt

79118, Filipe Magalhães
filipepgmagalhaes@tecnico.ulisboa.pt

Problem

The problem that we propose to solve consists on using a smartphone to authenticate his owner and block access to confidential files.
Nowadays smartphones are a necessity and not a luxury anymore. Everyone has a smartphone and use it while they're eating, walking, driving (don't do this, please), working and some other things ending in ing.
So we can say that if we want to authenticate someone, using the smartphone for authentication is a good idea because we will know if the person is or isn't on the surrounding area.
Regarding the context, security is needed to prevent someone(an attacker) to access confidential files if, for example the user goes somewhere for a short period of time and forgets to lock his computer or even if it is locked but a stranger tries to break the password. This is possible to achieve communicating via Bluetooth, to connect the smartphone to the computer and encrypting data.

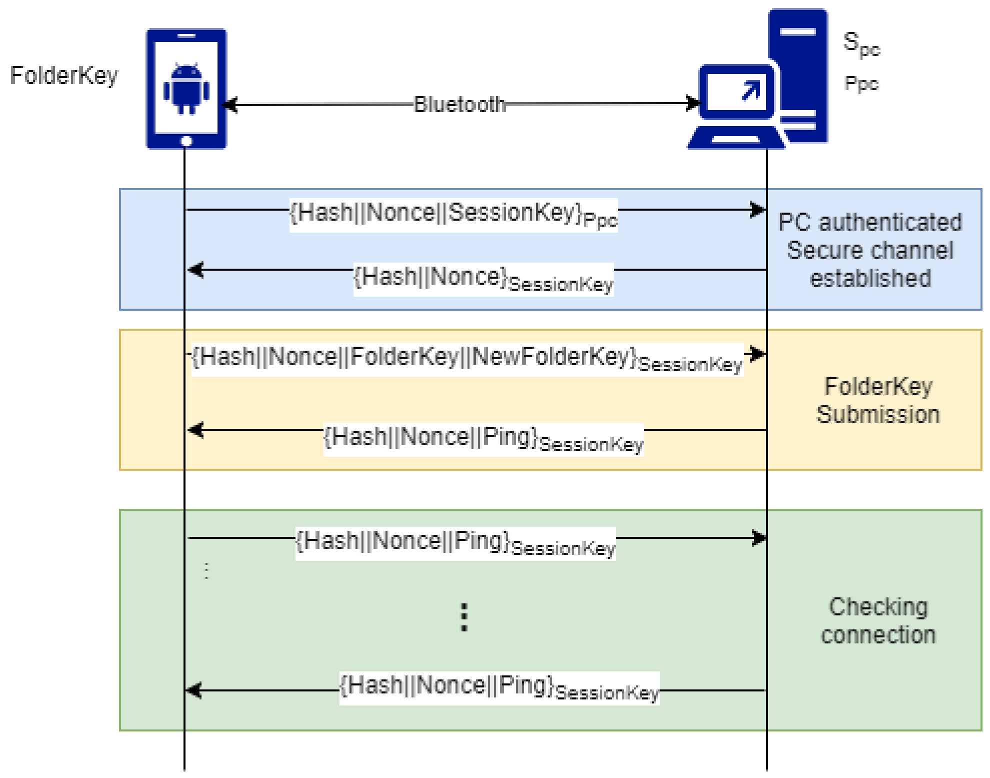
Requirements

- The user can only decrypt the files when in the physical presence of the linked smartphone (Confidentiality of the files/Authentication of the file access);
- The files should be available(decrypted) as long as the security requirements are met;
- The files should maintain integrity;
- The files should be secured(encrypted) after there is no communication with the smartphone for 10 seconds.
- Computer and smartphone must be paired before trying start a connection.

We assume:

- Only the authorized user can perform root actions on the smartphone and computer;
- A malicious user does not have access to the computer while the authorized user and his smartphone are connected;
- The computer and smartphone operating system is secure, no attacks on memory can be made and there must be no malware installed;
- The computer and the smartphone have correctly synchronized clocks, with a margin of 3 seconds
- The files on the directory are encrypted, but their names, approximate size, and structure remain in plaintext;
- Our software is installed in a secure fashion, with restricted read and write permissions: an attacker cannot modify the program nor read the private key

Solution



FolderKey
Bluetooth
S_{pc}
P_{pc}
$\{Hash||Nonce||SessionKey\}_{Ppc}$
$\{Hash||Nonce\}_{SessionKey}$
PC authenticated Secure channel established
$\{Hash||Nonce||FolderKey||NewFolderKey\}_{SessionKey}$
$\{Hash||Nonce||Ping\}_{SessionKey}$
FolderKey Submission
$\{Hash||Nonce||Ping\}_{SessionKey}$
$\{Hash||Nonce||Ping\}_{SessionKey}$
Checking connection

Figure 1: Message flow

Our solution establishes a secure channel between the smartphone and computer, through which the smartphone will send the key with which files are encrypted, followed by constant messages to ensure the connection liveness.

The asymmetric algorithm used is RSA in ECB mode and OAEP with SHA-1 and MGF1 padding. The symmetric algorithm used for messages is AES in CBC mode, with PKCS5 padding. For files, the symmetric algorithm used is AES in CBC mode, with PKCS5 padding. Files are also appended a fixed string of size 64 bytes, used to check for correct decryption of the file, and prepended with the file size and iv (different for each file). The RSA key pair is generated in the computer on installation (running *generate_key.py*)

As the server in the computer starts, a qr code with its public asymmetric key is shown, to distribute it to the smartphone if needed. The smartphone app, after having the pc public key configured will connect to the server (the in-app selected bluetooth device), with a message encrypted with it. This message contains an hash of its contents to ensure integrity, a nonce consisting of a timestamp (at the level of the second) and a sequence number starting at 0, to ensure freshness,and the symmetric session key and iv. The computer replies with an acknowledgment message ciphered with the session key, and with the same hash and nonce style used before, that will be used in every message.

With the connection established, the smartphone sends the key used to decrypt the files, along with a second key, used to encrypt the files once this session is over. With this information, the computer decrypts the protected files, and keeps in memory the folder key along with the session key and iv. Afterwards, the computer and smartphone keep on exchanging a series of ping messages, still under the same encryption, hash and nonce scheme, to ensure the connection is alive.

If the connection is interrupted, the ping messages stop flowing, or the program is ordered to close, the session is terminated, all files are encrypted (plain text versions are shredded) and the session and folder keys are deleted. The verification that the ping messages are flowing is made every 5 seconds, and the session is closed immediately if there was no ping in the last 5 seconds.These values are configurable in the program, should different conditions arise.

The asymmetric key used by the server can be changed as frequently as the protection desired by the user requires, it is just a matter of generating a new key in the computer, and updating the smartphone configuration using the qr code.

The basic version was comprised of the unsecured channel, with file encryption, the intermediate version added the secure channel, and the advanced version, all other features, including the bluetooth support.

Results

All the proposed features were implemented.

Evaluation

The time necessary to make the files available is dominated by the time to decrypt the files. Figure 2 shows that it scales linearly with the total size of the stored files, with a reasonable performance for small sizes. The use of memory and cpu by the program during the session is negligible.

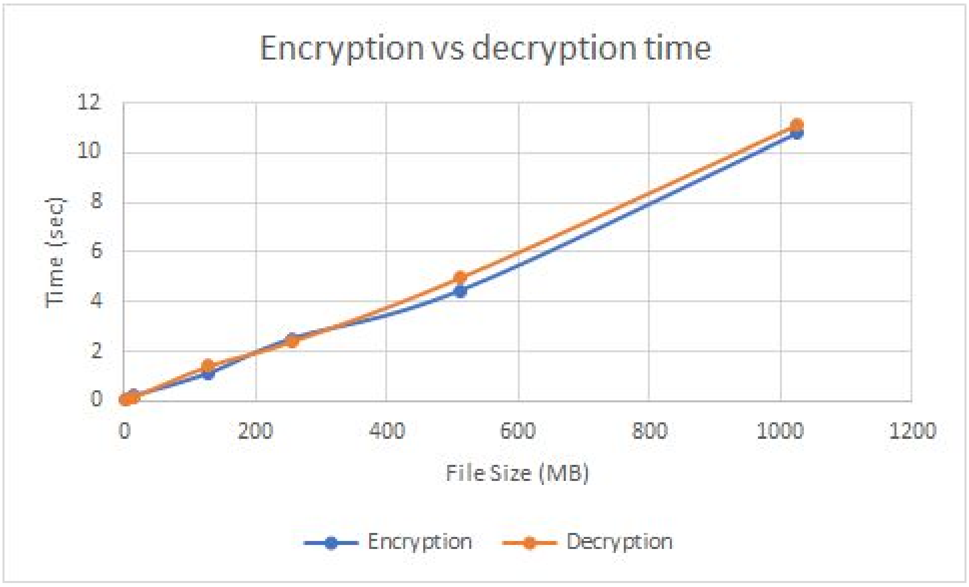


Figure 2: File Encryption Evaluation

Our solution provides satisfactory levels of security under the declared requirements, since the files are promptly encrypted as soon as the connection is detected lost. The customary replay attacks, man in the middle or eavesdropping on the communication channel are not useful against our solution. The algorithms used are the current accepted secure standard, being used in a secure fashion. The security of the files is further assured by the change of the folder key at the end of every session.

The necessary trust in the software of the devices is unfortunate, but unavoidable. The trust in the low range of the underlying connection (bluetooth) is also unfortunate, as it opens a window of attack using a repeater of the signal, or taking advantage of a larger than desirable bluetooth connection range. Our solution could be improved by timing the round trip time of a message, to detect slowdown in the connection that might indicate distance or an attack, however, it would be sensitive to regular interference at the level of the signal or cpu use in either of the machines.

Conclusion

Our solution solves the proposed scenario, providing security to sensitive files, even when the user forgets to properly secure them. The security of the files hinges on the smartphone in this solution. This allows us to offer security based on proximity, but leaves an uncomfortable situation should the smartphone become non functional or the app uninstalled. Backups of the information stored in the smartphone are an entirely new problem of security.

References

(short list of essential tools and publications with a brief description of how it was used for the project)

Computer: (Python 2.7 modules):

- PyBluez: Bluetooth communication (https://github.com/karulis/pybluez);
- pycrypto: cryptographic operations (https://github.com/dlitz/pycrypto);
- qrcode: generate qr code (https://pypi.python.org/pypi/qrcode);
- Pillow: display qr code (https://github.com/python-pillow/Pillow);

Smartphone (developed for Android 5.0):

- Default Java 7 libraries: generate and handle keys;
- Default Android libraries: bluetooth communication;
- com.google.android.gms.vision: qr code recognition (https://developers.google.com/android/reference/com/google/android/gms/vision/package-summary);
- javax.crypto : encrypt/decrypt messages(https://developer.android.com/reference/javax/crypto/package-summary.html);