DIMA PROJECT OFFICIAL DOCUMENTATION

"Watchdog"

Claudio Rizzo
800471

Emanuele Uliana
799256

4/7/2014

Teacher: Prof. Luciano Baresi

Version 1.0

Contents

1 Project context and purpose

1.1 Context

1.2 Purpose

2 Project planning

2.1 Time schedule

3 Requirements analysis

3.1 Actors

3.2 Functional requirements

3.2.1 Mobile phones association

3.2.2 Mobile phone remote localization

3.2.3 Mobile phone remote mark

3.2.4 Mobile phone remote alarm triggering

3.3 Non-functional requirements

3.3.1 Privacy and security: problems and solutions

The remote control of a cellphone is a critical activity which has many security and pivacy requirements: the next paragraphs show them briefly: for in-depth explainations see the design section (4.3).

Sender authentication

Letting alone for a while the authentication of the sender (telephone), it is fundamental that the sender (person) is trusted (if different from the owner of the target cellphone itself); that is the reason for a password based authentication: in the initialization wizard the user is required to insert a password which will be needed to send a message to that telephone (the basic assumption is that the password is known only by the mobile owner and by some people, possibly nobody, he trusts). The password is stored in the application preferences hashed along with the hashing salt (a random token) to avoid both time-to-memory attacks (such as rainbow tables) and the equality of two hashes generated from two equal passwords. The salt will be sent to any authenticated telephone (a mobile which is allowed to send command messages to the original one).

Message integrity/authentication/non forgeability/non repudiation

The command messages have some specific security requirements (plus confidentiality which is explained in the netx paragraph:

Integrity

The message received must be exactly the one sent: every transmission error or tampering must be detected and cause the abort of the current message session: no retransmission is done.

Authentication
The receiver must have a secure way to understand which telephone the received message comes from.

Non forgeability
Nobody should be able to forge a command message which is both valid and correctly authenticated.

Non repudiation
The sender must not be able to deny he sent a specific message (if he actually did it).

Digitally signing every command message can ensure integrity, authentication, non repudiation and a weak defense against non forgeability: symmetric encryption (and in particular AES-256 in GCM mode of operation) is needed for full protection.

Message confidentiality
No one should be able to detect that and which command is sent to a mobile phone, so the command message is encrypted with the symmetric cipher AES-256 in GCM mode of operation (used for performance reasons and for a supplementary integrity check).

Asymmetric keys management
Digital signatures (and shared secrets computation as we will see) require asymmetric cryptography: in the initialization wizard the application generates and stores in the preferences a key pair based on the elliptic curves; the reasons for this choice are performances and the smaller key length with respect to other keys (like RSA and DSA ones) at a fixed level of security. This makes the 140 characters (bytes) Android limit for a single sms no more a problem.

Symmetric key/initialization vector management
AES-256, being a symmetric cipher, encrypts and decrypts a specific message with the same key, and, given that the communication channel is not secure, the two parts must agree on the same key in some way; in particular ECDH is used to compute a common secret once and for all, then, when in need to send a message, the sender picks up a random 32 bytes salt, forwards it to the receiver, then both parts use a keyschedule algorithm (PBKDF2 with HMAC-SHA-256) to derive the same key starting from the secret and the salt. Furthermore the GCM mode of operation requires for every message the sender to generate a 12 bytes random initialization vector and to send it to the receiver.

Public keys mutual authentication
While dealing with asymmetric cryptography, the main problem is to bind a public key with a real user to avoid active Man-In-The-Middle (MITM from now on) attacks. Neither a Pulic Key infrastructure (PKI) or a Web Of Trust (WOT) is employed because they are both potentially insecure for various reasons (in the PKI case the presence of a trusted element, a certification autorithy hierarchy, which may be compromised/untrusted/fake; in the WOT case the presence of a net of trusted elements, the ones who signed a specific public key, which might be fake/bad persons; furthermore a key with no signatures is not automatically a fake one, but there isn't a way to tell), so the application uses a modified version of the Socialist Millionaire Protocol (SMP) to authenticate to each one each other key; this requires the two parts to have a common secret (an answer to a particular question set up on the fly by the users), which is easy to achieve, since the two users are likely to be the same person or two people who trust themselves.

3.3.2 Human friendly interface and transparency

3.3.3 Performance

3.4 Use cases

3.4.1 Initialization wizard

3.4.2 Mobile phones association

3.4.3 Remote control: localization

3.4.4 Remote control: mark stolen/lost/both/found

3.4.5 Remote control: alarm triggering/untriggering

4 Design

4.1 Application Architecture

4.2 Design Patterns

4.3 Crypto protocols and alogrithms

4.3.1 Elliptic Curves key pair generation

4.3.2 Socialist Millionaire Protocol

Public key request

Public key sending

Question sending

Hash sending

Ack and password salt sending

Second half

Error management

4.3.3 Elliptic Curves Diffie Hellman key exchange

4.3.4 Command Protocol

First message

Second message

Third message

Fourth message

Error management

Timeout management

4.3.5 Elliptic Curves Digital Signature Algorithm

4.3.6 AES256GCM

5 Testing

5.1 Crypto testing

5.2 Protocol testing

6 Installation and usage manual

6.1 Installation

6.2 Usage

6.2.1 Initialization wizard

6.2.2 Change application settings

6.2.3 Send a command message