Cutia

Paul Feuvraux

July 11, 2017

1 Introduction

This paper aims to explain Cutia, a simple protocol comprehensive by everyone.

2 Motivations

I thought about this protocol because I was bored to see that all encrypted messaging protocols are always using SRP, HMAC, DH, and other things that aren't easy to understand for people who would like to design an encrypted messaging service. So I thought designing a protocol with AES and RSA only would be funny to do.

3 Terms

- **Session**: Generic term used to talk about either a Conversation Session or a Group session.
- **Conversation Session** (CS): A conversation session between two users.
- **Group Session** (GS): A conversation session between more than users.
- **Key Agreement key** (KAK): 4096-bit asymmetric pair of RSA keys generated at the client-side.
- **Private Key** (PKA): Used to decrypt the SK of a CS. It is a part of the KAK.
- **Public Key** (PKB): Used to encrypt the SK of a CS before exchanging it. It is a part of the KAK.
- **Session Key** (SK): 256-bit symmetric key used to encrypt and decrypt messages in a Session. It is generated at the client-side.
- **Passphrase** (P): User's defined alphanumeric UTF-8 passphrase during registration on the client side.
- **Key Encryption Key** (KEK): 256-bit symmetric key derived from P at the client-side used to encrypt the user's CEK.

- **Content Encryption Key (CEK)**: 256-bit symmetric key randomly generated at the client-side which is used to encrypt the user's PKA, and every SK that he has to store.
- **Key derivation function**: Every derivation function is performed with PBKDF2. We denote this process as $KDF(x, s, i)$, where x is the passphrase, s the Salt, and i is the strengthen by factor and always equals 7000.
- **Symmetric Encryption function**: Every symmetric encryption is performed with AES under the Galois/Counter mode on 128-bit block cipher. We denote this process as $EncSym(k, x, i, t)$, where k is the symmetric key, x is the content to be encrypted, i is the initialization vector, and t is the authentication tag.
- **Symmetric Decryption function**: Every symmetric decryption is performed with AES under the Galois/Counter mode on 128-bit block cipher. We denote this process as $DecSym(k, x, i, t$, where k is the symmetric key, x is the encrypted content to be decrypted, is is the initialization vector, and t is the authentication tag.
- **Asymmetric Encryption function**: Every asymmetric encryption is performed with RSA. We denote this process as $EncAsym(k, x)$, where k is a public key and x is the content to be encrypted.
- **Asymmetric Decryption function**: Every asymmetric decryption is performed with RSA. We denote this process as $DecAsym(k, x)$, where k is a private key and x is the content to be encrypted.

4 User

4.1 Registration

To register, the user has to provide a Passphrase P. The user's KAK is generated while this one is registering. The PKA has to be encrypted before being sent to the server. PKB is sent to the server and stored in plain text.

4.1.1 P derivation & CEK generation

Once the user's Passphrase P defined, a random Salt is generated on 128 bits. We proceed to a key derivation to obtain the user's KEK such as $KEK = KDF(P, Salt, 7000)$. In the meantime, the CEK is generated.

4.1.2 Storage & Encryption of PKA

The user's PKA has to be encrypted before being sent to the server. A random initialization vector (IV) and authentication tag (AT) are randomly generated on 128 bits. We proceed to the encryption of PKA as $EncSym(CEK, PKA, IV, AT)$. Once encrypted, the encrypted PKA is sent to the server with the IV, AT such as $PKA = (PKA||IV||AT)$.

4.1.3 Storage & Encryption of CEK

A random IV and AT are generated on 128 bits. CEK is encrypted under the KEK such as $EncSym(KEK, CEK, IV, AT)$. Before sending CEK to the server we encapsulate the cryptographic parameters IV, AT, the encrypted CEK (eCEK) and the Salt such as $CEK = (eCEK||Salt||IV||AT)$. We're now able to send the encrypted CEK to the server.

4.2 Connection

Once the user authenticated in the system of the application, the client gets user's KAK and the CEK which are stored encrypted in the server. The user types his Passphrase P.

4.2.1 P derivation

From the encrypted CEK, we extract the Salt. The client proceeds to the derivation of the user's Passphrase P such as $KEK = KDF(P, Salt, 7000)$.

4.2.2 CEK decryption

The client proceeds to a symmetric decryption function to obtain the decrypted CEK such as $CEK = DecSym(KEK, CEK, IV, AT)$.

4.2.3 PKA decryption

From the KAK the client gets the PKA and decrypts it such as $DecSym(CEK, PKA, IV, AT)$.

4.3 Change the user's P

The user might need to modify his Passphrase P.

4.3.1 Derivation of P

The decrypted CEK is needed. The user edits his Passphrase P. Once modified, the client proceeds to a derivation of P to obtain the KEK. From the CEK we get the Salt. Now the client is able to proceed to the derivation of P such as $KEK = KDF(P, Salt, 7000)$.

4.3.2 Encryption of CEK

A random initialization vector and authentication tag are generated, both on 128 bits. The client proceeds to the CEK encryption such as $EncSym(KEK, CEK, IV, AT)$. The client encapsulates the CEK and its cryptographic parameters such as $CEK = (CEK||Salt||IV||AT)$ and sends the packet to the server.

5 Paranoiac mode

This mode is a CS that doesn't support more than one device of every user and doesn't support any history. SK is stored in the device, but isn't sent to the server.

5.0.1 SK exchange

SK is generated by one of the two participants (users) of the CS. The user's client that generated SK gets the other user's PKB and encrypt SK under PKB such as $EncAsym(PKB, SK)$. Once SK encrypted, it is sent to the other user.

5.0.2 Message Encryption

For every message are generated an initialization vector (IV) and an authentication tag (AT), both are generated on 128 bits. Every message are encrypted by the Symmetric encryption function such as $EncSym(SK, message, Iv, AT)$. Once the message encrypted, the client encapsulates the encrypted message EM and the cryptographic parameters such as $m = (EM||IV||AT)$.

5.0.3 Message Decryption

The client gets the newly arrived message. From m it extracts the encrypted message EM, the IV and the AT. The client proceeds to a symmetric decryption to get the plain text message M such as $M = DecSym(SK, EM, IV, AT)$.

6 Basic mode

This mode is a CS that supports keys history. This is useful to provide a multi-device messaging service.

6.1 Structure of the history

For every SK an ID is generated, and every message encrypted under a certain SK is composed of the ID of the SK. Then the history is composed of an or several encrypted SK and their ID.

6.2 SK exchange

SK and its ID are generated by one of the two participants (users) of the CS. The user's client that generated SK gets the other user's PKB and encrypt SK under PKB such as $EncAsym(PKB, SK)$. Once SK encrypted, it is sent to the other user such as $SK = (SK||ID)$.

6.3 SK encryption

We generate a random initialization vector (IV) and a random authentication tag (AT), both on 128 bits. Every user's client encrypt the SK under their CEK such as $EncSym(CEK, SK, IV, AT)$. Once encrypted, they encapsulate the SK and its parameters such as $SK = (SK||ID||IV||AT)$ and they store this packet in the server.

6.4 Entering a CS

The client gets the encrypted SK as a packet composed of the parameters of the SK. It extracts the IV, AT, and its ID. The client proceeds to a decryption of SK such as $DecSym(CEK, SK, IV, AT)$ and decrypt the most recent encrypted messages which have been encrypted under the decrypted SK.

6.5 Regenerating SK

One of the two participants (users) generates a new SK and its ID. The user who generated the new SK gets the other user's PKB and encrypt the SK with the Asymmetric encryption function such as $EncAsym(PKB, SK)$ and encapsulate the encrypted SK and its ID such as $SK = (SK||ID)$.

6.6 Message encryption

For every message are generated an initialization vector (IV) and authentication tag (AT), both on 128 bits. The plain text message m is encrypted under SK by the asymmetric encryption function such as $em = EncSym(SK, m, IV, AT)$, where em is the encrypted message. Once encrypted, em is encapsulated with its parameters such as $em = (em||IV||AT||ID)$.

6.7 Message decryption

The client extracts the cryptographic parameters from the message itself. The client decrypts the encrypted message em under SK such as $m = DecSym(SK, em, IV, AT)$.

6.8 Browsing older messages

The Id of SK is not used to authenticate messages since we're using the Galois/Counter mode which is self authenticated. But the ID of the SK is used to browse older messages which were encrypted under another SK.

6.8.1 Getting the appropriate ID

Every message is composed of an ID which corresponds with a SK stored in the server. The client reads the ID of the message and gets the corresponding SK to decrypt it.

7 Group mode

This mode is a GS.

7.1 SK exchange

One of the participants (users) generates a new SK and its ID. The user also gets the public keys of every participant and encrypts the message as many times as there are pubic keys (PK) such as $EncAsym(PK, SK)$ and encapsulate SK and its ID as a packet *packet* such as $packet = (SK||ID)$. Once this done, the user sends *packet* to every participant.

7.2 Joining a group

The one's client who invited the user gets the PKB of this user and encrypt the SK of the group such as $EncAsym(PKB, SK)$. Once encrypted, SK is put in a packet $packet$ such as $packet = (SK||ID)$.