VEGA MESSENGER
SECURITY WHITEPAPER

Version 1.01

September 2017

Content

Overview

VEGA Messenger is developed by Accelior S.A., a privately held Luxembourg based technology company.

With strong focus on security and privacy, VEGA's most important objective is to provide the most secure and reliable instant messaging experience to its users.

For this purpose, all information exchanged by the participants during a chat session (text, photo, video, etc.) is by default completely End- to -End Encrypted without the user's intervention.

Due to the nature of this implementation design, VEGA engineers and any other third party cannot decrypt and access any of these messages ensuring privacy for its users. In this paper, the designs and algorithms related about the cryptography implementation are detailed.

App distribution

VEGA Messenger application is available for the most popular smartphone platforms available today: Android and iOS.

Users can receive the VEGA Messenger app for enterprise in different ways.

1. Via Enterprise Mobile Device Management solution (MDM): For many in IT, the ability to secure smartphones and tablets is the primary reason for investing in MDM. IT administrators can centrally enforce security policies on all mobile devices supported by the software, controlling settings such as password restrictions, data encryption and feature selection. For example, IT can require that all corporate data be encrypted and cameras be disabled on users' mobile devices. MDM can block unauthorized apps from being installed on a device and can detect if a device has been jailbroken or rooted. Jailbreaking an iOS device overrides the operating system's limitations on the types of applications, extensions and themes that can be installed on the system. Rooting an Android device permits privileged control over the Android subsystem. In both cases, the device can become seriously compromised and more vulnerable to malware. Some malware even relies on a device being jailbroken or rooted in order to inflict real damage.

2. Via Apple Enterprise distribution program: Distribute proprietary, in-house iOS apps within your organization. Securely host and deploy apps to your employees' iOS devices.

3. Via Google Play for Enterprise: Android’s built-in management features enable IT admins to fully manage devices used exclusively for work. For BYOD and corporate-owned devices used for both work and personal purposes, admins can create and manage a separate work profile. Apps in managed Google Play are installed in the work profile, giving admins full control over the app and its data. Any apps or data outside the work profile remain private to the user.

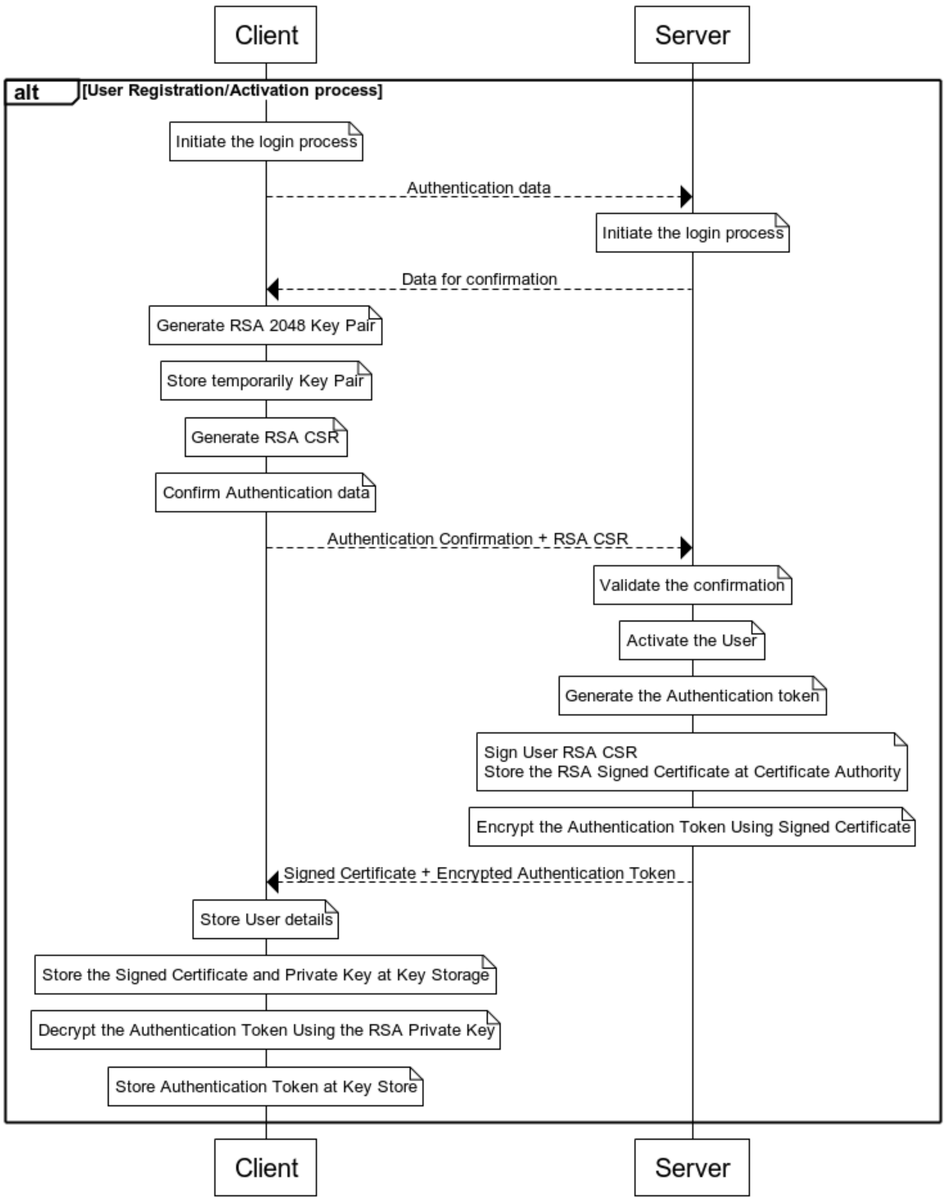
Transport Protocol

VEGA communicates over TCP/IP connection using OpenSSL/TLS 1.2. The mobile apps are secured against SSL Man in the middle attack (MITM) with root certificate pinning. Pinning is an effective way to remove the "conference of trust". VEGA Messenger app pins the certificate (or public key) and no longer needs to depend on others - such as DNS or CAs when making security decisions relating to a peer's identity.

Registration process

Each time new user is trying to login on a new device, a registration process is being performed. During this process:

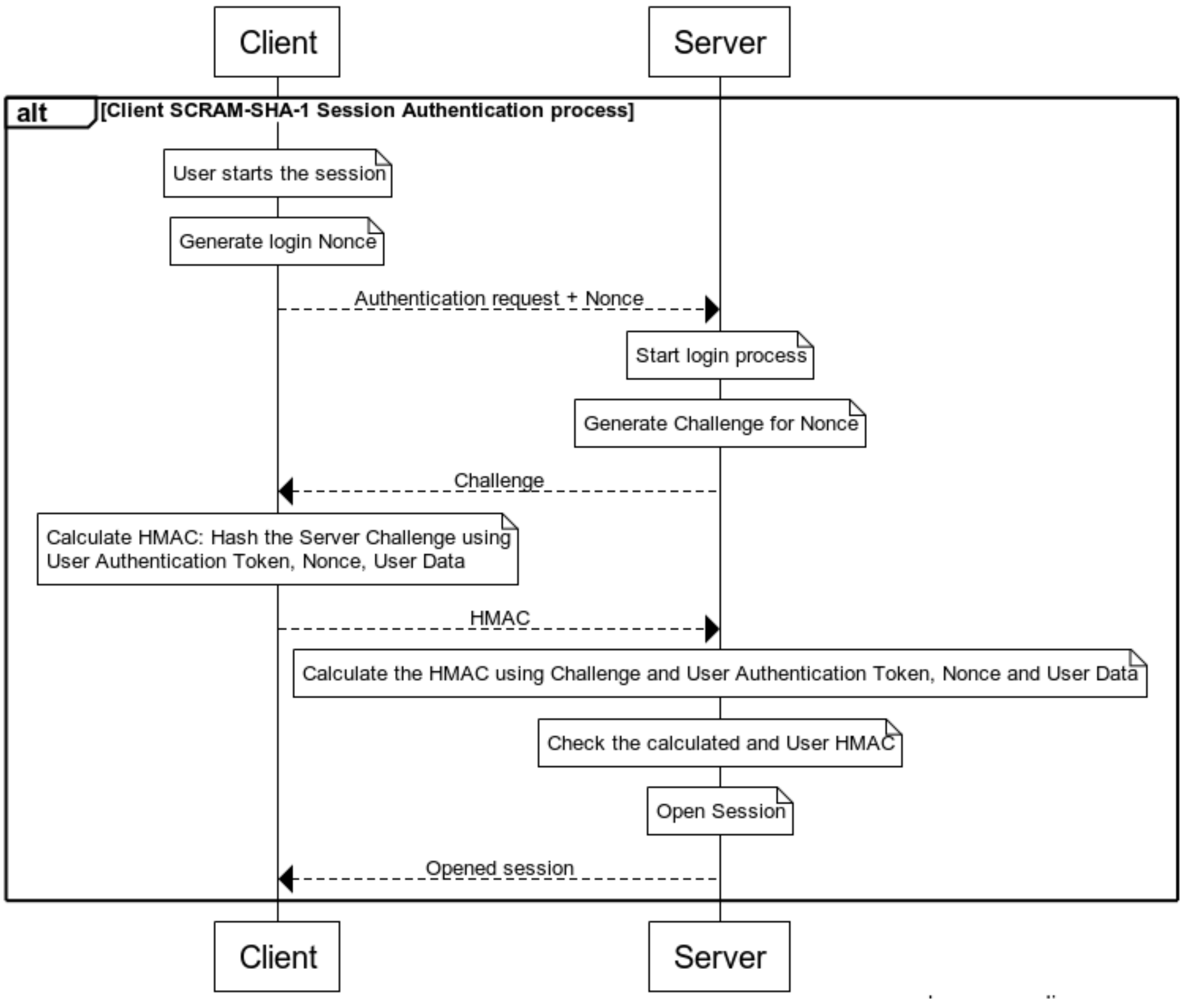
1. The user confirms his login credentials
2. An RSA key pair is generated
3. RSA User Certificate is being signed and stored by the Certificate Authority (Server)
4. Server generates a User Authentication Token, encrypted using user's public key and sent to the user which can subsequently decrypt it with his private key.


Client
Server
alt
[User Registration/Activation process]
Initiate the login process
Authentication data
Initiate the login process
Data for confirmation
Generate RSA 2048 Key Pair
Store temporarily Key Pair
Generate RSA CSR
Confirm Authentication data
Authentication Confirmation + RSA CSR
Validate the confirmation
Activate the User
Generate the Authentication token
Sign User RSA CSR
Store the RSA Signed Certificate at Certificate Authority
Encrypt the Authentication Token Using Signed Certificate
Signed Certificate + Encrypted Authentication Token
Store User details
Store the Signed Certificate and Private Key at Key Storage
Decrypt the Authentication Token Using the RSA Private Key
Store Authentication Token at Key Store
Client
Server

Authentication Process

VEGA client is initiating the Simple Authentication and Security Layer (SASL) mechanism using SCRAM-SHA1 (Salted Challenge Response Authentication Mechanism) algorithm to authenticate user sessions.

The client generates the Nonce number and sends request to the server to start the authentication. Server generates for this request a Challenge, which is passed to the client. Client calculates the Hash of the Nonce, User Id, Challenge with the User Authentication Token and sends it to the server, without sending the Authentication Token. Server can calculate the same Hash having the same parameters and can validate the received hashes. Client never send the authentication token. No replay attack are possible (by virtue of Nonce and challenge mechanisms). Server generated tokens are used versus passwords implementation.


Client
Server
alt
[Client SCRAM-SHA-1 Session Authentication process]
User starts the session
Generate login Nonce
Authentication request + Nonce
Start login process
Generate Challenge for Nonce
Challenge
Calculate HMAC: Hash the Server Challenge using User Authentication Token, Nonce, User Data
HMAC
Calculate the HMAC using Challenge and User Authentication Token, Nonce and User Data
Check the calculated and User HMAC
Open Session
Opened session
Client
Server

RSA keys exchange

Server is storing client signed RSA public keys. To establish a first Client to Client communication a request for the recipient's public key is sent allowing proper authentication. The server behaves as a CA (certification authority) in a PKI system, i.e. store public keys and distribute them if someone asks.
If a user adds his friend to a contact list a request is then sent to the server for the public key of his friend, then check (public key and user identity should be signed with his/her private key) and store it on his device. Private keys are safely kept on the client device and are never sent. Private keys are used for authentication and signing.

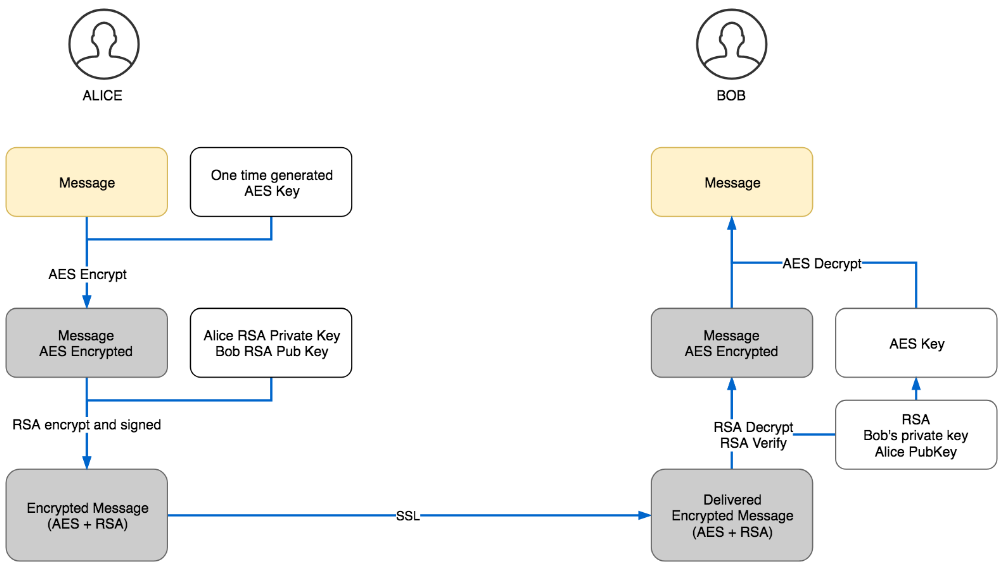
Message End-to-End Encryption

End-to-end encryption is implemented via two modes. The first mode (Pubkey secret mode) is used before the symmetric keys are defined between the participants (Alice and Bob). The second mode (Shared secret mode) is used when the symmetric encryption keys are exchanged.

Every messages types (text, images, videos, audio messages, location info, etc.) are encrypted.

Pubkey secret mode

Based on Random number generation, Alice will generate an AES key for the encryption. A unique AES key is used to encrypt the message and then a combination of encrypted message and key is being cyphered by recipients' RSA public key (only readable by the recipient) and signed by sender's private key (anyone can verify the signature of Alice).

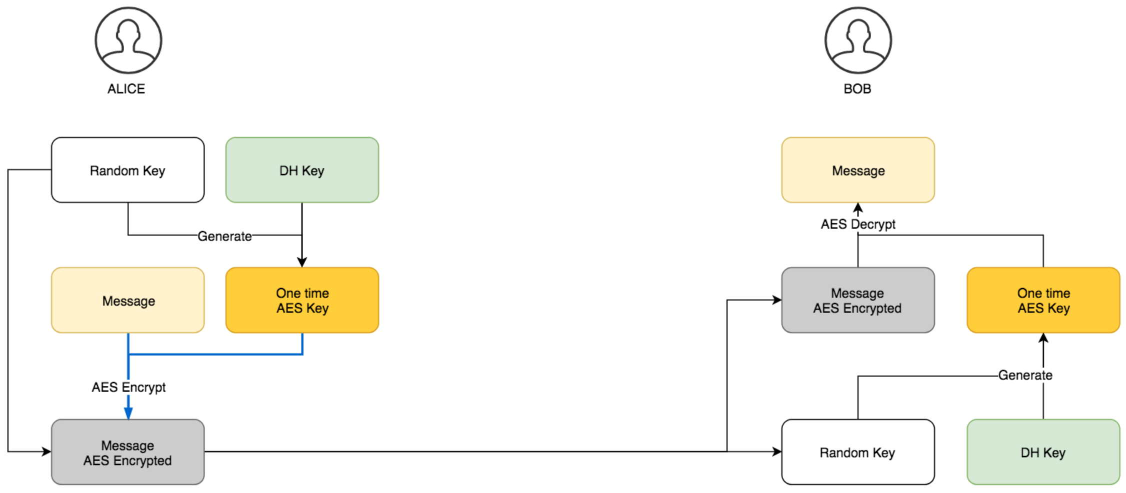

ALICE
BOB
Message
One time generated AES Key
AES Encrypt
Message AES Encrypted
Alice RSA Private Key Bob RSA Pub Key
RSA encrypt and signed
Encrypted Message (AES + RSA)
SSL
Delivered Encrypted Message (AES + RSA)
RSA Decrypt RSA Verify
RSA Bob's private key Alice PubKey
AES Key
AES Decrypt
Message

Encrypted message is being sent to Bob. Bob with the help of his private key can decrypt and validate the cyphered message and then retrieve the AES encryption key. Bob can now AES decrypt the Message payload itself and read the message.

Shared secret mode

This communication mode between Alice and Bob is occurring when a secure symmetric encryption key exchange (see Symmetric encryption key exchange) has been established.

Each message is encrypted by a unique AES key and random generated IV Key. The AES key is generated based on random number, nonce number, message data and users shared key (see Symmetric encryption key exchange).
Since Bob has a shared key and the parameters used for a specific key generation, he can generate the AES key and decrypt the message.



Symmetric encryption key exchange

In order to securely use symmetric encryption for messages Alice and Bob need to use a authenticated key exchange cryptographic method (arithmetic Diffie Hellmann) to generate common shared set of keys.

- Alice generates (randomly) her secret K_A and computes $g^{K}{}_{A}$.
- Bob generates (randomly) his secret K_B and computes $g^{K}{}_{B}$.
- Alice sends $g^{K}{}_{A}$ to Bob.
- Bob sends $g^{K}{}_{B}$ to Alice.
- Alice computes $(g^{K}{}_{B})^{K}{}_{A}$, using her secret K_A and the value sent by Bob.
- Bob computes $(g^{K}{}_{A})^{K}{}_{B}$, using his secret K_B and the value sent by Alice.

Alice and Bob ends up with the same value, which is the shared secret that they thereafter use for AES encryption they use to communicate to each other.

Alice and Bob also own asymmetric key pairs usable for digital signatures, and they use them: Alice signs whatever she sends to Bob, and Bob verifies that signature (using Alice's public key). Similarly, Bob signs what he sends to Alice, and Alice verifies that signature (using Bob's public key).

At the end, if all verifications were successful, Alice and Bob share a value "secret" that is known only by them.

1. Shared key size is 1536 bits
2. Authenticated Key exchange process is achieved by usage of Public/Private Key Pairs of Alice and Bob
3. If one of the verifications fails the process is still proceeded with next steps and sends random data instead of real data in order to maintain constant execution and response time (against various Timing attacks).

AES encryption algorithm

For encryption AES-128 block cipher is being used (for short AES-128 or just AES).
In order to use this algorithm both parties have to know the secret key.
AES allows to encrypt a 128-bit block of a message, so in order to encrypt longer messages we will have to use a mode of operation.

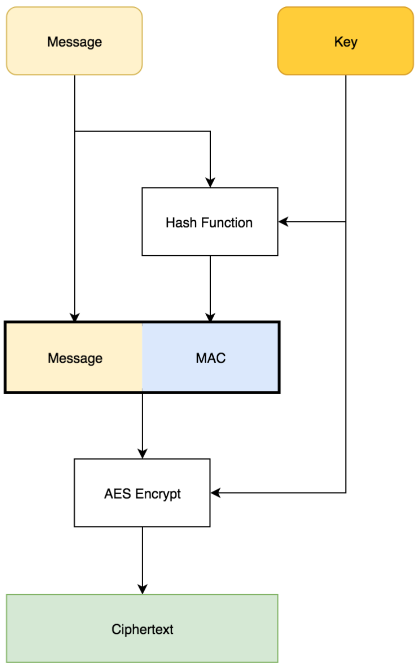
Our mode of operation of choice is Counter mode (AES-CTR). We will use 64bits Nonce (a.k.a. Initialization Vector or IV) and 64bits for the counter.
AES-CTR has many properties that make it an attractive encryption algorithm for in high-speed networking. AES-CTR uses the AES block cipher to create a stream cipher. Data is encrypted and decrypted by XORing with the key stream produced by AES encrypting sequential counter block values and AES-CTR can be pipelined and parallelized.

VEGA prefers to use AES-128 instead of AES-256. Briefly, there is a long-known problem with how AES deals with 256-bit AES keys. AES does multiple rounds of transforming each chunk of data, and it uses different portions of the key in these different rounds. The specification for which portions of the key get used when is called the “key schedule”. The key schedule for 256-bit keys is not as well designed as the key schedule for 128-bit keys. And in recent years there has been substantial progress in turning those design problems into potential attacks on AES 256.

Authenticated encryption

Authenticated Encryption (AE) is being used and is a form of encryption which provides confidentiality, integrity, and authenticity.

VEGA uses "authenticate-then-encrypt" method. AES encryption includes MAC calculation and verification. Alice computes MAC (Keccak SHA-3). Bob calculates the MAC as well and compares the received data. This excludes possibilities of Data corruption and tampering.



Random numbers for cryptography

Several places of our cryptographic framework require good cryptographically secure random numbers. Apple and Android provide cryptographically secure random number generator (RNG, see Randomization Services Reference).

Due to recent events involving NSA we cannot blindly trust this implementation, so we build security on top of this random number generator. In the implementation, we take the OS provided Random Number method and next we use user actions (time based sort of actions tracked by device sensors) as an extra randomness parameter and apply a hashing function to the provided random parameters.

Key regeneration and storage policies

Encryption keys used are stored in the device' secure key storage. Keys are being deleted when user desintall his app or during new registration process where new keys are being created and exchanged. For each key we keep a counter and a threshold value (N). When N is reached the current key is revoked and a new key is being exchanged after each N usages.

Hashing algorithms

Hashing algorithm is used widely for different applications of Hashing and MAC calculation (i.e. hashing with a secret key):

- Authenticated encryption
- Key generation
- Key exchange
- Randomization

As an implementation of this function the SHA-3 Keccak function has been selected. Official reference with specifications, by Keccak's designers: http://keccak.noekeon.org/Keccak-reference-3.0.pdf

Media Data exchange

Media files (audio, video) are encrypted using the same same principle algorithm as for message encryption. The Media file is encrypted with the unique AES key. The resulting encrypted file is uploaded via https to the server for delivery. In order to decrypt the file, one would need to know the key used for encryption. Alice sends the message with the URL to the file and key information to Bob the same way he would send an end-to-end encrypted message. Bob decrypts the message and decodes the URL of the file for download. After downloading the media file, Bob can decrypt the media to be displayed. After the file transfer is performed, file is deleted from the server.

Server Side Data Persistence

Servers store the pending delivery of messages when one of the participants has not yet received a message (i.e. participants are offline). This way we achieve the proof of delivery of all the messages. All of the messages are being end-to-end encrypted and server does not keep any information about the keys. Attackers can do nothing to the encrypted message even if they invade the databases. The same principle is applied to the Media files storage as all the shared files are being end-to-end encrypted.
After the delivery of messages, stored messages are deleted from the server.

Group Messaging

Group messages use the same end-to-end encryption algorithms for Messages as for the direct messaging. Key exchange, management and generation is a responsibility of the Group Administrators. Key exchange process is being performed end-to-end encrypted way between the Group Administrator and all other group participants. Thus, we ensure secure and encrypted way of key exchange.

Audio/Video Calls

Usually, VOIP communication between users (peers) is direct (P2P: peer -to -peer) and thus data will not transit through relay servers. In some circumstances when network communications issues can arise over i.e. international communication, VOIP data will pass by VEGA VOIP relay servers.
All VOIP data are encrypted using the following encryption methods:

- AES Encrypted signalling messages
- DTLS (Datagram Transport Layer) – Packet exchange
- SRTP – Transport Audio / Video

Temporary encryption key are generated for each VOIP call.

Push notification security

Both implementations use push notification messages for notifications when a user is offline or to deliver call signalling messages. None of the Push notifications messages contain sensitive information and are RSA-2048 bits encrypted. Upon receiving a notification, the client connects to the server and retrieves all sensitive information VIA the secure AES encrypted channel. Those mechanisms ensure notification data security, avoids the risks of message leaking through third- party systems. Android uses Google's Firebase Cloud Messaging and iOS uses Apple Push Notification Services (APNS).