REVERSE ENGINEERING WHATSAPP ENCRYPTION FOR CHAT MANIPULATION AND MORE

Roman Zaikin – Security Researcher
Oded Vanunu – Head of Products and Vulnerability Research
WhatsApp as evidence in court

Social media in court: your tweets could be used as evidence against you

June 22, 2018 9:22pm AEST

As we increasingly use social media platforms such as Facebook, Twitter, Instagram and WhatsApp to communicate with each other, many of us are unaware of the ways in which our posts might later resurface – and get us into trouble with the law.
Burned to death because of a rumour on WhatsApp

By Marcos Martinez
BBC Monitoring
The Communication Flow

WhatsApp Cloud
WhatsApp Behind the Scenes

**Encryption:** Open Whisper System -> Signal -> WhatsApp

**Communication:** WebSocket -> protobuf2 -> JSON
**ENCRIPTION:  Open Whisper System -> Signal -> WhatsApp**

On November 18, 2014, Open Whisper Systems announced a partnership with WhatsApp to provide **end-to-end encryption** by incorporating the Signal Protocol into each WhatsApp client platform.

On April 5, 2016, WhatsApp and Open Whisper Systems announced that they had finished adding end-to-end encryption to "every form of communication" on WhatsApp, and that users could now verify each other's keys.
COMMUNICATION: WebSocket -> protobuf2 -> JSON

The WebSocket API is an advanced technology that makes it possible to open a two-way interactive communication session between the user's browser and a server without having to poll the server for a reply.

The protobuf is a method of serializing structured data. It is useful in developing programs to communicate with each other – think XML, but smaller, faster, and simpler.

JSON is a JSON 😊
Is someone can decrypt the traffic?

End-to-end encryption

Privacy and security is in our DNA, which is why we have end-to-end encryption. When end-to-end encrypted, your messages, photos, videos, voice messages, documents, status updates and calls are secured from falling into the wrong hands.

WhatsApp end-to-end encryption ensures only you and the person you’re communicating with can read what’s sent, and nobody in between, not even WhatsApp. Your messages are secured with locks, and only the recipient and you have the special keys needed to unlock and read your messages. For added protection, every message you send has an unique lock and key. All of this happens automatically. No need to turn on settings or set up special secret chats to secure your messages.
WhatsApp Reversing Process

Before generating the QR code, WhatsApp Web generates a Public and Private Key that is used for encryption and decryption Process.
These keys were created by using curve25519_donna by using random 32 bytes.

In cryptography, Curve25519 is an elliptic curve offering 128 bits of security and designed for use with the elliptic curve Diffie–Hellman (ECDH) key agreement scheme. It is one of the fastest ECC curves and is not covered by any known patents.
To decrypt the data we will start to create a decryption code. This will take the private key from WhatsApp Web instead of the random bytes because we need to have the same keys in order to decrypt the data:

```python
self.private_key = curve25519.Private("".join([chr(x) for x in priv_key_list]))
self.public_key = self.private_key.get_public()
```
WhatsApp Reversing Process

Then, after scanning the QR code with the phone we have to take the generated secret:
WhatsApp Reversing Process – Shared Secret

self.secret

0-31

32-63

64-143

self.shared_secret

secret_hmac

data

HKDF

self.shared_secret_ex

0-31

32-63

64-79

aes_key 1

check_hmac

data

key_decrypted

0-31

32-64

have to be equal

self.aes_key

self.hmac_key
Then we have 2 interesting functions:

- **setSharedSecret** – This function divides the secret into slices and configure the shared secret.
- **E.SharedSecret** – This function uses two parameters which were the first 32 bytes and the private key from the QR generation:

```javascript
setSharedSecret: s["default"].wrapSync(function(e) {
  var t =etroit("default").decodeToJsArray(e), n = Array(32), a = n.slice(0, 32), i = h.get().keyPair.privKey, j = ArrayBuffer(32)
  return E.sharedSecret(n, i).then(function(e) {
    return v["default"].extractAndExpand(e, ",", 80)
  }).then(function(e) {
    // Code continues
  });
```
WhatsApp Reversing Process – Shared Secret

Next we have the expanded shared secret which is 80 bytes:

```
return E.sharedSecret(n, i).then(function(e) {
  e = ArrayBuffer(32) {
    return v["default"].extractAndExpand(e, "", 80)
  });
}).then(function(e) {
  var i = new Uint8Array(e,0,32)
  , r = new Uint8Array(e,32,32)
  , o = new Uint8Array(e,64,16)
  , s = new Uint8Array(n.concat(a));
```

By diving in we can see that the function uses the HKDF, is a simple hmac key derivation function (KDF) function.
WhatsApp Reversing Process – Shared Secret

- self.secret
  - 0-31
  - 32-63
  - 64-143

- self.shared_secret
- secret_hmac
- data

- self.shared_secret_ex
  - 0-31
  - 32-63
  - 64-79

- aes_key 1
- check_hmac
- data

- have to be equal

- key_decrypted
  - 0-31
  - 32-64

- self.aes_key
- self.hmac_key
We next have the hmac validation function which takes the expanded data as parameter ‘e’ and divides it into 3 parameters:

i – The first 32 bytes of shared expended is the **aes key**

r – The next 32 bytes is the **hmac**

o – The last 16 bytes is the **aes data part**

```javascript
40310  "return E, sharedSecret(n, i). then(function(e) { e = ArrayBuffer(32) }
40311  "return v["default"], extractAndExpand(e, "", 80)
40312  ".then(function(e) { e = ArrayBuffer(88) ()
40313  var i = new Uint8Array(e, 0, 32) i = Uint8Array(32)
40314  , r = new Uint8Array(e, 32, 32) r = Uint8Array(32)
40315  , o = new Uint8Array(e, 64, 16) o = Uint8Array(16)
40316  , s = new Uint8Array(n, .concate());
40317  "return new C.HmacSha256(r).sign(s).then(function(e) {
40318  var n = m["default"].encode(e)
40319  , r = m["default"].encode(els.slice(32, 64));
40320  if (r !== n)
40321  "return void 1["default"] .error("Map:saveSharedSecret hmac mismatch");
40322  var s = N["default"].(build(o, new Uint8Array(a)).readByteA{y());
40323  return (0, 
```
WhatsApp Reversing Process – hmac_sha256

Diagram showing the process with self.secret, self.shared_secret, secret_hmac, data, and key_decrypted nodes connected with arrows indicating the flow and dependencies.
Then the function HmacSha256 will be called with the parameter ‘r’ and it will sign the data with the parameter ‘s’, after that ‘n’ will receive the hmac verifier which will be compared to ‘r’ (the hmac from extended shared secret).

In python it will look like this:

```python
check_hmac = HmacSha256(shared_expended[32:64], self.secret[:32] + self.secret[64:]
if check_hmac != self.secret[32:64]:
    raise ValueError("hmac mismatch")
```
WhatsApp Reversing Process – hmac_sha256
The last encryption related function in this block is ‘aesCbcDecrypt’ which uses two parameters:

- **s** – which is a concatenation between the **last 16 bytes of the expanded shared secret** and the **data from byte 64 of the secret**.
- **i** – which is the **aes key**.
WhatsApp Reversing Process – AES Keys

- self.secret
  - 0-31
  - 32-63
  - 64-143
- self.shared_secret
- secret_hmac
- data2
- HKDF
- self.shared_secret_ex
  - 0-31
  - 32-63
  - 64-79
- aes_key1
- check_hmac
- data1
- key_decrypted
- have to be equal
- self.aes_key
- self.hmac_key

0-31
32-64
WhatsApp Reversing Process – AES Keys

This way we will get the AES Key ‘t’ and HMAC Key ‘n’
WhatsApp Reversing Process – AES Keys

self.secret

0-31

self.shared_secret

32-63

secret_hmac

64-143

data

HKDF

self.shared_secret_ex

0-31

32-63

64-79

aes_key 1

check_hmac

data

key_decrypted

0-31

32-64

have to be equal

self aes_key

self hmac key
WhatsApp Reversing Process – Code

```python
self.secret = None
self.private_key = None
self.public_key = None
self.shared_secret = None
self.shared_secret_ex = None
self.aes_key = None

self.private_key = curve25519.Private("".join([chr(x) for x in priv_key_list]))
self.public_key = self.private_key.get_public()

assert (self.public_key.serialize() == "".join([chr(x) for x in pub_key_list]))

self.secret = base64.b64decode(ref_dict["secret"])
self.shared_secret = self.private_key.get_shared_key(curve25519.Public(self.secret[:32]), lambda key: key)
self.shared_secret_ex = HKDF(self.shared_secret, 80)

cHECK_HMAC = hmac_sha256(self.shared_secret_ex[32:64], self.secret[:32] + self.secret[64:])
if CHECK_HMAC != self.secret[32:64]:
    raise ValueError("hmac mismatch")

key_decrypted = aes_decrypt(self.shared_secret_ex[:32], self.shared_secret_ex[64:] + self.secret[64:])
self.aes_key = key_decrypted[:32]
self.mac_key = key_decrypted[32:64]
```
WhatsApp Reversing Process – protobuf data

By using the keys we can decrypt any incoming message, the decryption result will be the protobuf message.

```python
self.decrypted_content = AESDecrypt(self.conn_data['aesKey'], content[32:])
print self.decrypted_content
self.decrypted_serialized_content = whatsapp_read(self.decrypted_content, True)
```

WhatsAppWebClient > decryptIncomingMessage()
In order to deserialize the protobuf we have to create our mapping, based on whatsapp protobuf that can be found in the file app:

```javascript
exports = {
  WebMessageInfo: a,
  PaymentInfo: b,
  WebNotificationsInfo: c,
  NotificationMessageInfo: d,
  TabletNotificationsInfo: e,
  WebFeatures: u

  p(u, {
    labelsDisplay: [1, E, u.FLAG],
    voipIndividualOutgoing: [2, E, u.FLAG],
    groupsV3: [3, E, u.FLAG],
    groupsV3Create: [4, E, u.FLAG],
    changeNumberV2: [5, E, u.FLAG],
    queryStatusV3Thumbnail: [6, E, u.FLAG],
    liveLocations: [7, E, u.FLAG],
    queryVname: [8, E, u.FLAG],
    voipIndividualIncoming: [9, E, u.FLAG],
    quickRepliesQuery: [10, E, u.FLAG],
    payments: [11, E, u.FLAG],
    stickerPackQuery: [12, E, u.FLAG],
    liveLocationsFinal: [13, E, u.FLAG]
  })
};
```
This is a part of our protobuf file:

```protobuf
message WebMessageInfo {
  optional MessageKey key = 1;
  optional Message message = 2;
  optional uint64 messageTimestamp = 3;
  optional STATUS status = 4;
  optional string participant = 5;
  optional bool ignore = 6;
  optional bool starred = 7;
  optional bool broadcast = 8;
  optional string pushName = 9;
  optional string mediaCiphertextSha256 = 10;
  optional bool multicast = 11;
  optional bool urlText = 12;
  optional bool urlNumber = 13;
  optional STUBTYPE messageStubType = 14;
  optional bool clearMedia = 15;
  optional string messageStubParameters = 16;
  optional uint32 duration = 17;
  optional string labels = 18;
  optional bytes paymentInfo = 19;
}
```
Burp Extension
Accessing the Keys – Burp Extension Keys

Let’s start with WhatsApp Web. Before generating the QR code, WhatsApp Web generates a Public and Private Key that is used for encryption and decryption.
Accessing the Keys – Burp Extension Secret

After the QR code is created, after scanning it with a phone, we can send the following information to WhatsApp Web over a WebSocket.
The Extension

WhatsApp Decryption and Encryption Extension by Dikla Barda, Roman Zaikin

Ref object:

Private Key:

Public Key:

Connect

Clear

Incoming Encrypt Decrypt Outgoing

Connection Status: Connected

Action Status: OK
Decrypt the **incoming data**
Decrypt the incoming data

**conversation** – This is the actual content which is sent.

**participant** – This is the participant that actually sent the content.

**fromMe** – This parameter indicates if I sent the data or someone else in the group.

**remoteJid** – This parameter indicates to which group/contact the data is sent.

**id** – The id of the data. The same id will appear in the phone databases.
Decrypt the **outgoing data**
Decrypt the **outgoing data**
Decrypt the **outgoing data**
Manipulation #1 – fake reply from someone in the group
Manipulation #2 – Fake reply to someone not in the group

Demo
Manipulation #3 – Send a private message in group chat to a specific person
Manipulation #4: send messages to myself

Demo
THANK YOU!