



AUGUST 3-8, 2019

MANDALAY BAY / LAS VEGAS

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

Willy Barton via Shutterstock

✉ Email      As we increasingly use social media platforms such as Facebook, Twitter,  
   43 Twitter      Instagram and WhatsApp to communicate with each other, many of us  
   65 Facebook      are unaware of the ways in which our posts might later resurface – and  
   in LinkedIn      get us into trouble with the law.

BBC  Sign in

News Sport Weather Shop Reel Travel More

Search



# NEWS

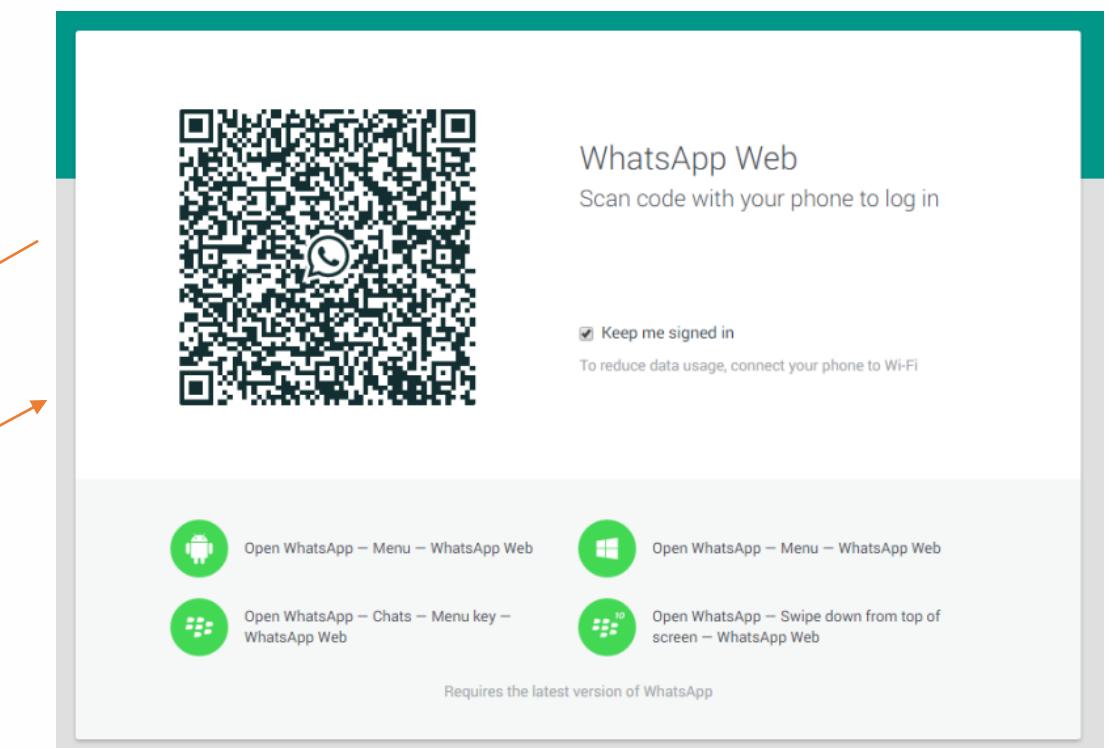
Home | Video | World | UK | Business | Tech | Science | Stories | Entertainment & Arts | Health | World News TV | More ▾

## Burned to death because of a rumour on WhatsApp

By Marcos Martínez  
BBC Monitoring



# The Communication Flow



# WhatsApp Behind the Scenes

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

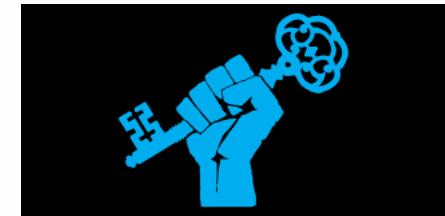


**COMMUNICATION:** WebSocket -> protobuf2 -> JSON





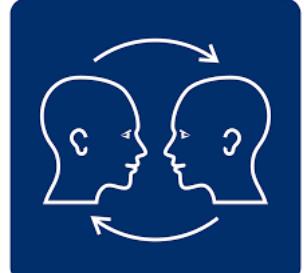
## ENCRYPTION: Open Whisper System -> Signal -> WhatsApp



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



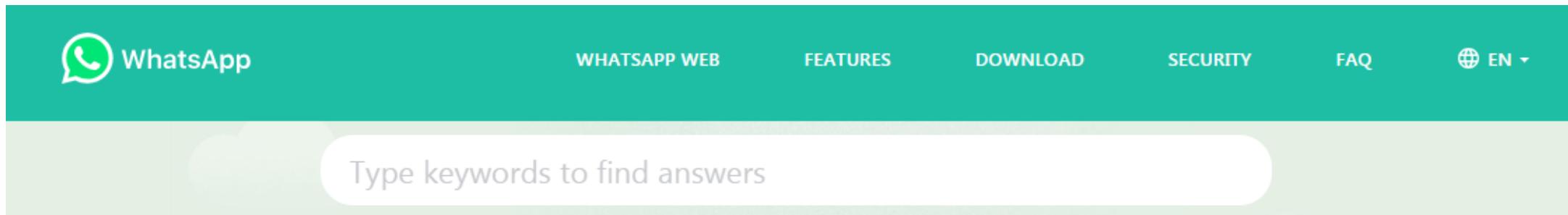
The **WebSocket API** is an advanced technology that makes it possible to open a two-way interactive communication session between the user's browser and a server without having to poll the server for a reply.

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?



The image shows the top navigation bar of the WhatsApp website. It features a teal header with the WhatsApp logo (a green speech bubble with a white phone icon) on the left. To the right of the logo are several menu items: "WHATSAAPP WEB", "FEATURES", "DOWNLOAD", "SECURITY", "FAQ", and "EN" (with a globe icon). Below the header is a search bar containing the placeholder text "Type keywords to find answers".

Android → Security and Privacy

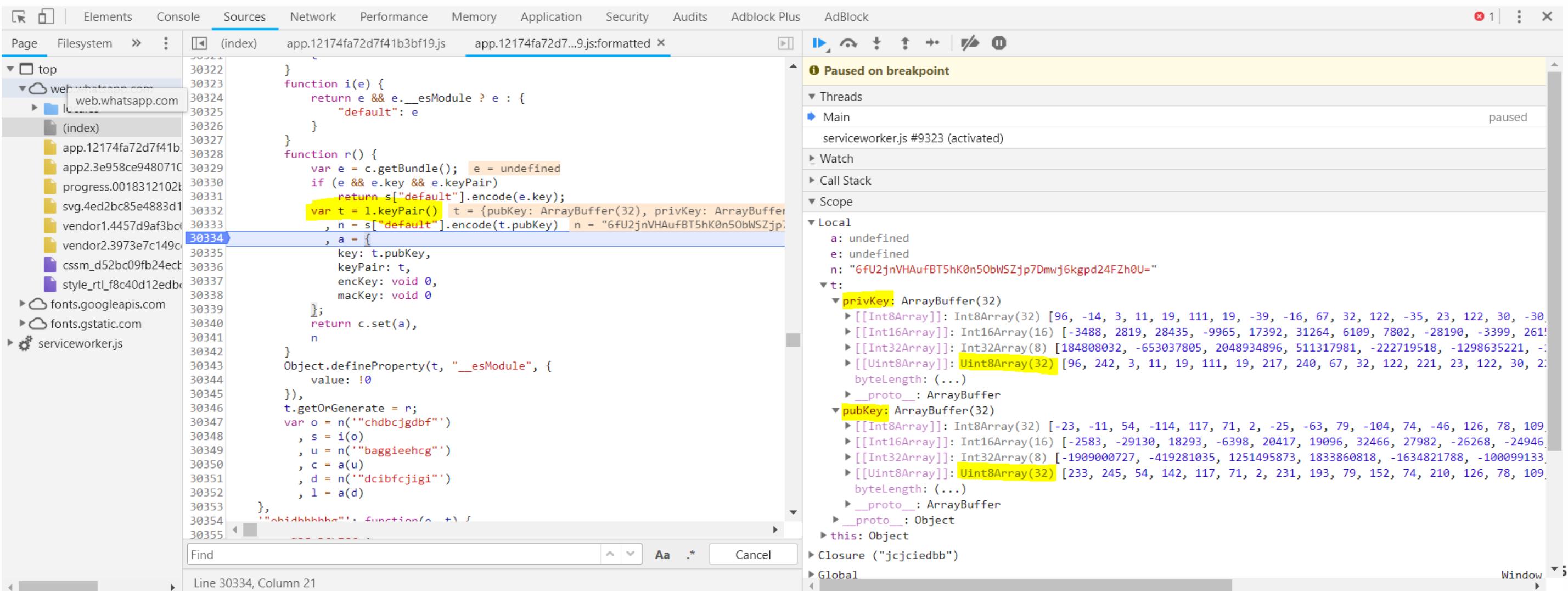
## End-to-end encryption

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



The screenshot shows the browser's developer tools with the "Sources" tab selected. The left sidebar lists the file structure, including files from web.whatsapp.com and various vendor scripts. The main pane displays the source code of `app.12174fa72d7f41b3bf19.js`. A specific line of code is highlighted in yellow: `var t = l.keyPair(); t = {pubKey: ArrayBuffer(32), privKey: ArrayBuffer(32), n: s["default"].encode(t.pubKey), a: {key: t.pubKey, keyPair: t, encKey: void 0, macKey: void 0}};`. The line number 30334 is highlighted with a blue box. The right sidebar shows the debugger interface, indicating the code is "Paused on breakpoint". The "Threads" section shows the "Main" thread is paused. The "Local" section shows variables `a`, `e`, and `n` defined. The `t` variable is expanded to show its `privKey` and `pubKey` properties as `ArrayBuffer(32)` objects.

```

30322 }
30323     function i(e) {
30324         return e && e.__esModule ? e : {
30325             "default": e
30326         }
30327     }
30328     function r() {
30329         var e = c.getBundle(); e = undefined
30330         if (e && e.key && e.keyPair)
30331             return s["default"].encode(e.key);
30332         var t = l.keyPair(); t = {pubKey: ArrayBuffer(32), privKey: ArrayBuffer(32),
30333             n: s["default"].encode(t.pubKey), a: {
30334                 key: t.pubKey,
30335                 keyPair: t,
30336                 encKey: void 0,
30337                 macKey: void 0
30338             }};
30339         return c.set(a),
30340         n
30341     }
30342     Object.defineProperty(t, "__esModule", {
30343         value: !0
30344     }),
30345     t.getOrGenerate = r;
30346     var o = n('"chdbcjgdbf"')
30347         , s = i(o)
30348         , u = n('"baggieehcg"')
30349         , c = a(u)
30350         , d = n('"dcibfcjigi"')
30351         , l = a(d)
30352     },
30353     '"ohidhhhhha": function(o) +}
30354     -----
30355

```

# WhatsApp Reversing 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



The screenshot shows a browser developer tools interface with a file tree on the left and a code editor on the right. The file tree shows a directory structure under 'web.whatsapp.com' including 'locales', '(index)', and several other files like 'app.12174fa72d7f41b...', 'app2.3e958ce9480710...', etc. The code editor displays a portion of a JavaScript file:

```
Page Filesystem > :
```

```
22182     }
22183     function i() {
22184         f || (f = n('caaaibgdja'))
22185     }
22186     function r(e) {
22187         var t = void 0;
22188         return void 0 === e ? (t = new Uint8Array(32),
22189             window.crypto.getRandomValues(t)) : t = new Uint8Array(e),
22190             t[0] &= 248,
22191             t[31] &= 127,
22192             t[31] |= 64,
22193             c({
22194                 pubKey: 32,
22195                 privKey: t,
22196                 basepoint: h
22197             }, function(e) {
22198                 var n = f._curve25519_donna(e.pubKey, e.privKey, e.basepoint);
22199                 if (n)
22200                     throw new Error("Curve25519:keyPair Error Code " + n);
22201                 return {
22202                     pubKey: u(e.pubKey, 32),
22203                     privKey: t.buffer
22204                 }
22205             })
22206 }
```

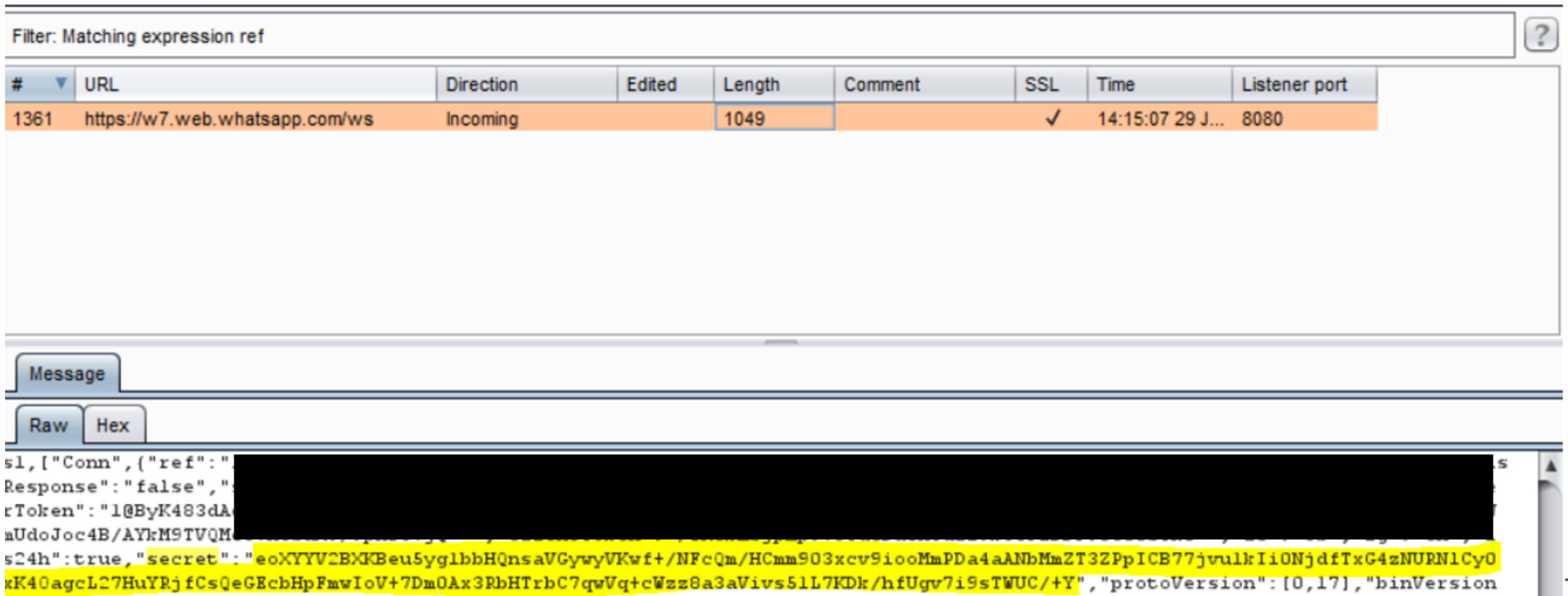
# WhatsApp Reversing Process

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:

```
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

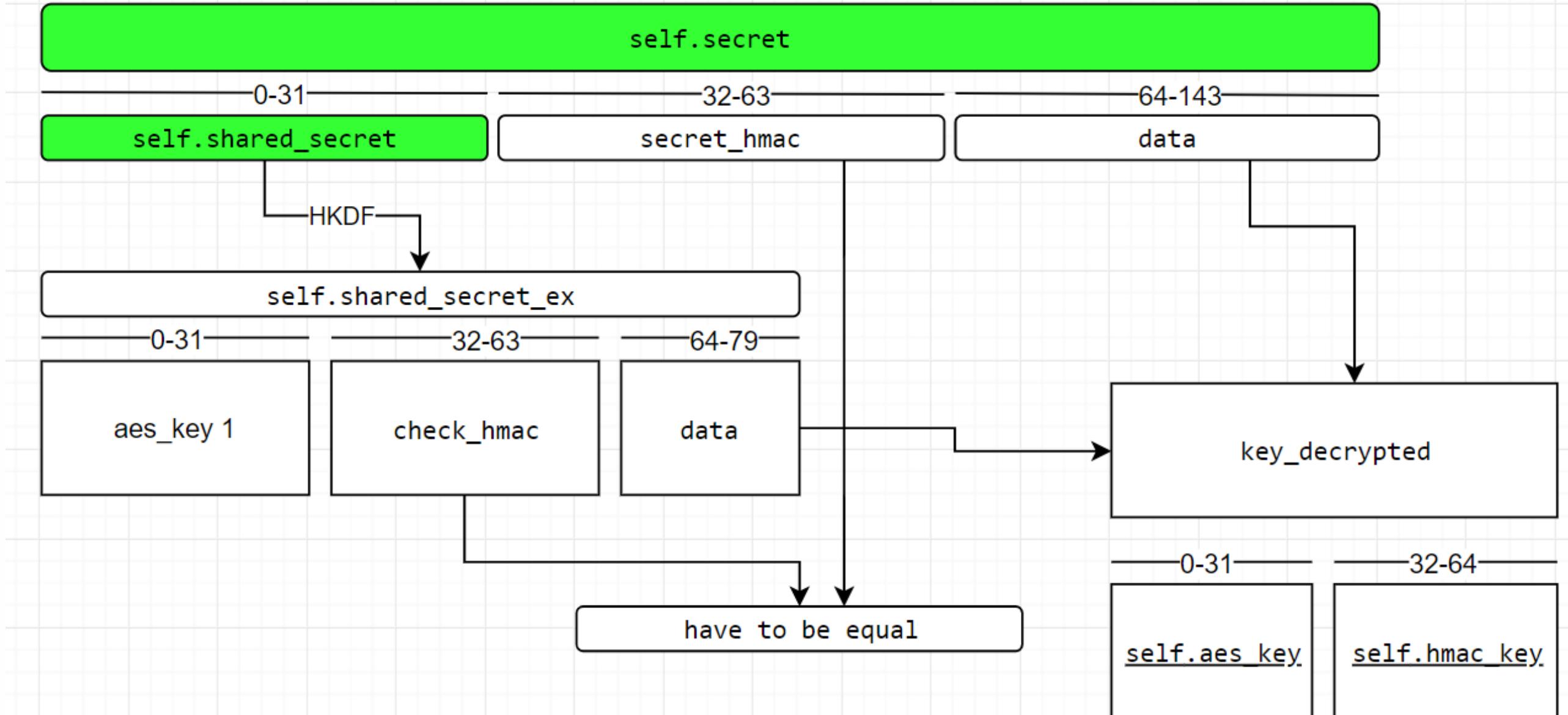
Then, after scanning the QR code with the phone we have to take the generated secret:



The screenshot shows a NetworkMiner capture interface. At the top, there's a table with columns: #, URL, Direction, Edited, Length, Comment, SSL, Time, and Listener port. A single row is selected, showing: # 1361, URL https://w7.web.whatsapp.com/ws, Direction Incoming, Edited (highlighted), Length 1049, Comment (with a checkmark), SSL checked, Time 14:15:07 29 J..., and Listener port 8080.

Below the table, there's a "Message" tab with "Raw" and "Hex" buttons. The "Raw" button is selected. The message content is partially visible, showing JSON-like data. A yellow highlight box surrounds the "secret" field, which contains a long string of characters: "eoXYYV2BxKBeu5yglbbHQnsaVGywyVKwf+/NFcQm/HCmm903xcv9iooMmPDa4aANbMmZT3ZPpICB77jvulkIiONjdfTxG4zNURN1Cy0xK40agcL27HuYRjfCsQeGEcbHpFmwIoV+7Dm0Ax3RbHTrbC7qwVq+cWzz8a3aViws51L7KDk/hfUgv7i9sTWUC/+Y".

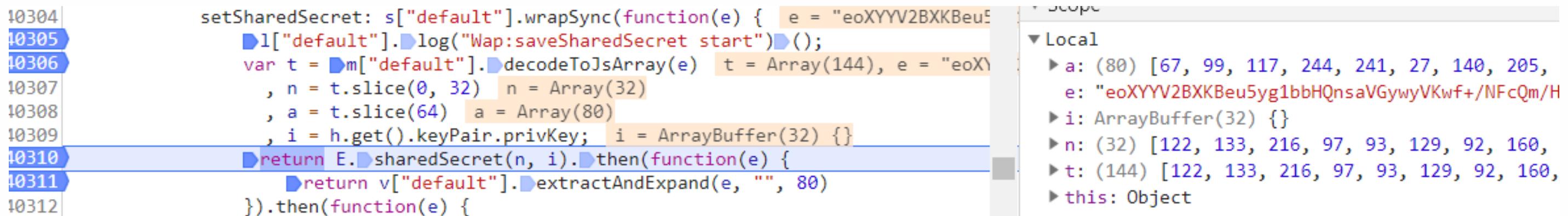
# WhatsApp Reversing Process – Shared Secret



# WhatsApp Reversing Process – Shared Secret

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:



The screenshot shows a debugger interface with the following code and local variables:

```
10304 setSharedSecret: s["default"].wrapSync(function(e) { e = "eoXYYV2BXKBeu5...  
10305   ▶ l["default"].log("Wap:saveSharedSecret start")();  
10306   var t = ▶ m["default"].decodeToJsArray(e) t = Array(144), e = "eoX...  
10307     , n = t.slice(0, 32) n = Array(32)  
10308     , a = t.slice(64) a = Array(80)  
10309     , i = h.get().keyPair.privKey; i = ArrayBuffer(32) {}  
10310   ▶ return E.sharedSecret(n, i).then(function(e) {  
10311     ▶ return v["default"].extractAndExpand(e, "", 80)  
10312   }).then(function(e) {  
10313     ▶  
10314   })  
10315 }  
10316 }  
10317 }  
10318 }  
10319 }  
10320 }  
10321 }  
10322 }  
10323 }  
10324 }  
10325 }  
10326 }  
10327 }  
10328 }  
10329 }  
10330 }  
10331 }  
10332 }  
10333 }  
10334 }  
10335 }  
10336 }  
10337 }  
10338 }  
10339 }  
10340 }  
10341 }  
10342 }  
10343 }  
10344 }  
10345 }  
10346 }  
10347 }  
10348 }  
10349 }  
10350 }  
10351 }  
10352 }  
10353 }  
10354 }  
10355 }  
10356 }  
10357 }  
10358 }  
10359 }  
10360 }  
10361 }  
10362 }  
10363 }  
10364 }  
10365 }  
10366 }  
10367 }  
10368 }  
10369 }  
10370 }  
10371 }  
10372 }  
10373 }  
10374 }  
10375 }  
10376 }  
10377 }  
10378 }  
10379 }  
10380 }  
10381 }  
10382 }  
10383 }  
10384 }  
10385 }  
10386 }  
10387 }  
10388 }  
10389 }  
10390 }  
10391 }  
10392 }  
10393 }  
10394 }  
10395 }  
10396 }  
10397 }  
10398 }  
10399 }  
10400 }  
10401 }  
10402 }  
10403 }  
10404 }  
10405 }  
10406 }  
10407 }  
10408 }  
10409 }  
10410 }  
10411 }  
10412 }  
10413 }  
10414 }  
10415 }  
10416 }  
10417 }  
10418 }  
10419 }  
10420 }  
10421 }  
10422 }  
10423 }  
10424 }  
10425 }  
10426 }  
10427 }  
10428 }  
10429 }  
10430 }  
10431 }  
10432 }  
10433 }  
10434 }  
10435 }  
10436 }  
10437 }  
10438 }  
10439 }  
10440 }  
10441 }  
10442 }  
10443 }  
10444 }  
10445 }  
10446 }  
10447 }  
10448 }  
10449 }  
10450 }  
10451 }  
10452 }  
10453 }  
10454 }  
10455 }  
10456 }  
10457 }  
10458 }  
10459 }  
10460 }  
10461 }  
10462 }  
10463 }  
10464 }  
10465 }  
10466 }  
10467 }  
10468 }  
10469 }  
10470 }  
10471 }  
10472 }  
10473 }  
10474 }  
10475 }  
10476 }  
10477 }  
10478 }  
10479 }  
10480 }  
10481 }  
10482 }  
10483 }  
10484 }  
10485 }  
10486 }  
10487 }  
10488 }  
10489 }  
10490 }  
10491 }  
10492 }  
10493 }  
10494 }  
10495 }  
10496 }  
10497 }  
10498 }  
10499 }  
10500 }  
10501 }  
10502 }  
10503 }  
10504 }  
10505 }  
10506 }  
10507 }  
10508 }  
10509 }  
10510 }  
10511 }  
10512 }  
10513 }  
10514 }  
10515 }  
10516 }  
10517 }  
10518 }  
10519 }  
10520 }  
10521 }  
10522 }  
10523 }  
10524 }  
10525 }  
10526 }  
10527 }  
10528 }  
10529 }  
10530 }  
10531 }  
10532 }  
10533 }  
10534 }  
10535 }  
10536 }  
10537 }  
10538 }  
10539 }  
10540 }  
10541 }  
10542 }  
10543 }  
10544 }  
10545 }  
10546 }  
10547 }  
10548 }  
10549 }  
10550 }  
10551 }  
10552 }  
10553 }  
10554 }  
10555 }  
10556 }  
10557 }  
10558 }  
10559 }  
10560 }  
10561 }  
10562 }  
10563 }  
10564 }  
10565 }  
10566 }  
10567 }  
10568 }  
10569 }  
10570 }  
10571 }  
10572 }  
10573 }  
10574 }  
10575 }  
10576 }  
10577 }  
10578 }  
10579 }  
10580 }  
10581 }  
10582 }  
10583 }  
10584 }  
10585 }  
10586 }  
10587 }  
10588 }  
10589 }  
10590 }  
10591 }  
10592 }  
10593 }  
10594 }  
10595 }  
10596 }  
10597 }  
10598 }  
10599 }  
10600 }  
10601 }  
10602 }  
10603 }  
10604 }  
10605 }  
10606 }  
10607 }  
10608 }  
10609 }  
10610 }  
10611 }  
10612 }  
10613 }  
10614 }  
10615 }  
10616 }  
10617 }  
10618 }  
10619 }  
10620 }  
10621 }  
10622 }  
10623 }  
10624 }  
10625 }  
10626 }  
10627 }  
10628 }  
10629 }  
10630 }  
10631 }  
10632 }  
10633 }  
10634 }  
10635 }  
10636 }  
10637 }  
10638 }  
10639 }  
10640 }  
10641 }  
10642 }  
10643 }  
10644 }  
10645 }  
10646 }  
10647 }  
10648 }  
10649 }  
10650 }  
10651 }  
10652 }  
10653 }  
10654 }  
10655 }  
10656 }  
10657 }  
10658 }  
10659 }  
10660 }  
10661 }  
10662 }  
10663 }  
10664 }  
10665 }  
10666 }  
10667 }  
10668 }  
10669 }  
10670 }  
10671 }  
10672 }  
10673 }  
10674 }  
10675 }  
10676 }  
10677 }  
10678 }  
10679 }  
10680 }  
10681 }  
10682 }  
10683 }  
10684 }  
10685 }  
10686 }  
10687 }  
10688 }  
10689 }  
10690 }  
10691 }  
10692 }  
10693 }  
10694 }  
10695 }  
10696 }  
10697 }  
10698 }  
10699 }  
10700 }  
10701 }  
10702 }  
10703 }  
10704 }  
10705 }  
10706 }  
10707 }  
10708 }  
10709 }  
10710 }  
10711 }  
10712 }  
10713 }  
10714 }  
10715 }  
10716 }  
10717 }  
10718 }  
10719 }  
10720 }  
10721 }  
10722 }  
10723 }  
10724 }  
10725 }  
10726 }  
10727 }  
10728 }  
10729 }  
10730 }  
10731 }  
10732 }  
10733 }  
10734 }  
10735 }  
10736 }  
10737 }  
10738 }  
10739 }  
10740 }  
10741 }  
10742 }  
10743 }  
10744 }  
10745 }  
10746 }  
10747 }  
10748 }  
10749 }  
10750 }  
10751 }  
10752 }  
10753 }  
10754 }  
10755 }  
10756 }  
10757 }  
10758 }  
10759 }  
10760 }  
10761 }  
10762 }  
10763 }  
10764 }  
10765 }  
10766 }  
10767 }  
10768 }  
10769 }  
10770 }  
10771 }  
10772 }  
10773 }  
10774 }  
10775 }  
10776 }  
10777 }  
10778 }  
10779 }  
10780 }  
10781 }  
10782 }  
10783 }  
10784 }  
10785 }  
10786 }  
10787 }  
10788 }  
10789 }  
10790 }  
10791 }  
10792 }  
10793 }  
10794 }  
10795 }  
10796 }  
10797 }  
10798 }  
10799 }  
10800 }  
10801 }  
10802 }  
10803 }  
10804 }  
10805 }  
10806 }  
10807 }  
10808 }  
10809 }  
10810 }  
10811 }  
10812 }  
10813 }  
10814 }  
10815 }  
10816 }  
10817 }  
10818 }  
10819 }  
10820 }  
10821 }  
10822 }  
10823 }  
10824 }  
10825 }  
10826 }  
10827 }  
10828 }  
10829 }  
10830 }  
10831 }  
10832 }  
10833 }  
10834 }  
10835 }  
10836 }  
10837 }  
10838 }  
10839 }  
10840 }  
10841 }  
10842 }  
10843 }  
10844 }  
10845 }  
10846 }  
10847 }  
10848 }  
10849 }  
10850 }  
10851 }  
10852 }  
10853 }  
10854 }  
10855 }  
10856 }  
10857 }  
10858 }  
10859 }  
10860 }  
10861 }  
10862 }  
10863 }  
10864 }  
10865 }  
10866 }  
10867 }  
10868 }  
10869 }  
10870 }  
10871 }  
10872 }  
10873 }  
10874 }  
10875 }  
10876 }  
10877 }  
10878 }  
10879 }  
10880 }  
10881 }  
10882 }  
10883 }  
10884 }  
10885 }  
10886 }  
10887 }  
10888 }  
10889 }  
10890 }  
10891 }  
10892 }  
10893 }  
10894 }  
10895 }  
10896 }  
10897 }  
10898 }  
10899 }  
10900 }  
10901 }  
10902 }  
10903 }  
10904 }  
10905 }  
10906 }  
10907 }  
10908 }  
10909 }  
10910 }  
10911 }  
10912 }  
10913 }  
10914 }  
10915 }  
10916 }  
10917 }  
10918 }  
10919 }  
10920 }  
10921 }  
10922 }  
10923 }  
10924 }  
10925 }  
10926 }  
10927 }  
10928 }  
10929 }  
10930 }  
10931 }  
10932 }  
10933 }  
10934 }  
10935 }  
10936 }  
10937 }  
10938 }  
10939 }  
10940 }  
10941 }  
10942 }  
10943 }  
10944 }  
10945 }  
10946 }  
10947 }  
10948 }  
10949 }  
10950 }  
10951 }  
10952 }  
10953 }  
10954 }  
10955 }  
10956 }  
10957 }  
10958 }  
10959 }  
10960 }  
10961 }  
10962 }  
10963 }  
10964 }  
10965 }  
10966 }  
10967 }  
10968 }  
10969 }  
10970 }  
10971 }  
10972 }  
10973 }  
10974 }  
10975 }  
10976 }  
10977 }  
10978 }  
10979 }  
10980 }  
10981 }  
10982 }  
10983 }  
10984 }  
10985 }  
10986 }  
10987 }  
10988 }  
10989 }  
10990 }  
10991 }  
10992 }  
10993 }  
10994 }  
10995 }  
10996 }  
10997 }  
10998 }  
10999 }  
10999 }
```

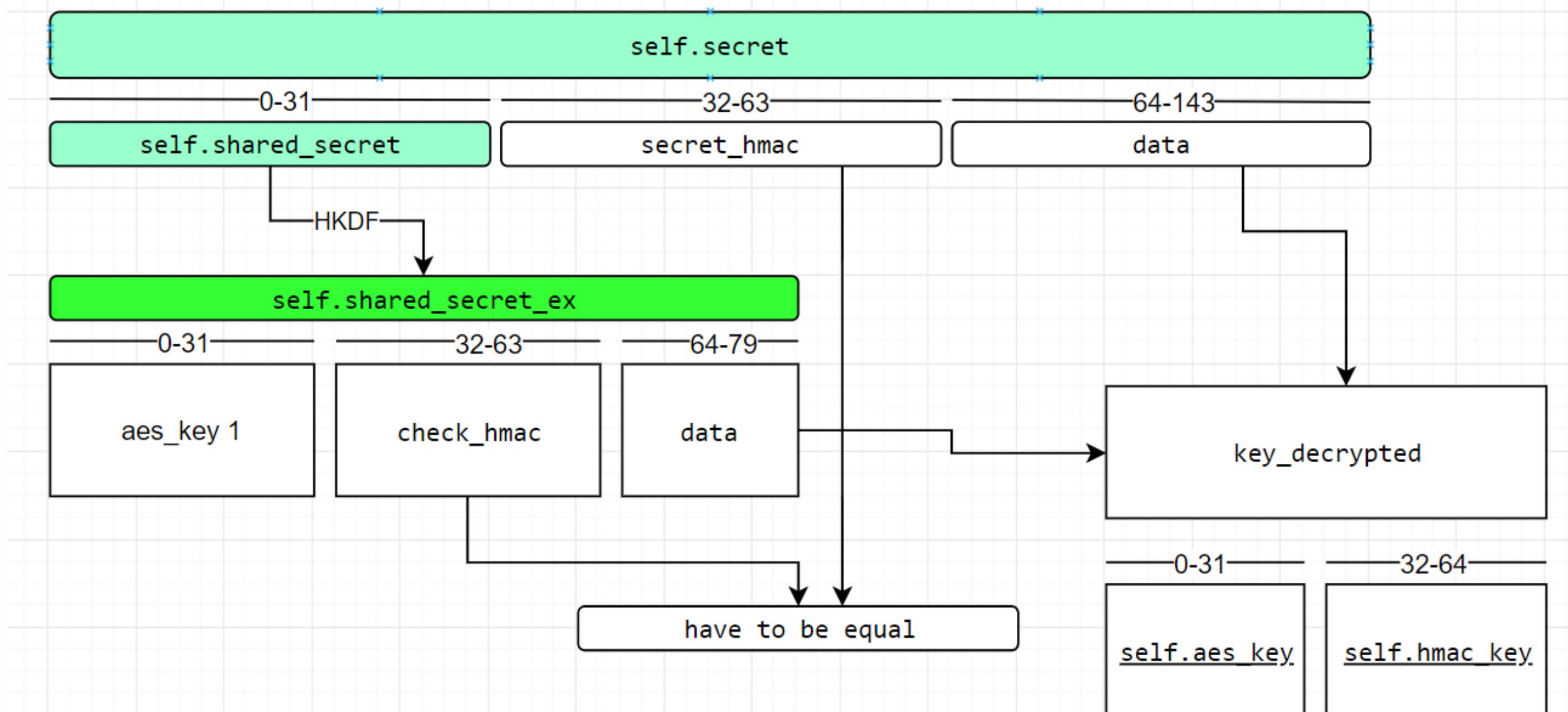
# WhatsApp Reversing Process – Shared Secret

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

```
40310         return E.sharedSecret(n, i).then(function(e) { e = ArrayBuffer(32) {}  
40311         return v["default"].extractAndExpand(e, "", 80)  
40312     }).then(function(e) {  
40313         var i = new Uint8Array(e, 0, 32)  
40314         , r = new Uint8Array(e, 32, 32)  
40315         , o = new Uint8Array(e, 64, 16)  
40316         , 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



# WhatsApp Reversing Process – hmac\_sha256

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**

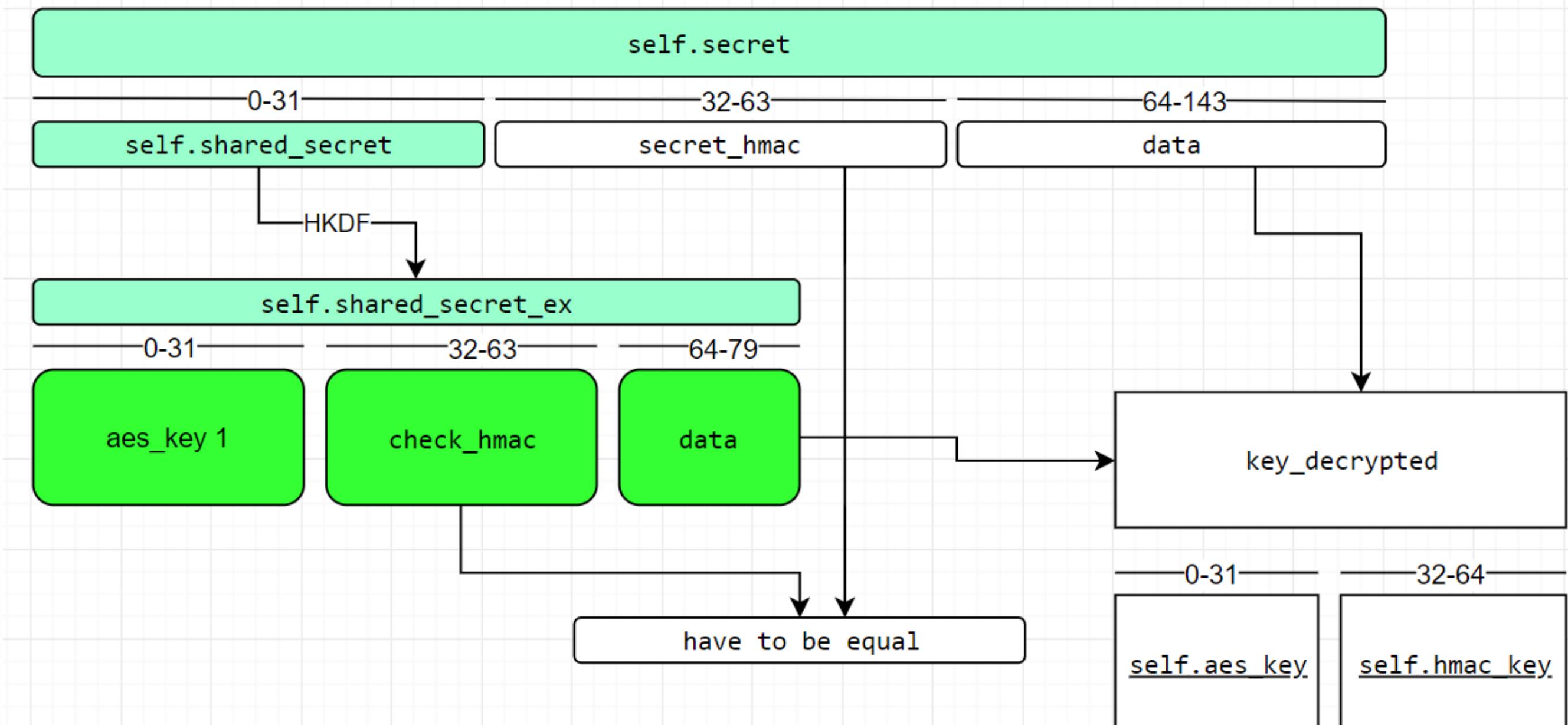
```

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(80) {}}
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.►concat(a));
40317      ►return ►new C.HmacSha256(r).►sign(s).►then(function(e) {
40318          var n = m["default"].encode(e)
40319          , r = m["default"].encode(t.slice(32, 64));
40320          if (r !== n)
40321              return void 1["default"].error("Wap:saveSharedSecret hmac mismatch")
40322          var s = N["default"].build(o, new Uint8Array(a)).readByteArray();
40323          return (0,

```

▼ Scope  
 ▼ Local  
 ▼ e: ArrayBuffer(80)  
 ► [[Int8Array]]: Int8Array(80) [16]  
 ► [[Int16Array]]: Int16Array(40) [  
 ► [[Int32Array]]: Int32Array(20) [  
 ► [[Uint8Array]]: Uint8Array(80) [  
 byteLength: (...)  
 ► \_\_proto\_\_: ArrayBuffer  
 ► i: Uint8Array(32) [16, 96, 27, 22,  
 ► o: Uint8Array(16) [196, 199, 28, 1  
 ► r: Uint8Array(32) [195, 18, 115, 2

# WhatsApp Reversing Process – hmac\_sha256



# WhatsApp Reversing Process – hmac\_sha256

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**)

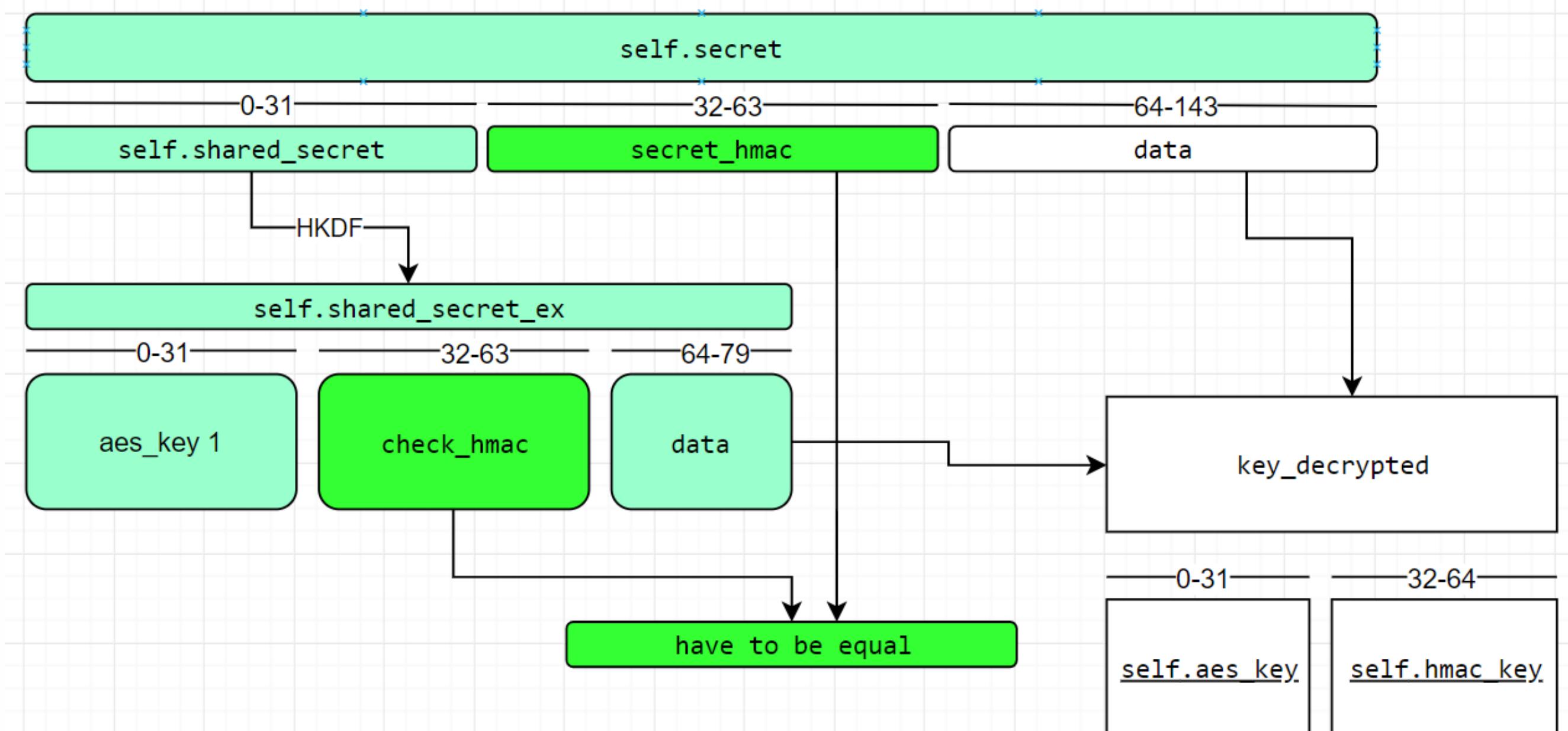
```
40317     ►return ►new C.HmacSha256(r).►sign(s).►then(function(e) { e = ArrayBuffer
40318         var n = ►m["default"].►encode(e) n = "ppvdN8XL/YqKDjJw2uGgDWzJmU92T6SAg
40319             , r = ►m["default"].►encode(t►slice(32, 64)); r = "ppvdN8XL/YqKDjJw
40320             if (r !== n)
40321                 return void l["default"].error("Wap:saveSharedSecret hmac mismatch")
40322             var s = N["default"].build(o, new Uint8Array(a)).readByteArray();
40323             return (0,
40324                 C.aesCbcDecrypt)(i, s).►then(function(e) {
```

```
▶ [[Uint8Array]]: Uint8Array(32) [166, byteLength: (...)]
▶ __proto__: ArrayBuffer
n: "ppvdN8XL/YqKDjJw2uGgDWzJmU92T6SAge"
r: "ppvdN8XL/YqKDjJw2uGgDWzJmU92T6SAge"
s: undefined
this: undefined
```

In python it will look like this:

```
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



# WhatsApp Reversing Process – AES Keys

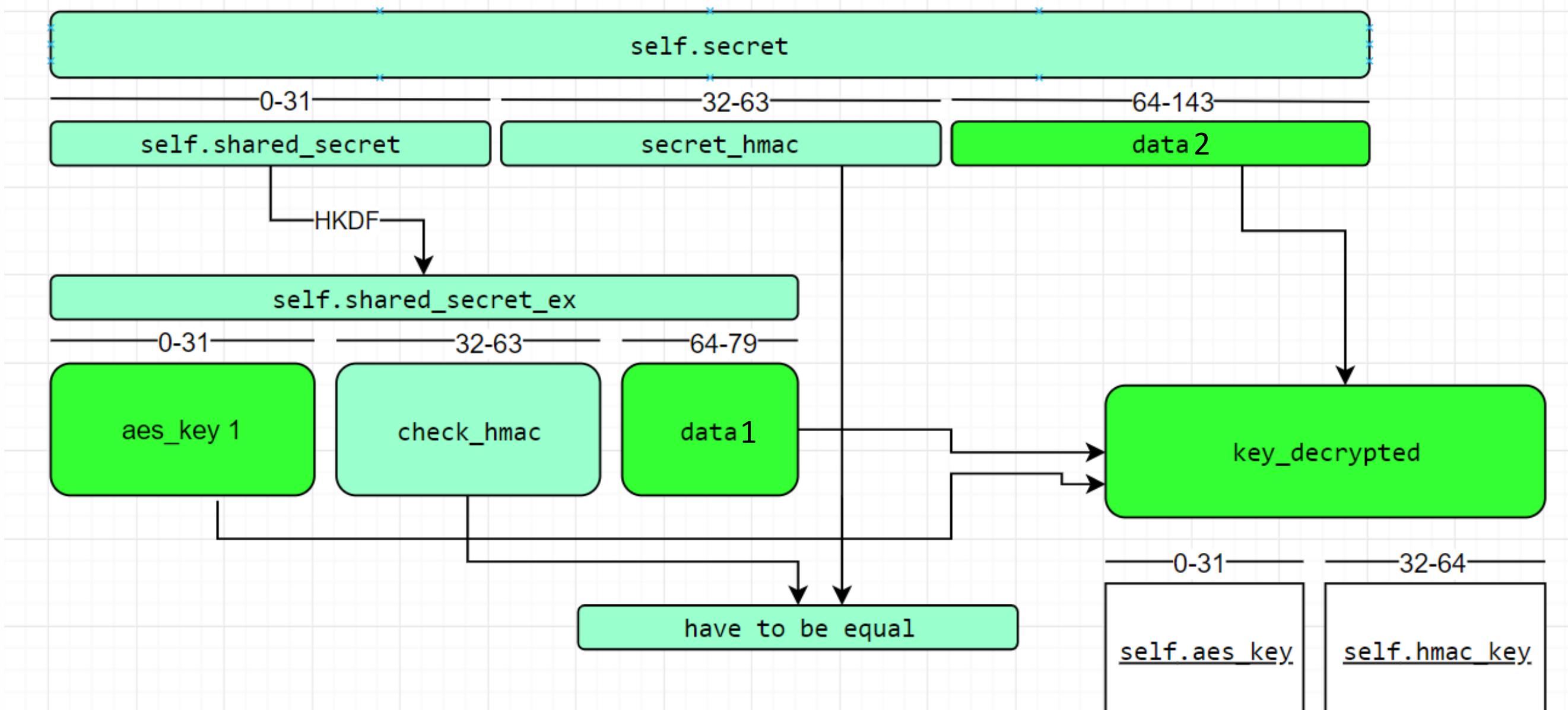
The last encryption related function in this block is ‘aesCbcDecrypt’ which uses two parameters:

- 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**.

```
40322      ▶ var s = ▶N["default"].build(o, ▶new Uint8Array(a)).readByteArray(); s = l
40323      ▶ return (0,
40324      ▶ C.aesCbcDecrypt)▶(i, s).▶then(function(e) {
40325          var t = new Uint8Array(e,0,32)
40326          , n = new Uint8Array(e,32,32);
40327          ▶h.▶setSecretKeys("0.1", t, n),
40328          l["default"].log("Wap:saveSharedSecret done")()
40329      })
40330      })
40331  })
```

...  
▶ proto: ArrayBuffer  
n: "ppvdN8XL/YqKDjJw2uGgDWzJ"  
r: "ppvdN8XL/YqKDjJw2uGgDWzJ"  
▶ s: Uint8Array(96) [196, 199,  
this: undefined  
▼ Closure  
▶ i: Uint8Array(32) [16, 96, 2  
▶ o: Uint8Array(16) [196, 199,

# WhatsApp Reversing Process – AES Keys



# WhatsApp Reversing Process – AES Keys

This way we will get the AES Key 't' and HMAC Key 'n'

```
40322     var s = N["default"].build(o, new Uint8Array(a)).readByteArray(); s = l
40323     return (0,
40324     C.aesCbcDecrypt)(i, s).then(function(e) {
40325         var t = new Uint8Array(e,0,32)
40326         , n = new Uint8Array(e,32,32);
40327         h.setSecretKeys("0.1", t, n),
40328         l["default"].log("Wap:saveSharedSecret done")()
40329     })
40330   }
40331 }
```

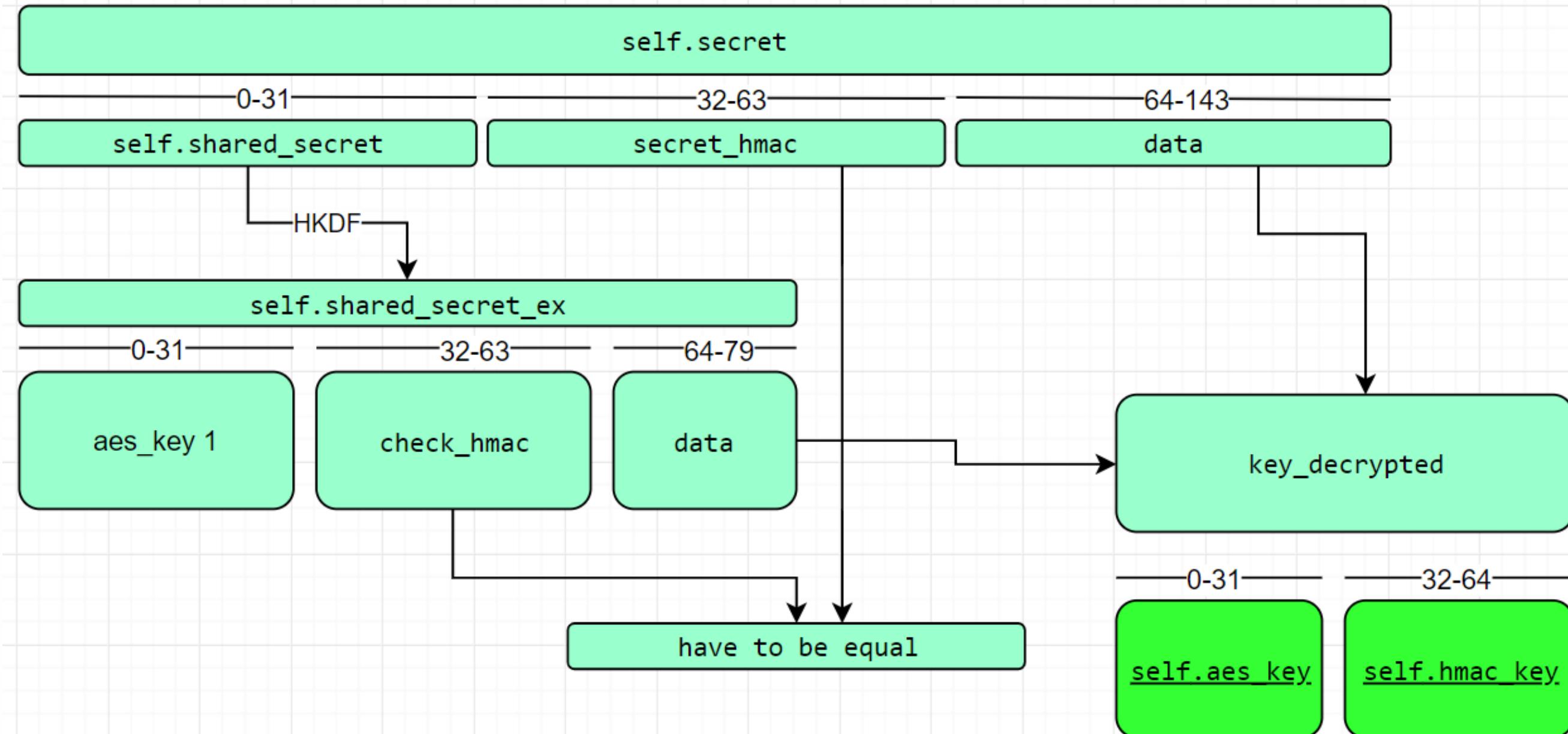
By https://... /

- ▶ \_\_proto\_\_: ArrayBuffer
- n: "ppvdN8XL/YqKDjJw2uGgDWzJ"
- r: "ppvdN8XL/YqKDjJw2uGgDWzJ"
- ▶ s: Uint8Array(96) [196, 199, ...]
- this: undefined

▼ Closure

- ▶ i: Uint8Array(32) [16, 96, 2 ...]
- ▶ o: Uint8Array(16) [196, 199, ...]

# WhatsApp Reversing Process – AES Keys



# WhatsApp Reversing Process – Code

```
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]))

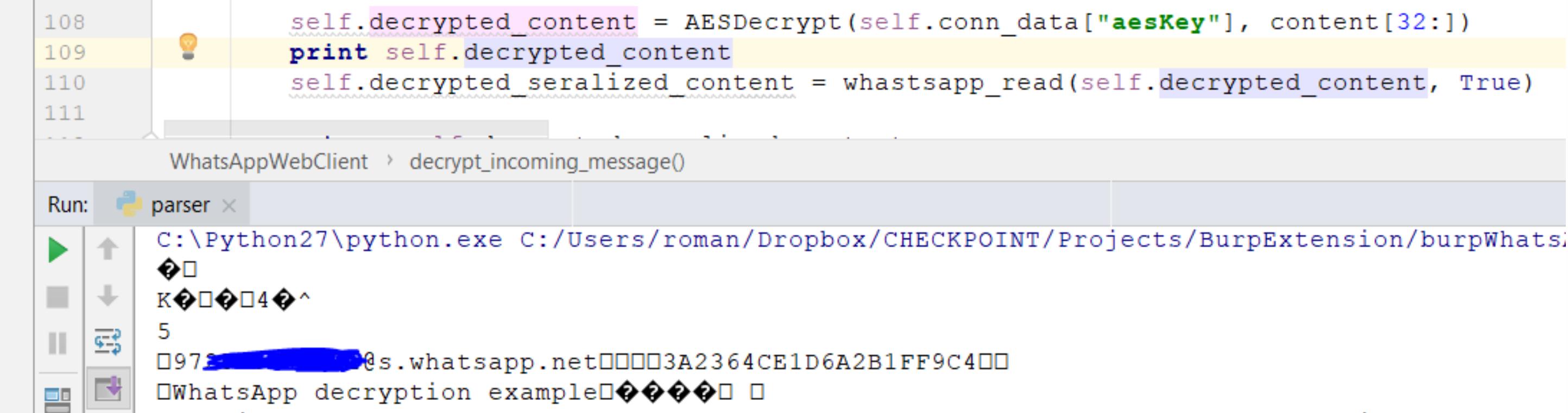
self.secret = base64.b64decode(ref_dict["secret"])
self.shared_secret = self.private_key.get_shared_key(curve25519.Public(self.secret[:32]), lambda key: key)
self.shared_secret_ex = HKDF(self.shared_secret, 80)

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.



The screenshot shows a debugger interface with the following details:

- Code View:** Shows Python code for decrypting WhatsApp messages. Lines 108-111 are highlighted:

```
108     self.decrypted_content = AESDecrypt(self.conn_data["aesKey"], content[32:])
109     print self.decrypted_content
110     self.decrypted_serialized_content = whatsapp_read(self.decrypted_content, True)
111
```
- Call Stack:** Shows the call stack: WhatsAppWebClient > decrypt\_incoming\_message()
- Run Tab:** Shows the command being run: C:\Python27\python.exe C:/Users/roman/Dropbox/CHECKPOINT/Projects/BurpExtension/burpWhatsApp.py
- Output Window:** Displays the decrypted WhatsApp message content, which includes a timestamp (5), a recipient identifier (@s.whatsapp.net), and a message body ("WhatsApp decryption example").

# WhatsApp Reversing Process – protobuf data

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



The image shows a file explorer on the left and a code editor on the right. The file explorer lists several files and folders under the 'app' directory, including 'app.12174fa72d7f41b3bf19', 'app2.3e958ce9480710cfe92', 'progress.0018312102bec44', 'svg.4ed2bc85e4883d1e276a', 'vendor1.4457d9af3bc60cc9a', 'vendor2.3973e7c149ced441', 'cssm\_d52bc09fb24ecbd5f9', 'style\_f8c40d12edb6fe7b2c', 'dyn.web.whatsapp.com', 'fonts.googleapis.com', 'fonts.gstatic.com', and 'serviceworker.js'. The code editor displays a snippet of JavaScript code that defines a mapping for a protobuf message. The code includes a list of fields with their types and flags, and an 'exports' object containing several functions named 'WebMessageInfo', 'PaymentInfo', 'WebNotificationsInfo', 'NotificationMessageInfo', 'TabletNotificationsInfo', and 'WebFeatures', all highlighted in yellow.

```
18999 },
19000 p(u, {
19001   labelsDisplay: [1, E, u.FLAG],
19002   voipIndividualOutgoing: [2, E, u.FLAG],
19003   groupsV3: [3, E, u.FLAG],
19004   groupsV3Create: [4, E, u.FLAG],
19005   changeNumberV2: [5, E, u.FLAG],
19006   queryStatusV3Thumbnail: [6, E, u.FLAG],
19007   liveLocations: [7, E, u.FLAG],
19008   queryVname: [8, E, u.FLAG],
19009   voipIndividualIncoming: [9, E, u.FLAG],
19010   quickRepliesQuery: [10, E, u.FLAG],
19011   payments: [11, E, u.FLAG],
19012   stickerPackQuery: [12, E, u.FLAG],
19013   liveLocationsFinal: [13, E, u.FLAG]
19014 }),
19015 e.exports = {
19016   WebMessageInfo: a,
19017   PaymentInfo: i,
19018   WebNotificationsInfo: r,
19019   NotificationMessageInfo: o,
19020   TabletNotificationsInfo: s,
19021   WebFeatures: u
19022 }
19023 },
19024 '"defdechchga": function(e, t, n) {
```

## WhatsApp Reversing Process – protobuf data

This is a part of our protobuf file:



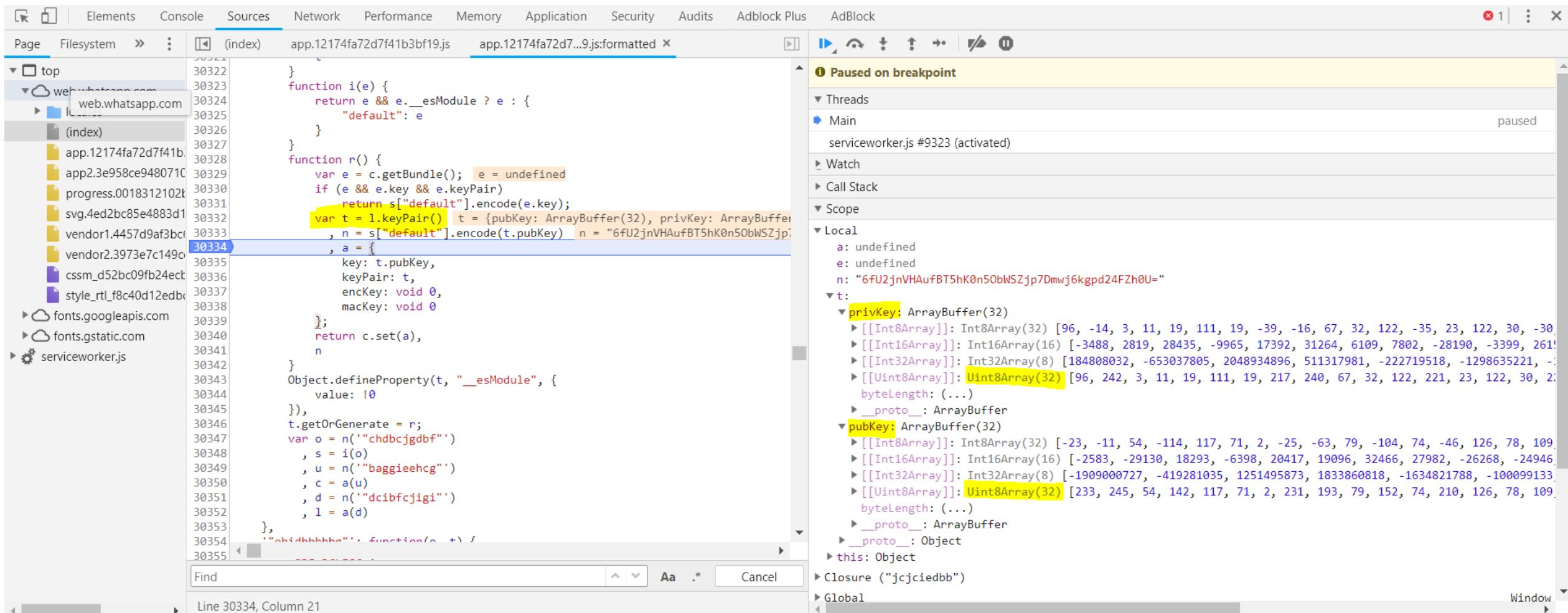
```
whatsapp.proto x
72     message WebMessageInfo {
73         optional MessageKey key = 1;
74         optional Message message = 2;
75         optional uint64 messageTimestamp = 3;
76         optional STATUS status = 4;
77         optional string participant = 5;
78         optional bool ignore = 6;
79         optional bool starred = 7;
80         optional bool broadcast = 8;
81         optional string pushName = 9;
82         optional string mediaCiphertextSha256 = 10;
83         optional bool multicast = 11;
84         optional bool urlText = 12;
85         optional bool urlNumber = 13;
86         optional STUBTYPE messageStubType = 14;
87         optional bool clearMedia = 15;
88         optional string messageStubParameters = 16;
89         optional uint32 duration = 17;
90         optional string labels = 18;
91         optional bytes paymentInfo = 19;
92     }
```



# 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



```

    30322 }
    30323     function i(e) {
    30324         return e && e.__esModule ? e : {
    30325             "default": e
    30326         }
    30327     }
    30328     function r() {
    30329         var e = c.getBundle(); e = undefined
    30330         if (e && e.key && e.keyPair)
    30331             return s["default"].encode(e.key);
    30332         var t = l.keyPair() t = {pubKey: ArrayBuffer(32), privKey: ArrayBuffer(32),
    30333             , n = s["default"].encode(t.pubKey) n = "6fU2jnVHAufBT5hK0n50bWSZjp7Dmwj6kgpd24FZh0U=",
    30334             , a = {
    30335                 key: t.pubKey,
    30336                 keyPair: t,
    30337                 encKey: void 0,
    30338                 macKey: void 0
    30339             };
    30340             return c.set(a),
    30341             n
    30342         }
    30343         Object.defineProperty(t, "__esModule", {
    30344             value: !0
    30345         }),
    30346         t.getOrGenerate = r;
    30347         var o = n('chdbcjgdbf')
    30348             , s = i(o)
    30349             , u = n('baggieehcg')
    30350             , c = a(u)
    30351             , d = n('dcibfcjigi')
    30352             , l = a(d)
    30353         },
    30354         "chidhhhhha": function(e) +}
    30355     }

```

The screenshot shows the Chrome DevTools Sources tab with the code for WhatsApp Web. A breakpoint is set at line 30334, and the execution is paused. The Local pane displays the 'privKey' and 'pubKey' variables, which are both instances of ArrayBuffer(32). The variable 'n' contains a long string representing the base64 encoded keys.

# Accessing the Keys – Burp Extension Secrets

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

Filter: Matching expression ref

?

#	URL	Direction	Edited	Length	Comment	SSL	Time	Listener port
1361	https://w7.web.whatsapp.com/ws	Incoming		1049		✓	14:15:07 29 J...	8080

# The Extension

B Burp Suite Professional v1.7.35 - Temporary Project - licensed to Checkpoint Ltd. [3 user license]

Burp Intruder Repeater Window Help Backslash

Target	Proxy	Spider	Scanner	Intruder	Repeater	Sequencer	Decoder	Comparer	Extender	Project options	User options
Alerts	Additional Scanner Checks	JSON Beautifier			Software Vulnerability Scanner			CSRF	Logger++	WhatsApp Decoder	

WHATAPP DECRYPTION AND ENCRYPTION EXTENSION BY DIKLA BARDA, ROMAN ZAIKIN

Ref object: `:_manufacturer":"samsung","device_model":"star2lte","os_build_number":"R16NW.G965FXXU1BRF8"},"pushname":"RomanZ","tos":0}`

Private Key: `09, 206, 235, 93, 219, 38, 23, 2, 193, 102, 123, 7, 182, 74, 201, 130, 254, 26, 215, 123, 115, 223, 200, 185, 19, 61, 89, 106, 156, 111]`

Public Key: `5, 229, 16, 179, 94, 246, 108, 29, 22, 106, 207, 209, 84, 186, 243, 144, 199, 30, 52, 52, 128, 207, 143, 197, 44, 167, 83, 180, 196, 37]`

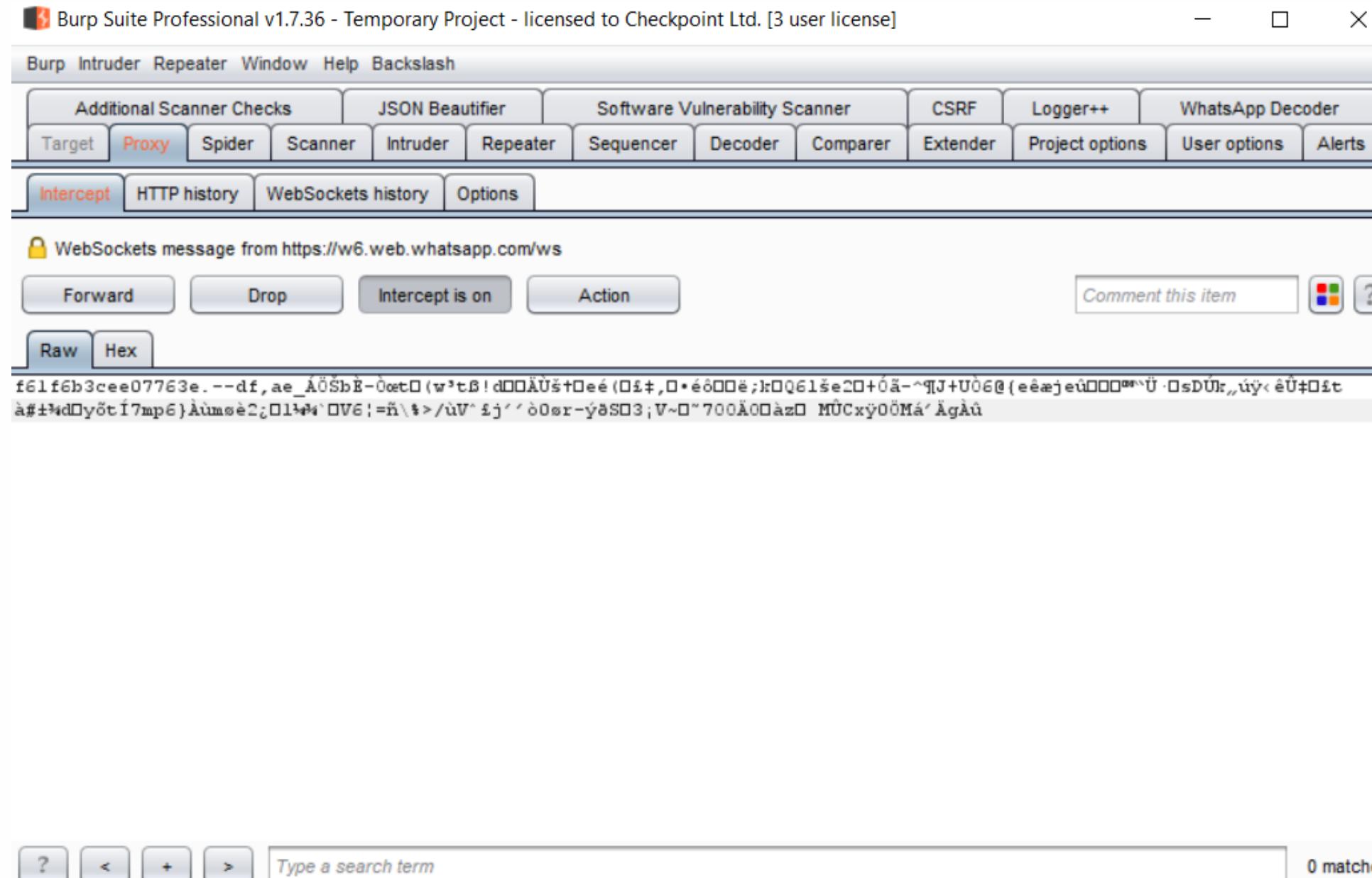
Connect Clear

Incoming Encrypt Decrypt Outgoing

CONNECTION STATUS: CONNECTED

ACTION STATUS: OK

# Decrypt the incoming data



Burp Suite Professional v1.7.36 - Temporary Project - licensed to Checkpoint Ltd. [3 user license]

Burp Intruder Repeater Window Help Backslash

Additional Scanner Checks JSON Beautifier Software Vulnerability Scanner CSRF Logger++ WhatsApp Decoder

Target Proxy Spider Scanner Intruder Repeater Sequencer Decoder Comparer Extender Project options User options Alerts

Intercept HTTP history WebSockets history Options

🔒 WebSockets message from https://w6.web.whatsapp.com/ws

Forward Drop Intercept is on Action Comment this item

Raw Hex

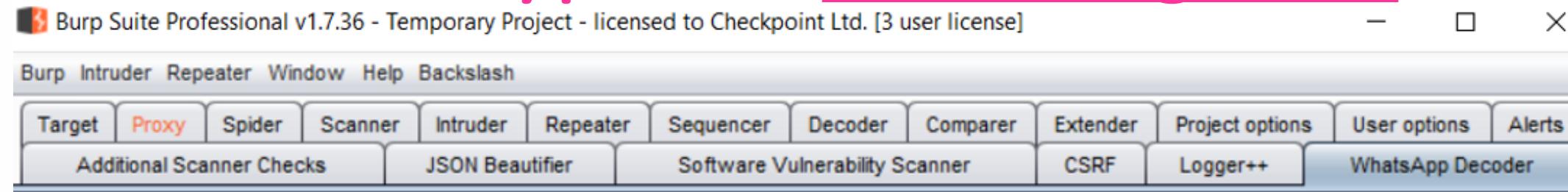
```
f61f6b3cee07763e.--df,ae_ÄÖŠbÈ-Öet□(w³tß!d□□ÄÜš+□eé(□í‡,□*éö□□ë;□□Q61še20+Öä-^qJ+Uö6@{eêæjeû□□□mñÜ·□sDÜr,,úý<éÜ#öit
à#‡ñdöyötí7mp6}Àùmøè2;□1ññ`□V6!=ñ\ñ>/ùV^ëj''ò0er-ýëSØ3;V~ñ~700À0□àz□ MÜCxÿÖÖMá'ÀgÀû
```

Type a search term

0 matches

@BLACK HAT EVENTS

# Decrypt the incoming data



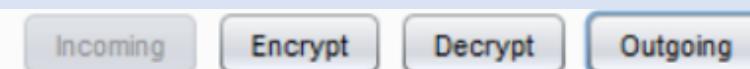
**conversation** – This is the actual content which is sent.

**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.



CONNECTION STATUS: CONNECTED

ACTION STATUS: OK

# Decrypt the outgoing data

```
91635
91636
91637
91638
91639     "default": e
91640   }
91641   }
91642   }
91643   }
91644   }
91645   }
91646   }
91647   }
91648   }
91649   }
91650   }
91651   }
91652   }
91653   }
91654   }

function n(e, t, a) { e = ArrayBuffer(32) {}, t = ArrayBuffer(32) {},
91639   return (0,
91640     c.aesCbcEncrypt)(e, new Uint8Array(a)).then(function(e) {
91641       return new d.HmacSha256(t).sign(e).then(function(t) {
91642         return p.build(t, e).readBuffer()
91643       })
91644     })
91645   }
91646   function r(e, t, a) {
91647     return s["default"].resolve(new Uint8Array(a)).then(function(a) {
91648       var i = f.encode(a.subarray(0, 32))
91649       , n = a.subarray(32);
91650       return new d.HmacSha256(t).sign(n).then(function(t) {
91651         var a = f.encode(t);
91652         if (i !== a)
91653           return Promise.reject("HMAC mismatch")
91654       })
91655     })
91656   }
91657 }

function s(e) {
91658   return new Promise(function(t, a) {
91659     var n = document.createElement("script");
91660     n.src = "https://www.google.com/recaptcha/api.js?onload=onLoad&render=explicit&hl=en";
91661     n.onload = onLoad;
91662     document.body.appendChild(n);
91663   })
91664 }
```

aesCbc

2 matches

serviceworker.js #9323 (activated)

▶ Watch

▶ Call Stack

▼ Scope

▼ Local

- ▶ a: ArrayBuffer(93)
  - ▶ [[Int8Array]]: Int8Array(93) [-8, 6, 9, 91, 75, 107, -4, 2, 49, 53, -8, 1, -8, 2, 52, -4, 76, 10, 52, 10]
  - ▶ [[Uint8Array]]: Uint8Array(93) [248, 6, 9, 91, 75, 107, 252, 2, 49, 53, 248, 1, 248, 2, 52, 252, 76, 10,
  - byteLength: (...)
- ▶ \_\_proto\_\_: ArrayBuffer
  - byteLength: (...)
  - constructor: f ArrayBuffer()
  - slice: f slice()
    - Symbol(Symbol.toStringTag): "ArrayBuffer"
  - get byteLength: f byteLength()
  - \_\_proto\_\_: Object
- ▶ e: ArrayBuffer(32) {}
- ▶ t: ArrayBuffer(32) {}

# Decrypt the outgoing data

## WHATSSAPP DECRYPTION AND ENCRYPTION EXTENSION BY DIKLA BARDA, ROMAN ZAIKIN

Ref object: `{"ref": "1@o1xj9nxF/ZFEL4PO0NgnQKTeMBzeChWM7VvFSgmvFBnR+yD3SL17+j", "wid": "1@o1xj9nxF/ZFEL4PO0NgnQKTeMBzeChWM7VvFSgmvFBnR+yD3SL17+j", "url": "http://www.google.com", "ip": "127.0.0.1", "port": 80, "type": "http", "status": "connected", "is_encrypted": false}`

Private Key: `[56, 181, 4, 127, 155, 134, 205, 206, 245, 18, 197, 18, 234, 160, 254, 237, 138, 196, 88, 156, 189, 12, 29, 88, 62, 156, 78, 177, 19, 42]`

Public Key: `[138, 224, 161, 129, 34, 113, 226, 100, 164, 130, 73, 68, 218, 64, 239, 183, 96, 123, 207, 100, 110, 33, 27, 131, 173, 172, 212, 5, 88, 115, 116, 117, 118, 119, 120, 121, 122, 123, 124, 125, 126, 127, 128, 129, 130, 131, 132, 133, 134, 135, 136, 137, 138, 139, 140, 141, 142, 143, 144, 145, 146, 147, 148, 149, 150, 151, 152, 153, 154, 155, 156, 157, 158, 159, 160, 161, 162, 163, 164, 165, 166, 167, 168, 169, 170, 171, 172, 173, 174, 175, 176, 177, 178, 179, 180, 181, 182, 183, 184, 185, 186, 187, 188, 189, 190, 191, 192, 193, 194, 195, 196, 197, 198, 199, 200, 201, 202, 203, 204, 205, 206, 207, 208, 209, 210, 211, 212, 213, 214, 215, 216, 217, 218, 219, 220, 221, 222, 223, 224, 225, 226, 227, 228, 229, 230, 231, 232, 233, 234, 235, 236, 237, 238, 239, 240, 241, 242, 243, 244, 245, 246, 247, 248, 249, 250, 251, 252, 253, 254, 255, 256, 257, 258, 259, 260, 261, 262, 263, 264, 265, 266, 267, 268, 269, 270, 271, 272, 273, 274, 275, 276, 277, 278, 279, 280, 281, 282, 283, 284, 285, 286, 287, 288, 289, 290, 291, 292, 293, 294, 295, 296, 297, 298, 299, 300, 301, 302, 303, 304, 305, 306, 307, 308, 309, 310, 311, 312, 313, 314, 315, 316, 317, 318, 319, 320, 321, 322, 323, 324, 325, 326, 327, 328, 329, 330, 331, 332, 333, 334, 335, 336, 337, 338, 339, 340, 341, 342, 343, 344, 345, 346, 347, 348, 349, 350, 351, 352, 353, 354, 355, 356, 357, 358, 359, 360, 361, 362, 363, 364, 365, 366, 367, 368, 369, 370, 371, 372, 373, 374, 375, 376, 377, 378, 379, 380, 381, 382, 383, 384, 385, 386, 387, 388, 389, 390, 391, 392, 393, 394, 395, 396, 397, 398, 399, 400, 401, 402, 403, 404, 405, 406, 407, 408, 409, 410, 411, 412, 413, 414, 415, 416, 417, 418, 419, 420, 421, 422, 423, 424, 425, 426, 427, 428, 429, 430, 431, 432, 433, 434, 435, 436, 437, 438, 439, 440, 441, 442, 443, 444, 445, 446, 447, 448, 449, 450, 451, 452, 453, 454, 455, 456, 457, 458, 459, 460, 461, 462, 463, 464, 465, 466, 467, 468, 469, 470, 471, 472, 473, 474, 475, 476, 477, 478, 479, 480, 481, 482, 483, 484, 485, 486, 487, 488, 489, 490, 491, 492, 493, 494, 495, 496, 497, 498, 499, 500, 501, 502, 503, 504, 505, 506, 507, 508, 509, 510, 511, 512, 513, 514, 515, 516, 517, 518, 519, 520, 521, 522, 523, 524, 525, 526, 527, 528, 529, 530, 531, 532, 533, 534, 535, 536, 537, 538, 539, 540, 541, 542, 543, 544, 545, 546, 547, 548, 549, 550, 551, 552, 553, 554, 555, 556, 557, 558, 559, 550, 551, 552, 553, 554, 555, 556, 557, 558, 559, 560, 561, 562, 563, 564, 565, 566, 567, 568, 569, 560, 561, 562, 563, 564, 565, 566, 567, 568, 569, 570, 571, 572, 573, 574, 575, 576, 577, 578, 579, 580, 581, 582, 583, 584, 585, 586, 587, 588, 589, 580, 581, 582, 583, 584, 585, 586, 587, 588, 589, 590, 591, 592, 593, 594, 595, 596, 597, 598, 599, 590, 591, 592, 593, 594, 595, 596, 597, 598, 599, 600, 601, 602, 603, 604, 605, 606, 607, 608, 609, 600, 601, 602, 603, 604, 605, 606, 607, 608, 609, 610, 611, 612, 613, 614, 615, 616, 617, 618, 619, 610, 611, 612, 613, 614, 615, 616, 617, 618, 619, 620, 621, 622, 623, 624, 625, 626, 627, 628, 629, 620, 621, 622, 623, 624, 625, 626, 627, 628, 629, 630, 631, 632, 633, 634, 635, 636, 637, 638, 639, 630, 631, 632, 633, 634, 635, 636, 637, 638, 639, 640, 641, 642, 643, 644, 645, 646, 647, 648, 649, 640, 641, 642, 643, 644, 645, 646, 647, 648, 649, 650, 651, 652, 653, 654, 655, 656, 657, 658, 659, 650, 651, 652, 653, 654, 655, 656, 657, 658, 659, 660, 661, 662, 663, 664, 665, 666, 667, 668, 669, 660, 661, 662, 663, 664, 665, 666, 667, 668, 669, 670, 671, 672, 673, 674, 675, 676, 677, 678, 679, 670, 671, 672, 673, 674, 675, 676, 677, 678, 679, 680, 681, 682, 683, 684, 685, 686, 687, 688, 689, 680, 681, 682, 683, 684, 685, 686, 687, 688, 689, 690, 691, 692, 693, 694, 695, 696, 697, 698, 699, 690, 691, 692, 693, 694, 695, 696, 697, 698, 699, 700, 701, 702, 703, 704, 705, 706, 707, 708, 709, 700, 701, 702, 703, 704, 705, 706, 707, 708, 709, 710, 711, 712, 713, 714, 715, 716, 717, 718, 719, 710, 711, 712, 713, 714, 715, 716, 717, 718, 719, 720, 721, 722, 723, 724, 725, 726, 727, 728, 729, 720, 721, 722, 723, 724, 725, 726, 727, 728, 729, 730, 731, 732, 733, 734, 735, 736, 737, 738, 739, 730, 731, 732, 733, 734, 735, 736, 737, 738, 739, 740, 741, 742, 743, 744, 745, 746, 747, 748, 749, 740, 741, 742, 743, 744, 745, 746, 747, 748, 749, 750, 751, 752, 753, 754, 755, 756, 757, 758, 759, 750, 751, 752, 753, 754, 755, 756, 757, 758, 759, 760, 761, 762, 763, 764, 765, 766, 767, 768, 769, 760, 761, 762, 763, 764, 765, 766, 767, 768, 769, 770, 771, 772, 773, 774, 775, 776, 777, 778, 779, 770, 771, 772, 773, 774, 775, 776, 777, 778, 779, 780, 781, 782, 783, 784, 785, 786, 787, 788, 789, 780, 781, 782, 783, 784, 785, 786, 787, 788, 789, 790, 791, 792, 793, 794, 795, 796, 797, 798, 799, 790, 791, 792, 793, 794, 795, 796, 797, 798, 799, 800, 801, 802, 803, 804, 805, 806, 807, 808, 809, 800, 801, 802, 803, 804, 805, 806, 807, 808, 809, 810, 811, 812, 813, 814, 815, 816, 817, 818, 819, 810, 811, 812, 813, 814, 815, 816, 817, 818, 819, 820, 821, 822, 823, 824, 825, 826, 827, 828, 829, 820, 821, 822, 823, 824, 825, 826, 827, 828, 829, 830, 831, 832, 833, 834, 835, 836, 837, 838, 839, 830, 831, 832, 833, 834, 835, 836, 837, 838, 839, 840, 841, 842, 843, 844, 845, 846, 847, 848, 849, 840, 841, 842, 843, 844, 845, 846, 847, 848, 849, 850, 851, 852, 853, 854, 855, 856, 857, 858, 859, 850, 851, 852, 853, 854, 855, 856, 857, 858, 859, 860, 861, 862, 863, 864, 865, 866, 867, 868, 869, 860, 861, 862, 863, 864, 865, 866, 867, 868, 869, 870, 871, 872, 873, 874, 875, 876, 877, 878, 879, 870, 871, 872, 873, 874, 875, 876, 877, 878, 879, 880, 881, 882, 883, 884, 885, 886, 887, 888, 889, 880, 881, 882, 883, 884, 885, 886, 887, 888, 889, 890, 891, 892, 893, 894, 895, 896, 897, 898, 899, 890, 891, 892, 893, 894, 895, 896, 897, 898, 899, 900, 901, 902, 903, 904, 905, 906, 907, 908, 909, 900, 901, 902, 903, 904, 905, 906, 907, 908, 909, 910, 911, 912, 913, 914, 915, 916, 917, 918, 919, 910, 911, 912, 913, 914, 915, 916, 917, 918, 919, 920, 921, 922, 923, 924, 925, 926, 927, 928, 929, 920, 921, 922, 923, 924, 925, 926, 927, 928, 929, 930, 931, 932, 933, 934, 935, 936, 937, 938, 939, 930, 931, 932, 933, 934, 935, 936, 937, 938, 939, 940, 941, 942, 943, 944, 945, 946, 947, 948, 949, 940, 941, 942, 943, 944, 945, 946, 947, 948, 949, 950, 951, 952, 953, 954, 955, 956, 957, 958, 959, 950, 951, 952, 953, 954, 955, 956, 957, 958, 959, 960, 961, 962, 963, 964, 965, 966, 967, 968, 969, 960, 961, 962, 963, 964, 965, 966, 967, 968, 969, 970, 971, 972, 973, 974, 975, 976, 977, 978, 979, 970, 971, 972, 973, 974, 975, 976, 977, 978, 979, 980, 981, 982, 983, 984, 985, 986, 987, 988, 989, 980, 981, 982, 983, 984, 985, 986, 987, 988, 989, 990, 991, 992, 993, 994, 995, 996, 997, 998, 999, 990, 991, 992, 993, 994, 995, 996, 997, 998, 999, 1000, 1001, 1002, 1003, 1004, 1005, 1006, 1007, 1008, 1009, 1000, 1001, 1002, 1003, 1004, 1005, 1006, 1007, 1008, 1009, 1010, 1011, 1012, 1013, 1014, 1015, 1016, 1017, 1018, 1019, 1010, 1011, 1012, 1013, 1014, 1015, 1016, 1017, 1018, 1019, 1020, 1021, 1022, 1023, 1024, 1025, 1026, 1027, 1028, 1029, 1020, 1021, 1022, 1023, 1024, 1025, 1026, 1027, 1028, 1029, 1030, 1031, 1032, 1033, 1034, 1035, 1036, 1037, 1038, 1039, 1030, 1031, 1032, 1033, 1034, 1035, 1036, 1037, 1038, 1039, 1040, 1041, 1042, 1043, 1044, 1045, 1046, 1047, 1048, 1049, 1040, 1041, 1042, 1043, 1044, 1045, 1046, 1047, 1048, 1049, 1050, 1051, 1052, 1053, 1054, 1055, 1056, 1057, 1058, 1059, 1050, 1051, 1052, 1053, 1054, 1055, 1056, 1057, 1058, 1059, 1060, 1061, 1062, 1063, 1064, 1065, 1066, 1067, 1068, 1069, 1060, 1061, 1062, 1063, 1064, 1065, 1066, 1067, 1068, 1069, 1070, 1071, 1072, 1073, 1074, 1075, 1076, 1077, 1078, 1079, 1070, 1071, 1072, 1073, 1074, 1075, 1076, 1077, 1078, 1079, 1080, 1081, 1082, 1083, 1084, 1085, 1086, 1087, 1088, 1089, 1080, 1081, 1082, 1083, 1084, 1085, 1086, 1087, 1088, 1089, 1090, 1091, 1092, 1093, 1094, 1095, 1096, 1097, 1098, 1099, 1090, 1091, 1092, 1093, 1094, 1095, 1096, 1097, 1098, 1099, 1100, 1101, 1102, 1103, 1104, 1105, 1106, 1107, 1108, 1109, 1100, 1101, 1102, 1103, 1104, 1105, 1106, 1107, 1108, 1109, 1110, 1111, 1112, 1113, 1114, 1115, 1116, 1117, 1118, 1119, 1110, 1111, 1112, 1113, 1114, 1115, 1116, 1117, 1118, 1119, 1120, 1121, 1122, 1123, 1124, 1125, 1126, 1127, 1128, 1129, 1120,`

# Decrypt the outgoing data

WHATSAPP DECRYPTION AND ENCRYPTION EXTENSION BY DIKLA BARDA, ROMAN ZAIKIN

Ref object: [{"ref": "1@o1xj9nixF/ZFEL4PO0NqntQKTeMBzeChWM7VvFSgmvFBnR+yD3SL17+J", "wid": "1@c.us", "connected": true, "id": 1}],

Private Key: [56, 181, 4, 127, 155, 134, 205, 206, 245, 18, 197, 18, 234, 160, 254, 237, 138, 196, 88, 156, 189, 12, 29, 88, 62, 156, 78, 177, 19, 42]

Public Key: [138, 224, 161, 129, 34, 113, 226, 100, 164, 130, 73, 68, 218, 64, 239, 183, 96, 123, 207, 100, 110, 33, 27, 131, 173, 172, 212, 5, 88,

Connect

Clear

```
[{"action": {"epoch": "15", "type": "relay"}, [{"message": {"conversation": "I did \ud83d\ude13", "messageTimestamp": "1533038302", "key": {"fromMe": true, "remoteJid": "[REDACTED]@s.whatsapp.net", "id": "3EB05D7090ABEBECF381"}, "status": "PENDING"}]]
```

Incoming

Encrypt

## Decrypt

## Outgoing

CONNECTION STATUS: CONNECTED

ACTION STATUS: OK



# DEMO

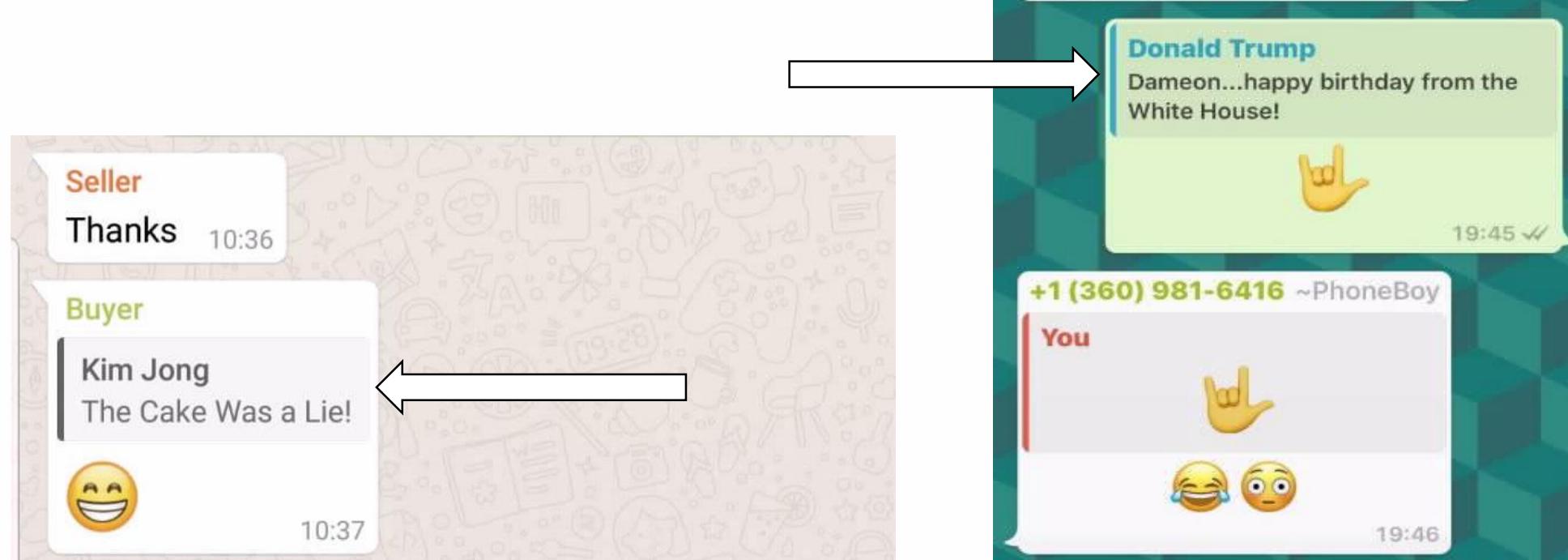


## Manipulation #1 – fake reply from someone in the group



Demo

## Manipulation #2 – Fake reply to someone not in the group

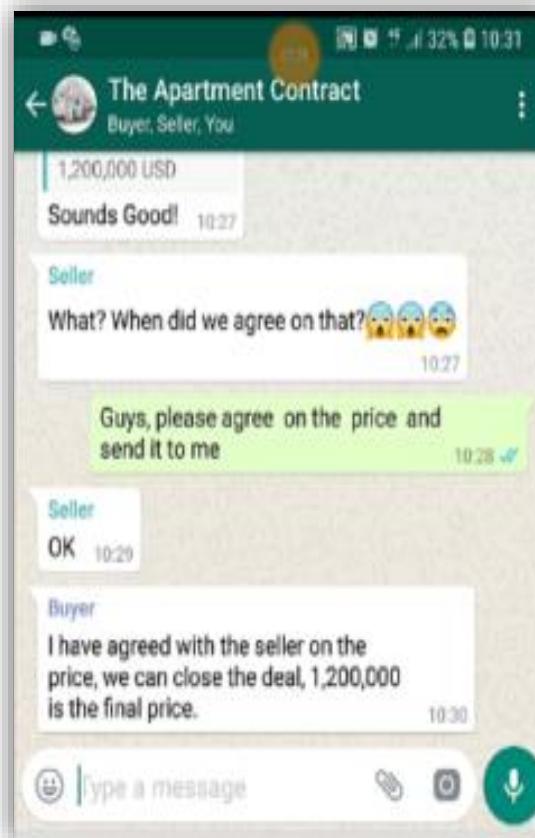


## Manipulation #3 – Send a private message in group chat to a specific person

Attacker



User 1

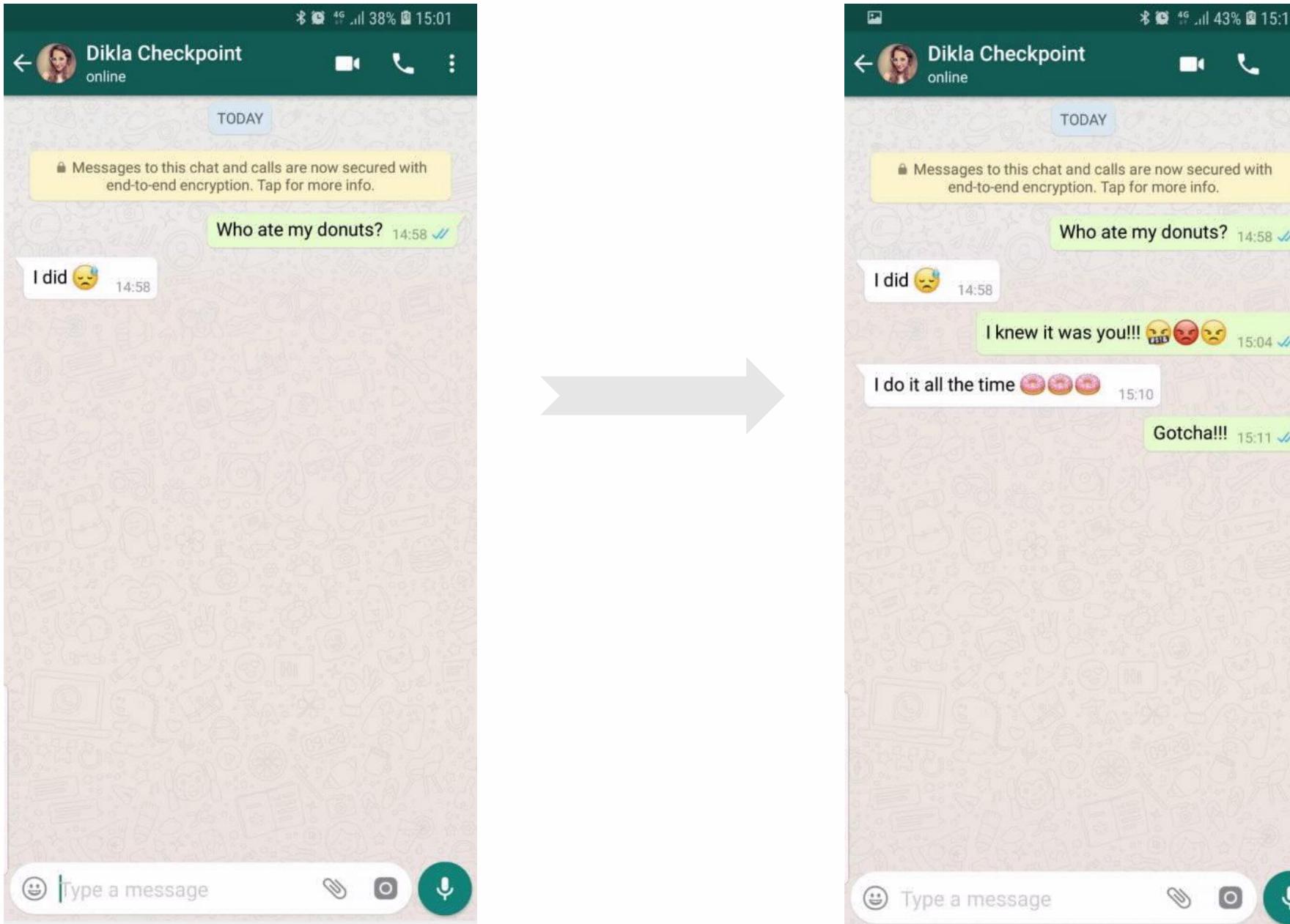


User 2



Demo

# Manipulation #4: send messages to myself



Demo



AUGUST 3-8, 2019  
MANDALAY BAY / LAS VEGAS

THANK YOU!