Mastering EDL Mode
By Scott Lorenz – Chief Forensic Analyst at Centex Technologies
12-11-2018
Edited by Andrew Rathbun

Document Purpose: To describe the various methods, techniques, and tips for placing devices in EDL Mode and extracting them using the UFED. This document is designed to change and update as needed.

General Overview: This document is designed to be a general reference for placing devices in EDL Mode and using Cellebrite's UFED 4PC and UFED Touch to extract the devices using decrypting and non-decrypting options provided in UFED extraction software. All tests were conducted using either the UFED 4PC and/or UFED Touch 2. Advanced methods of device extraction, utilizing Cellebrite's EDL exploit, are also discussed.

For more information on EDL extractions, detailed diagrams, and advanced information on EDL test points, see Mastering EDL Test Points.

Requesting Access to Mobile Device Forensics and Analysis forum: When requesting access or membership in the forum, please include information about who you are (name), what agency you are from or your business or company, and your interests in the forum or purpose. That information in your initial request will save time and prevent forum moderators from having to send requests for more information.

Navigating This Document: First of all, Ctrl+F is your friend! Each section of the document is a bookmark. Additionally, throughout this document there are links to photographs, diagrams, and other documents. Links may be in the text of the document or by clicking on a photograph or diagram. It may be necessary to download this PDF in order for those links to work properly as opposed to just viewing it in your web browser. The links are to photos and documents in the resources folder of the **Mobile Device Forensics and Analysis forum**. The Links in the Table of Contents lead to that corresponding section of the document.

Continual Updates: I am continually making updates to this document. That includes minor errors, changes in support or function of the UFED, and general formatting. **Special Thanks:** Thanks to Detective Andrew Rathbun from the Michigan State University Police Department for editing this document.

1 Overview of EDL Mode 6
1.1.1 Current information on EDL and UFED capabilities 6
1.1.2 EDL Research and Development 6
1.1.3 Do any other vendors use EDL? 6
1.1.4 Why use EDL for that Extraction? 6
1.1.5 What is EDL Mode? 7
1.1.6 Can all Qualcomm devices placed in EDL Mode be extracted? 7
1.1.7 Why doesn't Cellebrite use the EDL exploit on older phones running Qualcomm processors? 7
1.1.8 Does it harm devices to place them in EDL Mode? 7
1.1.9 Can EDL bypass user passcodes and pattern locks? 8
2 What is the difference between the decrypting and non-decrypting methods of EDL extractions with the UFED? 9
2.1 Non-decrypting Method 9
2.1.1 EDL works well even if the device doesn't 9
2.1.2 Why remove the battery? 9
2.2 Decrypting Method 10
2.2.1 Decrypting extractions and device booting 10-20-18 11
2.2.1.1 I used EDL to get a physical extraction and my user data is encrypted. Why? 11
2.2.1.2 Will other vendors or methods of using EDL for extractions work on encrypted devices? 12-10-18 11
2.2.1.3 Booting devices during Cellebrite's decrypting EDL extractions 11
2.2.1.4 If my device is specifically supported under its device profile in the UFED, why does Cellebrite make me choose decrypting or non-decrypting? 11
2.2.2 What if I extract an encrypted device using the non-decrypting method? 12
2.2.3 What happens if I extract a device that is NOT encrypted with the decrypting bootloader? 12
2.3 Secure Startup and EDL 13
2.3.1 How do you know if a device has Secure Startup? 13
2.3.2 Does Secure Startup mean the device is encrypted? 13
2.3.3 How do you know if Secure Startup is enabled on a running device? 13
2.3.4 What happens if I try to extract a device with Secure Startup using the EDL method? 13
2.3.5 Should a device be left running when seized to avoid Secure Startup? 14
2.3.6 If the Secure Startup passcode or pattern is known, must the Secure Startup pattern or passcode be removed for a successful decrypting EDL extraction in the UFED? (8-30-18) 14
2.3.6.1 Secure Startup with known passcode – UFED 4PC 14
2.3.6.2 Secure Startup with known passcode/pattern – UFED Touch2 (ZTE Z971) 8-30-18 15
3 Device intake and testing for EDL Mode 15
3.1 Identify the device 15
3.1.1 UFED Direct Support 15
3.1.2 Devices not specifically identified as supported in the UFED 15

3.1.2.1 Check the Device Processor (7-17-18) 15
3.1.3 Which Qualcomm processors can the UFED exploit via EDL Mode? (7-17-18) 16
3.1.3.1 Check for device encryption 16
3.2 Identifying devices that are likely encrypted or not encrypted 16
3.2.1 UFED's EDL exploit works on encrypted devices 17
3.2.2 Not all EDL extractions decrypt data 17
3.2.3 Why does the UFED allow me to perform an extraction of an encrypted device with a non-decrypting method? (11-5-18) 17
3.2.4 Other Decrypting and Non-Decrypting Extractions 17
3.2.4.1 Low-level physical extractions 18
3.2.4.2 JTAG, ISP, and Chip-Off do not decrypt data 18
3.2.4.2.1 JTAG 19
3.2.4.2.2 ISP and EDL 21
3.2.4.2.3 Chip-Off 22
3.2.5 Device encryption based on Android version 22
3.2.5.1 What is Android? 22
3.2.5.2 Where are the rules about Android encryption found? 22
3.2.5.3 Are Some OEMs violating the rules of Android? 23
3.2.6 Device encryption based on manufacture date and UFED support 23
3.2.7 Device encryption based on processors 24
3.2.7.1 The MSM8909 is the center of the encryption universe 25
3.2.8 Device encryption based on storage type 25
3.2.9 Device encryption based on storage size 26
3.2.10 Device encryption based on price 26
3.2.11 Devices running Android (Go edition) 26
3.2.12 Comparison of AT&T Prepaid Phones at Walmart 27
3.2.13 Examples of devices running Android 6 and above that are NOT encrypted 28
3.2.14 Android 9 Pie (Go edition) 28
3.2.15 Using the Android Debug Console in the UFED to determine if a device is encrypted 29
3.3 Methods for placing devices in EDL Mode 29
4 Testing, creating, and implementing EDL Mode 30
4.1 How do I check to see if the device is in EDL Mode? 30
4.2 Why would I test EDL in Device Manager before extracting a device? 30
4.2.1 EDL Mode is not the only event or condition that will trigger a handshake 30
4.3 UFED Android Debug Console - UFED version 7.10.1.1080 31
4.3.1 Using UFED Android Debug Console to determine extraction options 32
4.3.2 Using Android Debug Console to display device information - UFED version 7.10.1.1080 32
4.3.3 Using Android Debug Console to create and detect EDL Mode - UFED version 7.10.1.1080 32

4.3.4 Using Android Debug Console to create and detect FTM - UFED version 7.10.1.1080 34
4.3.5 Using Android Debug Console to create and detect DFU Mode - UFED version 7.10.1.1080 34
4.4 EDL Cables 35
4.4.1 Using the EDL Cable ”shorts” the device but it is not the same as creating eMMC faults 36
4.5 Button Combinations 37
4.5.1 What do button combinations really do? 37
4.6 Automated ADB via UFED’s direct support or generic support with unlocked devices 37
4.7 Manual ADB or Fastboot used by the examiner to create EDL 38
4.8 Devices not supported for entering EDL via ADB or Fastboot commands 38
4.9 FTM and DFU Mode to create EDL 39
4.9.1 ZTE Z835 using DFU Mode 39
4.9.2 How do I know if my device is in FTM or DFU Mode? 39
4.9.3 FTM behaves differently and requires separate menu selections than DFU and EDL 40
4.9.4 FTM method for generic EDL extractions 40
4.9.5 Why doesn’t Cellebrite just tell me what button to push all the time? 41
4.9.6 Shorting a device into DFU Mode 41
4.10 Cellebrite’s LG EDL method to extract data from LG devices 42
4.10.1 The UFED taking advantage of LG Advanced Flash (LAF) to create EDL Mode 42
4.10.2 Why is it necessary to move from LAF to EDL if LAF will dump the phone? 43
4.10.3 The new EDL method for LG phones is not found under generic options 43
4.11 LG EDL Recovery tool 43
4.12 Test points to create EDL Mode 44
4.12.1 Looking and probing for Test Points 45
4.12.2 USB Taps are not EDL Test Points 45
4.12.3 LG phones and Test Points – look for the sign of the cross 46
4.12.4 Test Points are not grounded 46
4.12.5 Non-EDL Test Points 46
4.12.6 Compare similar LG boards that run Qualcomm and MediaTek Processors 47
4.12.7 LG Test Points on phones with the same name and different Qualcomm processors 47
4.12.8 LG Test Points on phones running MediaTek processor, what do they do? 47
4.12.9 Which is easier – eMMC shorting or Test Points? 48
4.12.10 List of sample phones with Test Points – links to diagrams included 48
4.13 Creating eMMC faults (shorting) 48
4.14 Removing the short before extraction 49
4.15 Testing for solder bridges or other damage that creates permanent EDL Mode 49
4.16 Phone disassembly 50
4.16.1 Shorting ISP points 50

4.16.2 Why ISP, JTAG, and Chip-Off training and techniques are still relevant 50
4.16.3 Pinning devices and locating EDL short points 51
4.16.4 Finding ISP points without chipping a test device 52
4.16.5 Which ISP points create EDL? 52
4.16.6 What is the difference between the ISP data lines and data (D+) line in a USB cable? 52
4.16.7 Is soldering necessary? 53
4.16.8 Shorting Methods - Diagrams: 54
4.17 Shorting and pinning devices with Universal Flash Storage (UFS) 54
4.17.1 Samsung Galaxy S7 UFS BGA153 with MSM8996 55
4.18 Reverse pinning processors and UFS to locate short points 55
5 Troubleshooting EDL extractions and advanced methods made possible by the EDL exploit 55
5.1 Attention to detail 56
5.1.1 Check for EDL and check that EDL can be removed 56
5.2 Preparation 56
5.3 Get test devices 56
5.4 Common extraction errors and error messages (10-12-18) 57
5.4.1 Error # 1 - Device not in EDL Mode when the extraction starts (10-11-18) 57
5.4.2 Improper use of Cable #523 fails to create EDL Mode and causes the same USB Debugging message (10-12-18) 58
5.4.3 Error # 2 - Asking the UFED to create EDL Mode but failing to enable USB Debugging (10-12-18) 59
5.4.4 Error # 3 – Device in EDL Mode but not supported for extraction by the UFED (10-12-18) 60
5.4.5 Error # 4 – Supported devices extracting via non-decrypting EDL method but fails with decrypting EDL method (10-12-18) 61
5.5 Extractions of badly damaged, encrypted devices using the EDL exploit 61
5.5.1 Using Cellebrite's EDL exploit on encrypted devices that don't function 62
5.5.2 Damaged devices 62
5.5.3 Warning about advanced procedures 62
5.5.4 Damaged Devices - Bypassing and Surgery Diagrams and Videos 62
6 Links to Documents and Information on EDL in this Paper 63
7 Links to Original How-to Videos – Narrated (These are longer videos – may need to download to avoid audio syncing issues) 64
8 2nd EDL Webinar Videos – 9-12-18 64
9 Android Debug Console Extractions 65
10 Example of devices sold with Android 6 or higher that were not encrypted by default 65
11 Bibliography 65

1 Overview of EDL Mode

For those not familiar with EDL Mode and specifically Cellebrite's use of EDL Mode, I recommend you first read Shahar Tal's "*Practical Guide for Qualcomm EDL Physical Extractions*". That PDF document is located in the Resources Folder/EDL Extractions, on the Mobile Device Forensics and Analysis group. In that same folder is the PDF version of the "Cellebrite EDL Webinar 2-21-18". That will also give a general overview of EDL extractions – via Cellebrite's UFED. The 2nd Cellebrite EDL Webinar 9-12-18, is in PowerPoint format with links to videos of extractions. It is always best to refer to the source (developers) of forensic techniques and tools for the most accurate information. Although I was a presenter in the 1st and 2nd seminar, it did not and could not cover all techniques, methods, and abilities of EDL extractions. So I hope this document and its linked supplements will fill in some of the gaps.

1.1.1 Current information on EDL and UFED capabilities

In the world of mobile forensics something will change five minutes after publication. I will date certain portions of this paper or otherwise identify particular versions of software so that modifications can be made when updates are made or support for devices or processors changes. There are links to diagrams and flow charts and step-by-step guides on EDL extractions. Those will likely change with future UFED releases and changes to device manufacturer's hardware and software. I will try to make modifications or at least date items so users can assess whether the information is still correct or relevant.

1.1.2 EDL Research and Development

Examiners need vendors with heavy R&D to continue doing what they are doing. We want vendors like Cellebrite to pass tools down to those who will use them legally and responsibly. The exploit provided by Cellebrite is a powerful tool for many reasons and probably for some reasons that many examiners may not have considered. Cellebrite's EDL exploit can be used by inexperienced users but it also allows experienced users the ability to perform some advanced maneuvers to reach encrypted data that would normally be off limits. So while we want access to more devices from companies like Cellebrite, we also have to do research and experimentation to be able to get the most out of exploits like EDL. Methods for getting devices into EDL Mode vary significantly across the universe of phones.

1.1.3 Do any other vendors use EDL?

Cellebrite is not the only vendor that utilizes EDL Mode as an exploit. Other vendors and examiners use EDL Mode as a tool to extract data from devices using other techniques and software. This document is not designed to review or cover the many different vendors, software, or hardware in the world that make use of EDL Mode even though I have used some. I do believe Cellebrite stands apart in their use of the EDL exploit for many reasons, but especially their ability to perform decrypting extractions.

The techniques I use and describe in this document are designed to be a general overview of the various aspects of EDL Mode and working with various devices using Cellebrite UFED software. There is often more than one way to achieve EDL Mode and often more than one way to pull the same device using UFED software via EDL methods.

1.1.4 Why use EDL for that Extraction?

I try to use the simplest, safest, most effective technique available when dealing with evidence from actual cases. There is often more than one way to extract a physical image from a particular device. EDL Mode may or may not be the easiest method to accomplish the same task with that device. This paper is a demonstration of Cellebrite's use of EDL – limited to my ability with it. EDL is another tool and another method to add to your inventory of tools. So for purposes of experimentation and demonstration, in this paper I sometimes intentionally pick the most difficult method to perform a procedure just to demonstrate and show that it will or will not work. I also intentionally break devices to demonstrate what is possible with the EDL exploit.

1.1.5 What is EDL Mode?

Shahar Tal describes EDL simply in his Practical Guide in which he refers to a feature of devices running Qualcomm processors known as (Emergency Download) Mode, "…designed to allow low-level access to the chipset for device analysis, repair or re-flashing." EDL Mode was not developed by Cellebrite. EDL is a feature of Qualcomm processors that Cellebrite and other vendors exploit in order to extract data from the device. Like every other method for extracting data from mobile devices, achieving EDL and exploiting EDL varies greatly from device to device. For forensic tools, the EDL exploit also varies greatly from vendor to vendor. Although I am writing about Cellebrite's use of EDL, I have tried other vendors for EDL. My use of those other vendors has not been what I would call "exhaustive" as with my use of Cellebrite, but the significant difference in vendors that is very obvious is Cellebrite's ability to decrypt using their EDL exploit. Just because other vendors or methods can extract data by exploiting EDL does not mean that extraction will be decrypted.

1.1.6 Can all Qualcomm devices placed in EDL Mode be extracted?

No. Based on questions I have received on EDL, there may have been some initial confusion regarding devices that can be placed in EDL Mode versus devices that can be extracted using Cellebrite's exploit of certain devices having been placed in EDL Mode. Just because a device can be placed in EDL Mode doesn't mean it can be extracted using Cellebrite's EDL method.

Most devices running Qualcomm processors can be placed (forced) into EDL Mode. Cellebrite can take advantage of EDL Mode (extract the device) only on specific devices or on a wide range of devices running a specific model of Qualcomm processor. For example, most devices running the Qualcomm MSM8909 can be decrypted and extracted by Cellebrite using their EDL exploit. Alternatively, for example, I can place most devices running the Qualcomm MSM8210 in EDL Mode. However, to my knowledge, none of them can be extracted using Cellebrite's EDL exploit. Of course, there may be another vendor or method for exploiting a particular device or processor using EDL, but I am focusing on the method provided by Cellebrite's UFED.

1.1.7 Why doesn't Cellebrite use the EDL exploit on older phones running Qualcomm processors?

I personally get asked this question frequently. Of course, I can't speak for Cellebrite, but I will take some liberties based my modest experience with mobile forensics. Cellebrite was asked this question when they first released their EDL exploit. First, what Cellebrite has done and is doing with the EDL exploit is difficult. We sometimes take for granted the ability to push a button and extract decrypted data from a locked device. Cellebrite's EDL exploit requires a lot of R&D as does many of their other exploits. Devoting time and R&D manpower to breaking software from the past is not always practical or possible, especially when there are many working exploits already in place on those older devices.

1.1.8 Does it harm devices to place them in EDL Mode?

I always tell my Criminal Justice students not to use absolutes when dealing with the law. I therefore fear using absolutes when talking about digital forensics, but the general answer to the question "Does EDL damage a device?" is no. I say that relying on Cellebrite's expertise and my modest experience in placing hundreds of devices in EDL Mode using many different methods. I have some individual test devices that I have placed in EDL Mode easily over 100 times, mostly by shorting them, with no adverse effect.

Please do not misinterpret my statement to mean that no harm can come to devices by attempting to place them in EDL Mode. In my experience with EDL, it is the method of creating EDL that harms devices and not EDL Mode. It is most often user error that damages a device. One of the reasons EDL Mode was created was for when something goes wrong during the booting process. EDL is often the result of "something wrong". Shorting out a mobile phone is violating the terms of any warranty and most cell phone manufacturers and service providers would say it is dangerous.

About using eMMC shorting to force phones into EDL Mode, shorting the wrong location can damage a phone permanently. Disassembly of phones can damage phones so that they may not be extracted or turned on again.

Likewise, using ISP and JTAG techniques can be performed repeatedly on the same device without affecting it, or those techniques can be done the wrong way once and destroy it.

Having training in ISP or JTAG and experience soldering is very beneficial when shorting devices into EDL Mode. Using a microscope is highly recommended. Experience with device repair and disassembly is also very helpful. Knowing where to short is accomplished by training and practicing all of those disciplines. This is one of the many reasons why ISP, JTAG, and Chip-Off training is still essential. In my opinion, those classes will benefit you in many ways even in a world of increasing encryption. For more information on JTAG, Chip-Off and ISP see Section 3.2.4.2 in this document.

1.1.9 Can EDL bypass user passcodes and pattern locks?

Please keep in mind that user passcodes and pattern locks discussed here are not the same as Secure Startup. Secure Startup will be discussed in another section of this paper. Yes, Cellebrite's EDL exploit works on devices with user passcode and pattern locks. That includes the decrypting method and the non-decrypting method of extraction. With the non-decrypting EDL extraction, booting is not required. The non-decrypting method is a low-level extraction that, although takes place via processor exploits, is very much like an ISP extraction.

The same is true with the non-decrypting EDL extraction. Cellebrite is using EDL Mode to access and exploit the processor and then command it to dump the phone's storage through the USB port, as opposed to ISP which extracts data directly from the storage. With the non-decrypting EDL method, this is done pre-boot, which is why extracting data from devices that are not encrypted can be done on phones that are damaged and can't boot. It is this reason that EDL is something I push to be used on damaged devices, especially devices that are not encrypted.

With devices that are encrypted, Cellebrite will use the EDL exploit to force the phone to boot and apply their decrypting bootloader. The device will boot into Android and to the user screen just as if you powered it on yourself. The user passcode or pattern lock is still in place and you can interact with the screen. However, because the decrypting bootloader has been applied, the fact that the phone is still locked doesn't matter. Using the decrypting EDL method in the UFED means the phone must boot, locked or not.

The same is true for using ADB to create EDL, either having the UFED do it automatically or the examiner doing it through Command Prompt. It is necessary to know the passcode and unlock the phone in order to access Developer Options and USB Debugging, but there is no need to remove the user passcode or pattern for the decrypting extraction to work once the settings are in place.

2 What is the difference between the decrypting and non-decrypting methods of EDL extractions with the UFED?

Aside from the obvious answer that one decrypts data and the other doesn't, for the examiner, the most important answer to that question is: the process used by Cellebrite. I do not have the knowledge, access, or expertise to do anything but describe this in general terms. The first thing I tell anyone who asks is that, generally, the non-decrypting method is much easier. By easier, I mean there are fewer steps involved, no need for the phone to boot, and no requirement to create EDL more than once during a single extraction.

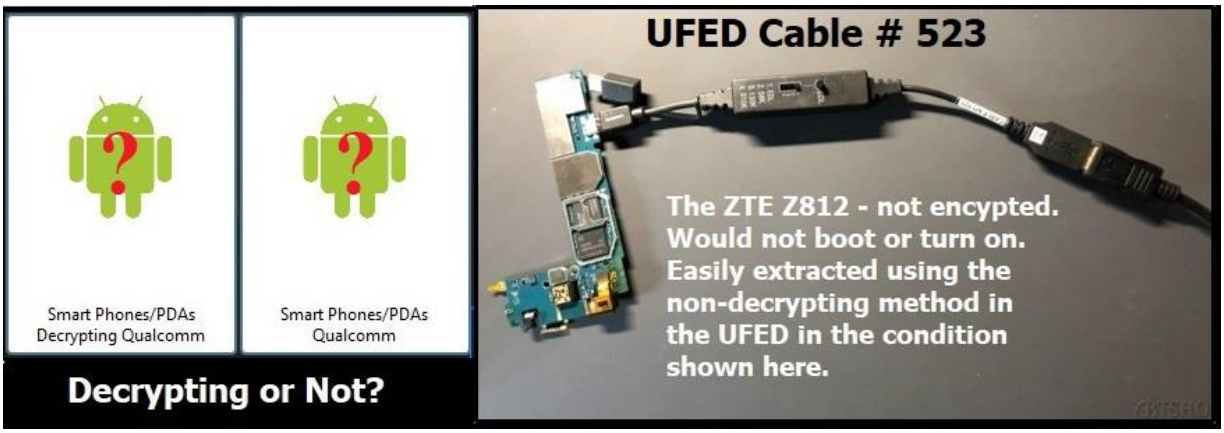


Figure 1 - Decrypting or Not - ZTE Z812

Based on the questions I frequently receive, both on and off the forum, the non-decrypting method is much easier than everyone imagines. Devices don't have to boot or power on at all for the non-decrypting procedure. Before Cellebrite released their EDL exploit, there were many devices that required ISP or Chip-Off in order to obtain a physical image. With the EDL exploit in the UFED, I can extract many of those same devices in about 20 minutes without a battery, without ever turning the phone on, and without soldering or disassembly – depending on the device.

2.1 Non-decrypting Method

On devices that are not encrypted, using the non-decrypting method, either under a supported device profile, or by selecting "Smart Phones/PDAs Qualcomm" will be a quick and easy, low-level extraction that will not require the device to boot or even function. This method can be performed on badly damaged devices and in most cases does not require a battery.

2.1.1 EDL works well even if the device doesn't

When I get a question about extraction possibilities for damaged devices, not only is EDL a consideration for me, it is often my first consideration. Using EDL to extract devices that are not encrypted requires very little activity or functionality from the phone – no battery, no booting, and no screen interaction. Although the process is not the same for encrypted devices, Cellebrite's EDL method still works well on damaged devices that are encrypted. I have devoted a section for damaged devices at the end of this paper but felt it was worth mentioning here.

2.1.2 Why remove the battery?

When I say, "does not require a battery for devices that are not encrypted", I mean most of the time the battery is just in my way for this extraction. I don't need it. I don't need the phone to boot and I don't want the phone to boot. Booting means the device can start up and thus I must consider all of the implications of starting a device that was seized in unknown condition or may have been shut down and stored without activating Airplane Mode. I don't have to worry about charging a battery. Devices that will not boot at all can be extracted like this. Multiple extractions without the battery will not change the hash. Any time I can retrieve evidence from a device without

booting the device, I will use that method – either exclusively or before other extractions that require that device to boot.

All that is needed is a working USB port, although I have been able to extract many devices without a working USB port (discussed later). Of course, if the device is in the same condition as depicted with ***Figure 1***, the Z812 photo (board only), eMMC shorting or the EDL cable will be needed to create EDL Mode. Working devices are needed for creating EDL via ADB and generally, working devices are needed for some button combos that create EDL. There are some exceptions for button combos to create FTM and DFU Modes (discussed later).

In terms of the effect on the image hash, EDL (non-decrypting) is similar to extracting a device via Chip-Off or ISP. The phone is not required to boot and thus the hash value of the image will not be affected even with multiple pulls (provided the phone is not booted between pulls). On devices I have tested with this method, multiple pulls did not change the image hash.

The non-decrypting method is procedurally the easiest of the two methods. The vast majority of devices I pull with this method are done without the battery and many times with only the logic board connected to USB. With few exceptions, a power button is not needed. After creating EDL, the device will go through two steps and immediately start to dump. If you have a device that is not encrypted, the non-decrypting selection is the method you should use. I have created a workflow chart of steps for using the non-decrypting EDL method in the UFED. (EDL Non-decrypting Pulling Steps) With the addition of the UFED's Android Debug Console in UFED version 7.10.1.1080, there is also another workflow for available for decrypting and non-decrypting methods.

2.2 Decrypting Method

On devices that are encrypted, select "Smart Phones/PDAs Decrypting Qualcomm" method. This method requires the phone to boot and therefore is not as easily deployed as the non-decrypting method. With few exceptions, devices require a battery for this method and the phone must be in general working order. Most mobile phones require a battery to boot and operate, even with USB power. But as with everything, there are exceptions to that.

The decrypting EDL extraction can generally be done without a working screen or without the screen attached at all because there is no user interaction necessary on the device screen – unless using ADB decrypting method (discussed later). Some devices require the screen to be attached to boot. There is also another exception to the general rule regarding no screen interaction. That exception involves a device with Secure Startup with the passcode known to the examiner (also discussed later).

When referring to write-ups on specific phone models, please keep in mind that Cellebrite makes continual changes and improvements to their software. So write-ups by me or others may not work the same way or be necessary due to an improved version of the UFED. I have seen this manifest itself with steps that are no longer required that once were.

A working power button may be necessary for this method to work. You may be prompted to place the device in EDL Mode more than one time in the same procedure. In this procedure, the phone will go through 6 steps and then an "executing" phase. A second request for EDL Mode, if requested, will occur in Step 4. The hash value of the image will be different if more than one extraction is made. You will see that there are several options for creating EDL Mode for a decrypting extraction. Which option you select can affect whether the device will extract. With some phones, the procedure is very specific. This is because of the timing in which the UFED causes the device to boot with the decrypting bootloader. I have created specific write-ups on certain models known to have this issue. On many other models, I have identified a procedure that works most often or is most reliable. But all of that is a moving target so I am constantly changing, adapting, and ready to apply multiple methods to one device.

Some devices will work with many different methods or will take care of themselves after the first EDL Mode is created. If a second request for EDL is made during this procedure, the method I use most often is 11A, depicted in my decrypting EDL diagram. 11A works on many different brands and models of phones when using eMMC

shorting (EDL Decrypting Pulling Steps). But my method and techniques are not the only way and may not be the best, or they may have been the best and now they are not needed. It's digital forensics, things will change.

2.2.1 Decrypting extractions and device booting 10-20-18

2.2.1.1 I used EDL to get a physical extraction and my user data is encrypted. Why?

This question comes up occasionally and the answer is usually the same for all scenarios. The answer is usually because the device was extracted in the off-state. Devices extracted in the off state to include Cellebrite's EDL non-decrypting method, are being extracted via a low-level process that just dumps the data from the phone's storage, either eMMC or UFS storage. If you extracted the device using Cellebrite's non-decrypting method in the UFED, and the user data was encrypted, extract the device again using the decrypting method. The decrypting method boots the phone and applies the decrypting bootloader.

2.2.1.2 Will other vendors or methods of using EDL for extractions work on encrypted devices? 12-10-18

There are other vendors that use EDL Mode as a method of accessing and dumping the device's storage. To my knowledge, Cellebrite is the only vendor that can decrypt data accessed via the EDL exploit. That can change in the future. I have used other vendors for EDL extractions. The easiest way to determine if the extraction is decrypting or not is if the device remained in the off-state throughout the process. If the device never booted during the extraction process and the device was encrypted, the user data in the extraction will be encrypted. Some vendors inform you when the extraction is completed that the user data will be encrypted.

As of 12-10-18, there is no way to decrypt data after extraction if that data was encrypted with most modern Android encryption protocols. One of the likely reasons for that is because the encryption key is not included with the data extracted via a low-level, off-state processes and is not supposed to be stored on the device's storage. That doesn't mean decrypting the user data after extraction will never happen but on modern Android devices and virtually all devices supported by Cellebrite's EDL exploit, decryption must take place on the running device before extraction.

2.2.1.3 Booting devices during Cellebrite's decrypting EDL extractions

If a device is encrypted, decryption of the data occurs after the device is booted or when the data is accessed on a running device after the application of some exploit. For EDL extractions, the UFED needs the device to boot at a particular time during their extraction process for this to work. That is the six step process followed by the "executing" phase we see in the UFED when using the decrypting method.

Users of the UFED's generic EDL options must select the decrypting method vs. the non-decrypting method for decryption to work. If you select the non-decrypting method in generic options in the UFED, the UFED will not apply the decrypting bootloader. You have instructed the UFED to just do a low-level, two-step process of dumping the eMMC or UFS storage without regard for whether or not it is encrypted. You will simply get all of the 1's and 0's. If the device is specifically supported under the device profile in the UFED, Cellebrite makes this decision for you on most devices. There are some devices specifically supported for EDL extraction under the device's profile in the UFED that Cellebrite requires the user to pick either decrypting or non-decrypting.

2.2.1.4 If my device is specifically supported under its device profile in the UFED, why does Cellebrite make me choose decrypting or non-decrypting?

The answer to this question is because some devices can either be encrypted by the user or not encrypted by default when sold. Regarding producing useable results (user data that can be viewed), the decrypting bootloader will work on encrypted and non-encrypted devices. Whereas using the non-decrypting method on an encrypted device will yield an encrypted user data extraction (user data that can't be viewed). If that is the case, why pick the non-decrypting method on any given device?

- The decrypting bootloader requires six (6) steps and requires the phone to boot.
- The non-decrypting method requires two (2) steps and does not require the phone to boot.

- The non-decrypting method is low-level, doesn't change the hash, and will also work on devices that can't boot due to damage.
- The decrypting method requires a phone to work well enough to boot, the hash will change, and a booted phone that was not shut down in Airplane Mode and booted in a non-faraday environment will be exposed when being forced to boot by the Cellebrite decrypting bootloader process.
- The decrypting method requires the phone to boot normally and some phones may boot to a charge-only mode and that can interfere with the loading of the decrypting bootloader. In that case if the phone is not encrypted, a low-level extraction not requiring the device to boot will work.

It is therefore up to the user to determine what method is the best course of action. This choice is not available on all devices supported under their device profile in the UFED. That is because some devices are known always to be encrypted or some devices just haven't been tested to determine if they are encrypted by default or not. As discussed earlier in this paper, with EDL extractions there is no harm in applying the decrypting bootloader to an unencrypted phone or extracting an encrypted phone with the non-decrypting bootloader. So it is safe to provide users with both options.

2.2.2 What if I extract an encrypted device using the non-decrypting method?

The device will extract just as easily as a device that is not encrypted, but the user data will be encrypted. To be clear, I can extract encrypted devices the same way I extract non-encrypted devices (no battery, etc.). Because you have told the UFED to use the non-decrypting method, it will perform a low-level extraction without regard for the condition of the user data. Extracting encrypted data will be just as easy as if it were not encrypted – the data is just useless because it is encrypted and the user data can't be decrypted after extraction.

Figure 2 - User Data Encrypted Notification

Using the non-encrypted method on an encrypted device will not harm the device. Just like pulling an encrypted device via ISP or Chip-Off, you will end up with encrypted user data. Physical Analyzer will tell you the device's user data is encrypted when you begin opening the extraction (*See Figure 2*). When that happens, simply deploy the decrypting EDL method and extract the device again, which will then yield a decrypted image.

2.2.3 What happens if I extract a device that is NOT encrypted with the decrypting bootloader?

The device will extract using the decrypting method but this will not harm the device or change the user data. You will have a good, useable extraction and there is no need to pull it with the non-decrypting EDL method after that. The phone will boot for this process to complete so it will change the hash value of the image. Not knowing whether the device is encrypted or not is a definite possibility today as not every device comes encrypted by default. Some devices that are not encrypted by default can easily be encrypted by the user. Some devices running up to Android 7.1.1 may not be encrypted by default, but most are. So, using the decrypting EDL method will work on both encrypted and non-encrypted devices. (See Section 3.2 for determining device encryption)

2.3 Secure Startup and EDL

Secure Startup is still an area of confusion misunderstanding in the forensic community. Cellebrite can bypass Secure Startup on specific devices through [Cellebrite Advanced Services (CAS)]. Whether or not a bypass of Secure Startup will be an exploit that can be passed down to users of the UFED…I hope that is possible in the future but there are good reasons why that may not occur.

Since Secure Startup means we will not be able to extract the device with any conventional method, this should be a short section of this paper. However, there is information to be gleaned from discussing Secure Startup and it helps explain how EDL extractions work in the UFED. I continually receive questions from examiners and investigators regarding Secure Startup and how it affects or does not affect seizure, storage, and extraction of devices.

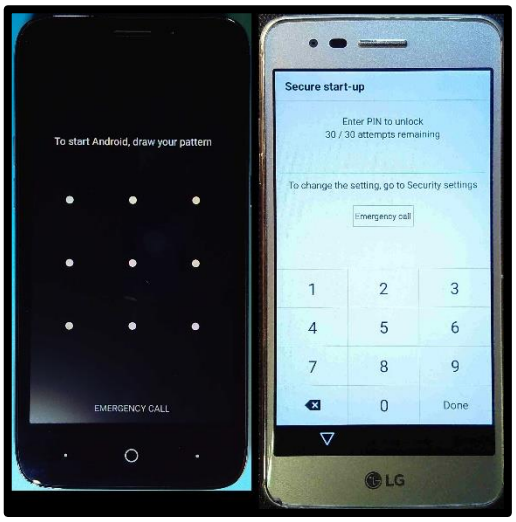


Figure 3 - Secure Startup Screen

2.3.1 How do you know if a device has Secure Startup?

The easiest way is to start or restart a device which will lead you to the Secure Startup screen as you can see from *Figure 3*. The screen may look different depending on the device OEM. Secure Startup is a feature of Android that allows the user to require a passcode or pattern to be entered before Android can be started. Thus, an examiner powering on a device will see this screen "To Start Android, draw your pattern" or enter your passcode – whichever method is picked by the user.

2.3.2 Does Secure Startup mean the device is encrypted?

Secure Startup answers the question that is a frequent topic of discussion and debate: "Is the device encrypted?" If the device has Secure Startup, the device is encrypted as this feature is only available with device encryption. The absence of Secure Startup does not mean the device is not encrypted. Secure Startup is an option given to a user when first setting up their device, or it may be added at any time by the user. Even though encryption may be enabled by default, Secure Startup is still a user option. (*See Figure 4*)

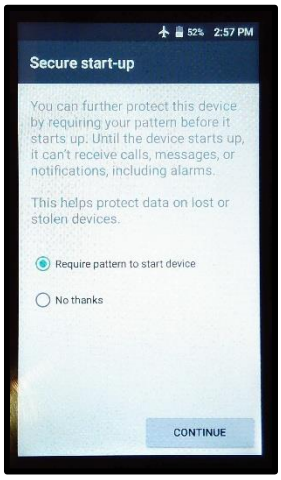


Figure 4 - Secure Startup choice

2.3.3 How do you know if Secure Startup is enabled on a running device?

Secure Startup only requires a passcode when the device is first powered on or restarted. This is separate from the user lock screen which requires a passcode or pattern when the screen is turned off or times-out. For devices that are first powered on or restarted, we will see the Secure Startup screen if it has been enabled. If the device is running and has a locked screen, there may be no way to know if Secure Startup is enabled on the device with just a manual inspection.

2.3.4 What happens if I try to extract a device with Secure Startup using the EDL method?

The answer is different depending on whether or not you use the non-decrypting method or the decrypting EDL extraction in the UFED. You might ask why an examiner would even attempt such an extraction. Consider a typical scenario in which the examiner may not know the device has Secure Startup enabled. If the device was seized in the off-state or if it was seized in the on-state and shut down, the person who seized the device may have only seen the user lock screen before shutdown. So there is no way to know if Secure Startup is enabled until the device starts.

If I receive a device I believe may or may not be encrypted, I may attempt a non-decrypting EDL extraction first, for reasons I cover in this paper. That will be an easy pull for a device that will remain in the off-state for my extraction and will not change the hash value. If the device is encrypted, I will find out when I attempt to open the image in Physical Analyzer. If it is, I simply attack the device with the decrypting EDL method to get my decrypted image. As discussed in this paper, there is no harm in using either method on devices whether encrypted or not.

A device with Secure Startup pulls exactly like a device without Secure Startup when using the non-decrypting EDL method in the UFED. The UFED will extract the device and when you open the image in Physical Analyzer you will be informed that the "User Data is Encrypted…". You will not be told that the device has Secure Startup nor will you see that on the device screen during the extraction when using a non-decrypting EDL extraction in the UFED. This is because the non-decrypting extraction does not require the device to boot – at all. This scenario also provides an excellent demonstration of how low-level the non-decrypting EDL extraction is and why it works on damaged and non-functioning devices so well. It will extract a physical image from a device running Secure Startup, which by definition and purpose prevents Android from starting. This is why non-decrypting EDL is virtually identical to ISP with regard to the end product.

In our scenario of the device seized without knowing whether Secure Startup is enabled, having extracted encrypted data with our first pull, we will now deploy the decrypting EDL method in the UFED expecting to get a decrypted extraction. It is only during that decrypting EDL extraction process that we will be informed the device has Secure Startup. I tested this exact scenario using a ZTE N9136 Prestige 2 running an MSM8909 loaded with Android 6.0.1 and encrypted by default. I enabled Secure Startup on the device and attempted decrypting and non-decrypting EDL extractions with the UFED.

2.3.5 Should a device be left running when seized to avoid Secure Startup?

I will answer that question relating only to EDL extractions. Secure Startup means the device is encrypted and so even though the non-decrypting method doesn't require Android to start, the extraction will yield encrypted user data and will, for practical purposes, be useless. That means the decrypting EDL method will have to be deployed. Even if the device is seized in the on-state (running) a decrypting EDL extraction using any technique to achieve EDL Mode will require the device to boot from EDL Mode. That means we will run right into the Secure Startup screen when the UFED orders the device to boot during the decrypting EDL extraction. A bootable device is necessary to apply Cellebrite's decrypting bootloader. All of that means keeping a running device alive in hopes of avoiding the Secure Startup screen really doesn't matter if we are considering only EDL extractions. Other extractions or exploits that do not require booting to apply or a shutdown may be different. Thus, keeping the device alive may be prudent with other scenarios.

2.3.6 If the Secure Startup passcode or pattern is known, must the Secure Startup pattern or passcode be removed for a successful decrypting EDL extraction in the UFED? (8-30-18)

This might be the only time interaction with a device screen will be necessary for EDL extractions – save ADB setup. We know that EDL extractions are not affected by screen lock. As long as the device fully boots, it doesn't matter that it boots to the user locked screen as long as it boots into Android.

2.3.6.1 Secure Startup with known passcode – UFED 4PC

Regarding Secure Startup with a known pattern or passcode, I tested this scenario using the ZTE N9136 again. I placed the device in FTM Mode (discussed later) which allowed the UFED to apply the decrypting bootloader for an extraction. The UFED caused the device to boot from EDL Mode and the device behaved as normal and responded to the UFED command to boot. The device reached the Secure Startup screen requesting the pattern shortly after the UFED extraction reached Step 6. I drew the known Secure Startup pattern immediately which allowed the device to continue and boot. This caused a temporary hiccup in which I heard the UFED repeatedly handshaking but the extraction was a success with this device. Although I didn't have to remove Secure Startup for the N9136, it may be necessary to remove Secure Startup completely with other devices if you have the known pattern or code.

2.3.6.2 Secure Startup with known passcode/pattern – UFED Touch2 (ZTE Z971) 8-30-18

I used the UFED Touch2 to extract a ZTE Z971 running Android 7.1.1 with an MSM8917 processor. This device is encrypted by default. I put a pattern lock on the device and set it up to require a pattern to start the device, thus enabling Secure Startup. I shut down the device and placed the device in DFU Mode. With the battery connected, I held volume down and volume up while connecting to the UFED Touch2. In step 6 of the extraction process, the phone is commanded to boot by the UFED Touch2, but with Secure Startup it stops and prompts the user for the pattern. I entered the known pattern for this device, it continued with the boot process and the extraction was successful. Note the device still boots to the user lock but that does not affect the extraction.

3 Device intake and testing for EDL Mode

Deciding which EDL extraction to use (decrypting or non-decrypting method) will depend on several factors that can include: the state and condition of the device, whether it is known if the device is encrypted or not, and to some extent the experience and training of the examiner. Understanding the statistics on the prevalence of default Android encryption, knowledge of device processors, and relying on experience from other examiners with particular devices are all in play with regard to how to deploy EDL or any other exploit.

3.1 Identify the device

3.1.1 UFED Direct Support

Check the device model for specific support for EDL extraction under the device's profile in the UFED (e.g. ZTE Z835). If the device is directly supported under its profile in the UFED, you can generally follow the instructions in the menus. If the device shows the decrypting bootloader option, it is likely supported for EDL. Navigate through the menus under that profile and look for specific instructions for entering EDL Mode for that device.

Figure 5 - ZTE Z835 UFED Support

3.1.2 Devices not specifically identified as supported in the UFED

If the device is not supported for EDL under the device profile in the UFED, it doesn't mean the UFED can't extract the device. Determine what processor it is running and check if that processor is one widely supported by Cellebrite for EDL extractions.

3.1.2.1 Check the Device Processor (7-17-18)

"**Widely Supported**" ***means that Cellebrite can use the EDL exploit to extract most devices running one of these processors***. There are exceptions, meaning there are a few devices running one of these widely supported processors that may fail. "**Limited Support**" ***means that Cellebrite can extract selected devices running these processors***. There are also one-offs – devices running processors not listed in either category which may extract using the UFED's EDL exploit. Cellebrite will list known devices that fall on the list of limited support. There will be devices running limited supported processors and widely supported processors, not specifically listed by Cellebrite, which will extract via the EDL exploit in the UFED, using the generic menu selections. Using the UFED to try generic EDL extractions on devices not listed as supported via EDL will not harm the device. However, careless techniques used to create EDL Mode can harm devices, so always use caution, research, and (whenever possible) test before attacking evidence.

3.1.3 Which Qualcomm processors can the UFED exploit via EDL Mode? (7-17-18)

Processors that can be exploited by the UFED via EDL Mode	
Widely Supported	**Limited Support**
MSM8909	MSM8996
MSM8916	MSM8917
MSM8936	MSM8937
MSM8939	MSM8940
MSM8952	MSM8953

3.1.3.1 Check for device encryption

If the device was released running Android 6.X.X or higher, the device could be encrypted by default but there are many exceptions. Someone on the **Mobile Device Forensic and Analysis Forum** will likely know or have a good idea about whether a device is encrypted or not. One of the benefits of EDL extractions is that trying different methods is generally harmless. So failing to extract, or extracting encrypted data, will only cost time. I see a significant number of new devices running Android 6, 7, and 8 that are not encrypted by default.

3.2 Identifying devices that are likely encrypted or not encrypted

I would like to start out by reminding anyone reading this document that I am not a developer or a coder and my knowledge of how Android encryption works is extremely basic at best. So my interest in encryption is pragmatic. What devices are encrypted? What are my options if a device that is likely encrypted comes into my lab? How do I make this determination? These questions account for a good percentage of the questions on listservs today and these conversations take place between very experienced examiners and beginners. From a forensic examiner's point of view, decisions regarding encryption are important and making the wrong decision can be fatal in some circumstances – meaning the evidence can be lost forever.

Let's first define what is meant when I or anyone else says the device is encrypted or the device is not encrypted. How are we making this statement about a phone we don't have in our hand? The answer is when examiners refer to devices that are encrypted or not, they are referring to the device model (not the specific piece of evidence) and by "**is/is not encrypted**" they mean "**out of the box**" (by default) when the user purchases the device, without regard for whether an individual user has encrypted their own device. Most contemporary Android devices can be encrypted by the user today, even if they are not sold encrypted. So the calculation that a particular device is not encrypted is typically based on examiner experience with that same model and the assumption that most users don't go out of their way to encrypt a device that is not encrypted when purchased.

There is a reason Google requires Android devices to be encrypted by default "out of the box". Because they know most of us won't do it if given the choice. It is for that reason examiners can say with some degree of confidence that a particular model is not encrypted when we find out that model is sold unencrypted "out of the box". We can say with even greater certainty that a device sold encrypted "out of the box" will always be encrypted when it comes into the lab. It is because converting such a device to something that is not encrypted is very difficult and not possible for most users. Google removed the ability to unencrypt devices that come encrypted by default, "out of the box".

The specifications of devices (hardware and software) can provide clues as to whether the device we are examining is encrypted. There are patterns with everything if we look. Regarding whether or not modern Android devices will be encrypted when sold is based on the specifications of that device. When purchasing a new phone running Android, there will be no mention of encryption on the box. Some packaging might mention the size of the storage or ram and the version of Android on the device. That information is helpful in providing clues about encryption but not conclusive.

Devices received in the lab that are locked means we cannot determine whether or not they are encrypted without having at least some limited access to the data. Are there patterns, rules, and clues to help us determine whether Android devices may be encrypted? There are. These specifications, characteristics, and other patterns are discussed in the next few sections.

3.2.1 UFED's EDL exploit works on encrypted devices

This is just another reminder that EDL is one of several exploits in the UFED that will produce decrypted extractions from encrypted devices. Getting a decrypted extraction using the EDL exploit is not done by accident. Cellebrite has developed a method for getting a decrypting extraction using the EDL exploit. Regarding the decision to select the decrypting or non-decrypting extraction; for devices that are supported for extraction via the EDL method, we can always extract the device with the non-decrypting method first. If we open the extraction in Physical Analyzer and find out the user data is encrypted, we can re-extract the device with the EDL decrypting bootloader. We have this option because neither method is harmful to the device whether encrypted or not (See section 2.2.2 this document).

3.2.2 Not all EDL extractions decrypt data

"Not all EDL extractions decrypt data" seems like a statement of the obvious because Cellebrite makes a distinction between the two extractions. Users of the UFED have no doubt what extraction they are about to perform under the EDL generic options. The UFED gives you a choice because the methods used to perform these extractions are different (See section 2). Still sometimes users may be confused after the UFED performs a successful extraction only to find out that the user data is encrypted. As discussed earlier in this document, Cellebrite is not the only vendor that uses the EDL exploit to extract data. However, Cellebrite is the only vendor to my knowledge (12-10-18), that also includes the ability to decrypt the data via the EDL exploit.

3.2.3 Why does the UFED allow me to perform an extraction of an encrypted device with a non-decrypting method? (11-5-18)

This will occur when using generic options. The simple answer is that the UFED doesn't know the device is encrypted when the device is locked, in the off-state, or otherwise not functioning. After the data is extracted, Physical Analyzer will detect encryption when it attempts to parse the image and will display the message (User data partition is encrypted…). The exception to detecting encryption on locked devices is that the UFED does have the ability to detect encryption on locked devices that can be placed in FTM Mode (In UFED Version 7.10.1.1080). Some locked devices can be placed in FTM Mode and some of those devices can be extracted via EDL as the UFED will use FTM Mode to create EDL. The Android Debug Console can detect device encryption on devices connected in FTM (See Section 4.3.4 on Android Debug Console and FTM).

The ADB Console can also detect encryption on running, unlocked devices with USB Debugging enabled. If USB Debugging is enabled, the device can be running and locked and encryption can be detected. On Android devices that are unlocked, we can also manually check the settings and see if the device is encrypted. Putting the FTM exception aside, for devices that are locked, we really can't tell if the device is encrypted because we can't get to the data on the device. The Android Debug Console cannot detect the presence of encryption on a device connected in EDL Mode. Of course if the device has Secure Startup, we know the device is encrypted. Devices with Secure Startup are always encrypted. If we power on a device with Secure Startup, we will see that screen and know the device is encrypted (See Section on Secure Startup).

3.2.4 Other Decrypting and Non-Decrypting Extractions

Cellebrite allows users to select or choose decrypting or non-decrypting when it comes to EDL extractions under generic options. Other extraction methods in the UFED do not give the user that option. For example, Smart ADB is a decrypting physical extraction but there is no non-decrypting version of Smart ADB to select in the UFED.

Cellebrite's exploit of LG's Advanced Flash (LAF) is a non-decrypting physical extraction but there is not a decrypting version of LAF for the user to select. The UFED can use LAF to place some LG devices in EDL Mode (See Section 4.10). So EDL is really the only extraction method in the UFED that allows us to choose decrypting or not.

3.2.4.1 Low-level physical extractions

Not all physical extractions decrypt data but encryption does not prevent the extraction. For experienced users this seems like an obvious statement but at one time it was not obvious for me as we all had to learn from experience and training. It is for this reason that a significant number of conversations on listservs revolve around examiners trying to figure out why they can't see the user data after successful physical extractions from various extractions methods and using a variety of forensics software and hardware. Low-level extractions do not result in decrypted data. You may get all of the 1's and 0's from the device, but those 1's and 0's will be encrypted.

LG Advanced Flash (LAF) is a good example of a low-level extraction that does not decrypt data. Using the UFED's EDL non-decrypting method is considered a low-level extraction. The device will not boot during the extraction. You are literally getting all of the 1's and 0's off of the device as the data is stored in its encrypted state. So with encrypted Android devices, the user data is encrypted if the device is locked or in the off-state. Of course, this is an over-simplification of encryption but generally if you are extracting data from an encrypted device in the off-state, your extraction (the user data) will be encrypted when you open it. For modern versions of Android, there is no way to decrypt data after it is extracted from an encrypted device. I hope I have to modify that last sentence someday.

3.2.4.2 JTAG, ISP, and Chip-Off do not decrypt data

Part of the encryption discussion has to involve or consider that the device we are examining is not supported for an EDL extraction. The encryption discussion comes into play with JTAG, ISP, and Chip-Off because these methods do not decrypt and thus, for extracting user data, these methods are not useful with encrypted devices. Also, if we are learning about EDL, we should discuss other options if an EDL extraction is not available. If the device is locked, we may not be able to use a bootloader or Cellebrite's Smart ADB. If the device or processor is not supported for an EDL extraction and other conventional, non-invasive exploits are not available, then we may consider trying other methods like JTAG, ISP or Chip-Off. So if we don't know whether or not the device in our lab is encrypted – either by default or by the user – we must consider and use these methods with caution.

This is another good time to suggest JTAG, ISP, and/or Chip-Off training for examiners. I mention this more than once in this document. These skills are still very useful today for a variety of situations. The most obvious situation is that there are still phones being sold new today on which these methods will work. Then there are the many thousands of older phones in the world today with evidence of a crime on them. Some of those phones are sitting on the shelf in evidence rooms in police departments all over the world. Some of them have yet to be discovered during an investigation coming to a jurisdiction near you in the future. Just to mention as anecdote, as I am writing this paragraph I am extracting evidence from a phone in a homicide case that was brought to me in the off-state after sitting unused by a suspect in a drawer for several years. The device is one I have used ISP on many times in the past – the ZTE Quest N817. It is running an MSM8610, which is not supported for an EDL extraction in the UFED but ISP is not a problem. For a phone that was given to me in the off-state with a swollen battery, ISP is the best option. The device will not be encrypted as it was sold around 2015 with Android KitKat. I also still receive phones that require the JTAG exploit to extract the data.

This document is not designed to teach JTAG, ISP, or Chip-Off. There are many who are much more qualified than I to teach and train others with those techniques. I will only describe those techniques generally because they do relate to the discussion of encryption, EDL, and generally the pool of options available for devices that come into the lab. So I will leave the technical details of these procedures out of this document.

One of the most useful skills I have came from knowledge obtained by studying and training to use JTAG, ISP, and Chip-Off. Learning to test, solder, and repair devices came from training and many hours of practicing with these

techniques and all of that allows me to access evidence on damaged devices that might otherwise be inaccessible (See Section 5.5 regarding extracting data from damaged devices).

3.2.4.2.1 JTAG

JTAG (Joint Test Action Group) is a processor exploit and is considered a low-level technique. By processor exploit, I am speaking in terms of forensic examinations, as forensic examiners can use this exploit only if they can communicate with the processor on any given device. Like many forensic exploits including EDL, JTAG was not created for forensic examinations, thus the term exploit is used. With many forensic techniques, the door was put in place for some other purpose. Forensic examiners use that door to retrieve evidence.

JTAG was created for the purpose of testing and repairing devices by the manufacturers. Not all processors are supported for JTAG (for forensic examinations) and some processors are only supported by a particular flasher box. JTAG (for forensics) is mostly used on older phones but there are still feature phones being sold new that can potentially be exploited by JTAG. Some processors are created and are only in production during a specific moment in time. Those processors become obsolete and replaced by faster processors with newer technology. But some processors are timeless and are still used on newer phones because there is still a market for cheap, feature phones. This is not only true for feature phones. There is a huge market for cheap Smart Phones. That market means that many, more modern, processors get recycled as well. This will be covered in more detail under Section 3.2.8 – Device encryption based on processors.

A good example of a 10 year old processor being used on contemporary phones is the Samsung Convoy 4, SM-B960V which was sold as a Verizon feature phone in 2016 (Click on Figure 6 for high-res). This device was added to support by the UFED in 2016 for a logical extraction only. These types of phones are found in retailers like Walmart. With a QSC6185 processor and 512MB of flash memory, this phone is not a candidate for ISP and Chip-Off of this type of storage would mean that the file system may have to be reconstructed to parse the data. JTAG is the best option for this type of phone. If the phone is not listed as directly support by one of the flasher boxes, then you may have to identify the JTAG TAPs on the device and then find a suitable “resurrector” profile with the

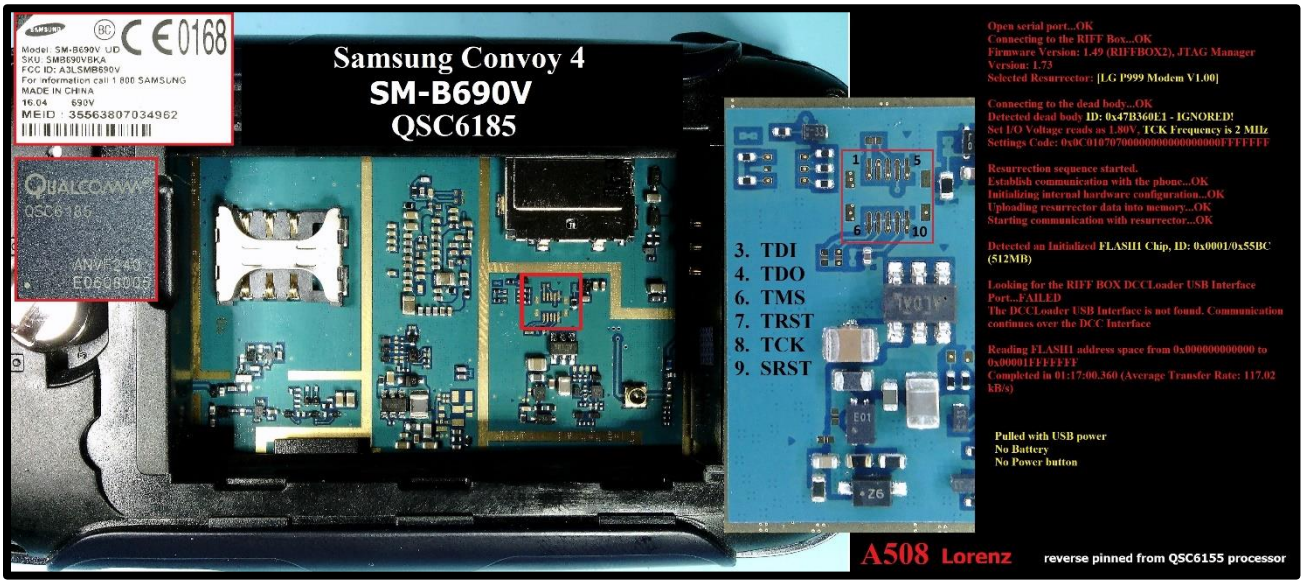


Figure 6 - Samsung Convoy 4 JTAG Pinout

proper settings using one of the flasher boxes used for JTAG.

Encryption is always a consideration for every decision but it is less of a specter on devices supported for JTAG. That is because devices running processors supported for JTAG are very unlikely to be encrypted. They just don’t

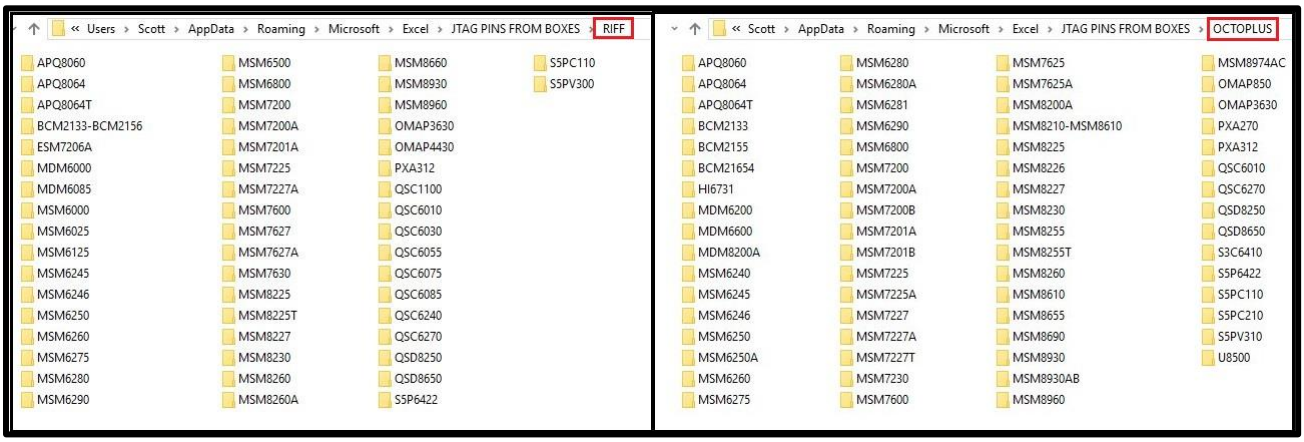


Figure 7 - JTAG Processors in RIFF and Octoplus

have the horsepower for it and the phones generally don't have the storage capacity or RAM to run newer versions of Android. To my knowledge, the processors listed as widely supported and listed as having limited support for EDL extraction by the UFED are not supported for JTAG. For the purpose of my database of phones, I have links to many JTAG pinouts for devices and I have organized them by processor. This allows me to select potential profiles in various flasher boxes when I am trying to JTAG a phone that is not listed for support but running a processor that is listed as supported for the JTAG exploit. Some of the processors supported by RIFF Box or Octoplus Box can support and run encryption, but not very well and not without a significant drop in performance. Phones running processors listed as supported for JTAG in the RIFF or Octoplus are very unlikely to be encrypted.

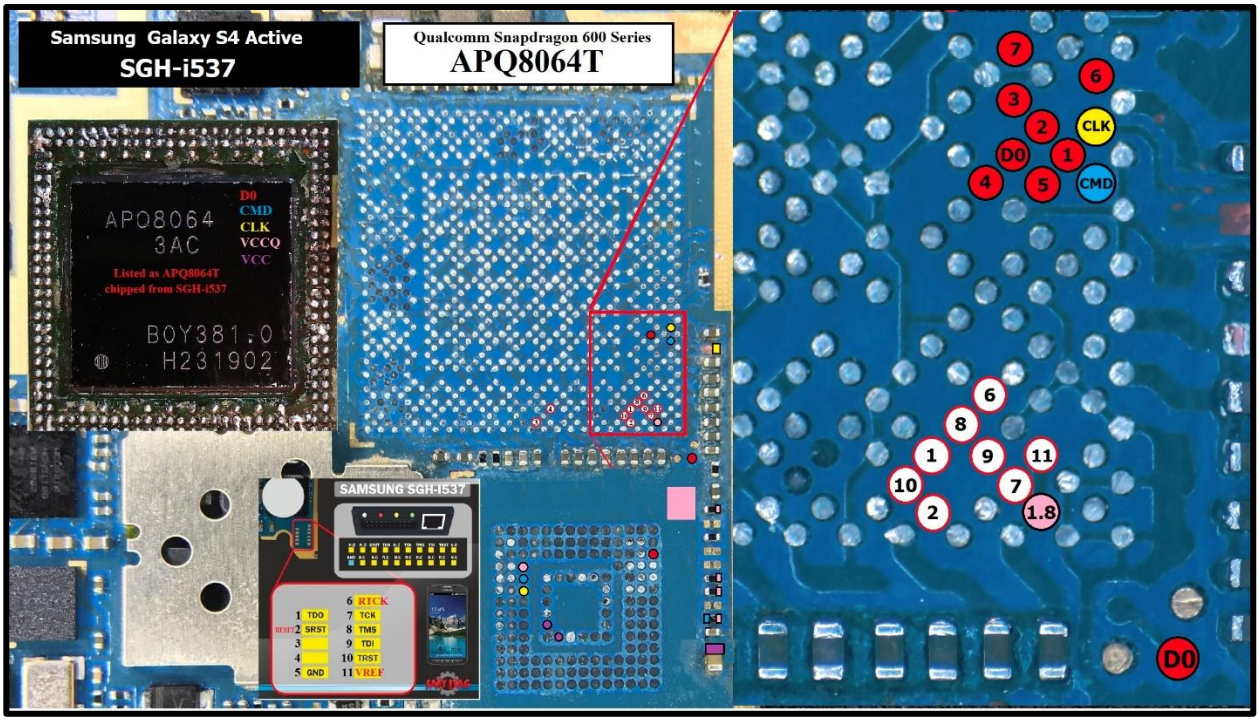


Figure 8 - APQ8064T JTAG & ISP Pinout

The JTAG exploit is accomplished by making contact with the processor via TAPs (Test Action Ports). These TAPs are connected to the processor on the device and commands can be given to the processor to dump the phone's

memory. Many newer devices have JTAG TAPs also but there is no JTAG support for many of those phones – in the forensic community.

While shorting ISP points creates EDL Mode, shorting JTAG TAPs does not create EDL Mode on devices I have tested. Some JTAG TAPs look like test points on other devices and in fact there are many pads all over logic boards that serve some other purpose other than for JTAG or ISP. The diagram of the SGH-i537 (Figure 8), shows JTAG and ISP points under the Qualcomm APQ8064T processor. The APQ8064T is not supported for EDL extraction in the UFED. But this phone can be exploited via JTAG, ISP, and Chip-Off. Knowing processor pinouts can be useful in locating ISP locations for ISP extractions and for creating EDL Mode via eMMC shorting.

3.2.4.2.2 ISP and EDL

In-System Programming (ISP) is not a processor exploit. The processor is not needed for this extraction and in many cases I have removed the processor in order to access ISP points that were not exposed on the logic board. The forensic technique of ISP extracts data directly from the eMMC or eMCP flash memory via some of the same flasher boxes used for JTAG. Since the processor is not needed for ISP, any device with eMMC or eMCP storage is a candidate for ISP. The exception is encryption. Encrypted devices can be acquired via ISP and Chip-Off but the user data will be encrypted.

ISP is a low-level process and is performed while the device is in the off-state. So if the device is encrypted, the extracted data will be encrypted. For an ISP extraction at least four (4) points need to be connected to the flasher box – the Data0 (D0), the command (CMD), and the clock (CLK), and ground. Power needs to be provided to the device/storage to include VCC and VCCQ or power can be provided through USB connection for many extractions. There are 8 data connections on eMMC/eMCP storage (Data0 – Data7). Faster extractions via ISP can be achieved by connecting more than one data line to flasher boxes that support that.

The data lines (D0-D7) are all connected to the processor, as are the CMD and CLK. So regarding creating "eMMC faults" or "shorting" to create EDL Mode, there are potentially ten locations to be shorted. It is just a matter of finding those locations on the logic board (as seen with ISP pinouts). More information on creating eMMC faults is found in section 4.13 – Creating eMMC faults.

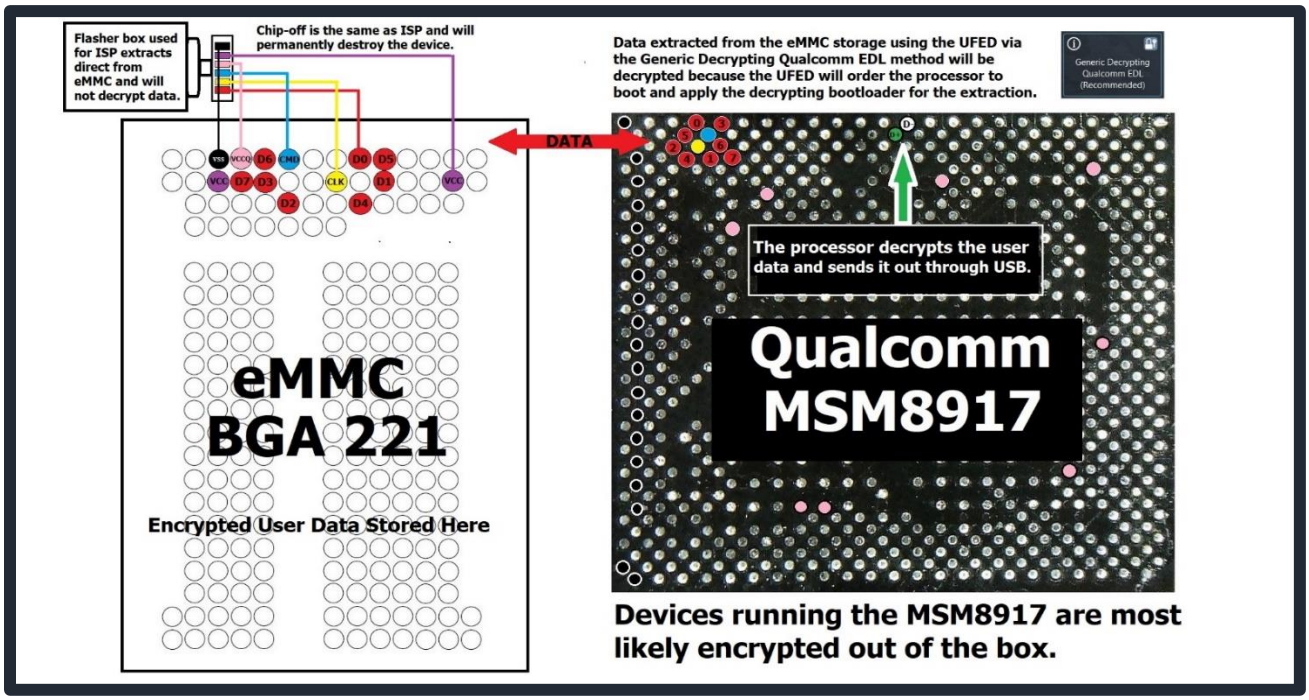


Figure 9 - BGA 221 with Qualcomm MSM8917 encryption

Note the eMMC storage in the diagram with the Qualcomm MSM8917 processor (Figure 9). The diagram demonstrates the connection between the processor and the storage. Data is transferred through USB via the D+ and D- lines marked on the diagram. So shorting one of the 10 ISP locations creates EDL Mode on the MSM8917 processor, thus placing the phone connected to USB in EDL Mode. The UFED uses EDL Mode to command the processor to dump the phone's memory. The UFED commands and data transfer takes place through the D+ and D- lines found inside the USB cable and ultimately connected to the processor as seen on the board pinout in Figure 7. If this is done via the non-decrypting EDL method in the UFED, the extraction will be low-level and yield the exact same result as extracting the data via ISP or Chip-Off. If this is done via the decrypting EDL method in the UFED, the UFED will order the phone to boot during the process so that a decrypting extraction can be performed. Phones running the MSM8917 are always encrypted so ISP or Chip-Off would not be an option for extracting the user data from this device.

3.2.4.2.3 Chip-Off

ISP is not fatal to the device but Chip-Off is. If there is not an available pinout for ISP, there is often a temptation to move to Chip-Off, because we think there is no other option. But if the device is encrypted, Chip-Off will yield nothing but encrypted user data and we have destroyed any chance of ever recovering the user data. The storage on encrypted devices are married to the hardware on the specific logic board of the phone on which they are mounted. The encryption key used to decrypt the user data is created via the user's passcode or pattern (or default passcode if the user does not lock the device) and other data created related to or stored in the hardware on that particular device. Therefore removing the storage via Chip-Off also means the encryption key can no longer be created and thus our encrypted extractions via Chip-Off or ISP will likely remain encrypted forever.

3.2.5 Device encryption based on Android version

Is it just me or are the rules of Android encryption confusing and inconsistent? Where are these "rules"? What happens if they are violated? Actually, the rules of Android encryption are quite consistent with everything else in the world of mobile devices. With regard to mobile devices and forensics I quickly learned that the words "never" and "always" should be avoided when answering questions about whether or not it is possible to access, recover, or extract data. The same rule is true with device encryption.

3.2.5.1 What is Android?

Android is an open source operating system for mobile devices developed by Google. The story of Android is much more complicated than that but this is just a brief overview. Google actually bought Android in 2005, and in 2008, the first device, the HTC Dream, was released running an Android operating system (Gragg, 2018).

3.2.5.2 Where are the rules about Android encryption found?

Even though Android is an open source operating system there are rules? These rules are found in the Android Compatibility Definition Document (CDD). In order for a device to run Android it must be compatible with the particular version of Android to be installed. These guidelines are available and on the Android Compatibility Definition Document webpage and the CDD for each version of Android from Android version 1.6 – 9 are available. Among many other requirements, the CDD provides the rules for Android encryption for that particular version of Android.

In January of 2015, Android 5.0 (Lollipop) CDD guidelines stated that, for devices supporting full-disk encryption (FDE), the FDE "**SHOULD** be enabled all the time after the user has completed the out-of-box experience" ((Android, 2015). That guideline was repeated in July of 2015, in Android 5.1 CDD guidelines. It wasn't until October of 2015, with Android 6.0 (Marshmallow) that the CDD guidelines changed from SHOULD to **MUST** with regard to devices being encrypted out of the box (Android, 2015). So Android 6.0 is when "out-of-box" encryption was mandated for Android devices (See Figure 10). However, there were and are exceptions to this rule.

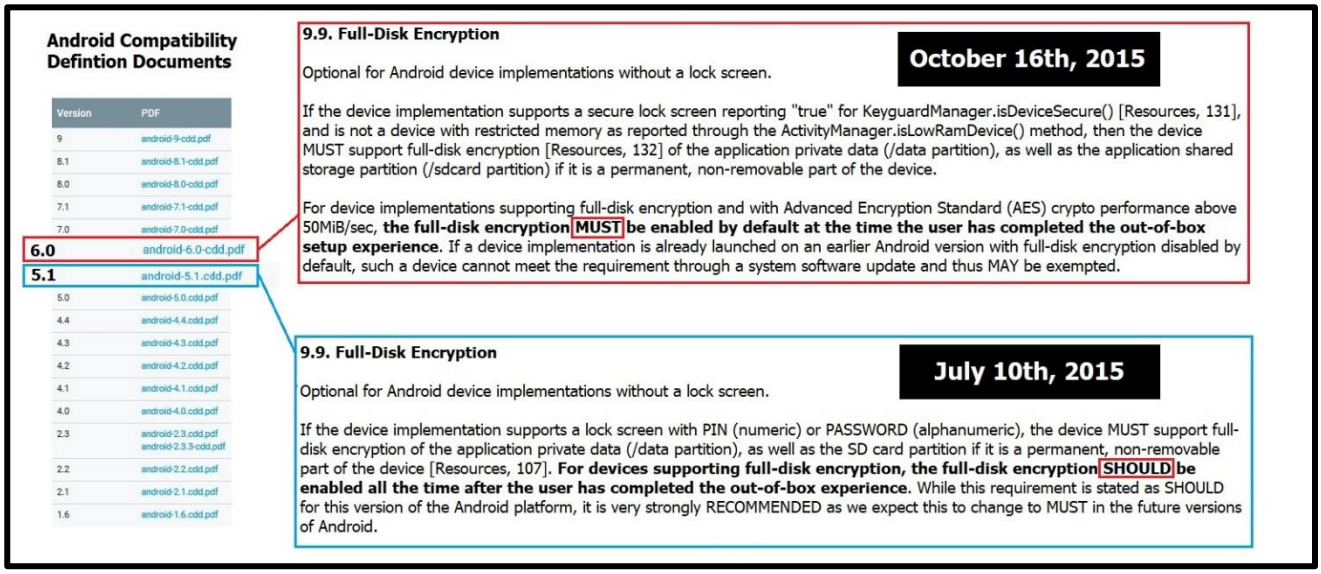


Figure 10 - Android Compatibility Definition Documents

3.2.5.3 Are Some OEMs violating the rules of Android?

If the Android CDD guidelines require all devices running Android 6 (Marshmallow) to be encrypted by default ("**MUST** be enabled"), were/are OEMs violating the CDD guidelines? After all, there are devices being released with Android 6, 7, & 8, that are not encrypted out of the box. The answer is that there are exceptions to the encryption requirement included in the CDD guidelines. One of the exceptions is that devices sold with older versions of Android that did not mandate default encryption, are exempted from mandatory encryption if they are updated to newer versions of Android. There are also exceptions to the "out-of-box" encryption mandate if the devices don't meet the hardware requirements needed to support encryption. The following excerpt is from the CDD for Android 7.0 – Nougat.

> *"For device implementations supporting data storage encryption and with Advanced Encryption Standard (AES) crypto performance above 50MiB/sec, the data storage encryption MUST be enabled by default at the time the user has completed the out-of-box setup experience. If a device implementation is already launched on an earlier Android version with encryption disabled by default, such a device cannot meet the requirement through a system software update and thus MAY be exempted."* (Android, 2017)

So just because a device is running Android 6, or higher doesn't mean it will be encrypted. If it was upgraded from an older version of Android before mandated encryption, or if the device just doesn't have the horsepower required to meet the minimum standards for "mandated" out-of-box encryption, it may not be encrypted. Keep in mind that devices falling into one or both of these categories can still be encrypted by the user if they chose to do so. It is easy and most of the time just requires a few minutes.

3.2.6 Device encryption based on manufacture date and UFED support

Going back in time is the best way to find devices that were not encrypted by default. Mandatory encryption was once only a rumor. Encryption was found on a few selected devices if the user chose to implement it, and then slowly more and more devices became encrypted by default. Cellebrite has always been quick to add support for

new devices hitting the market. That means their UFED Supported Phone List, put out in an Excel spreadsheet with each new UFED update, is a good place to get a quick look at device manufacture dates. The “Date Added” column in that Excel spreadsheet is a good way to filter by date to get a view of the past on a broad scale or to look at the date added to UFED support for a specific device. As an example, when you see a device that was added to the UFED for support in 2014, it is very unlikely the device will be encrypted. Using this technique is also useful to get a quick idea about whether a device can be exploited by JTAG or ISP. If I see a device added for support in the UFED in 2009, without knowing anything else about the device, I know it will likely be a device to consider for JTAG but not ISP. (See Figure 11, for an example of encryption related to device manufacture date)

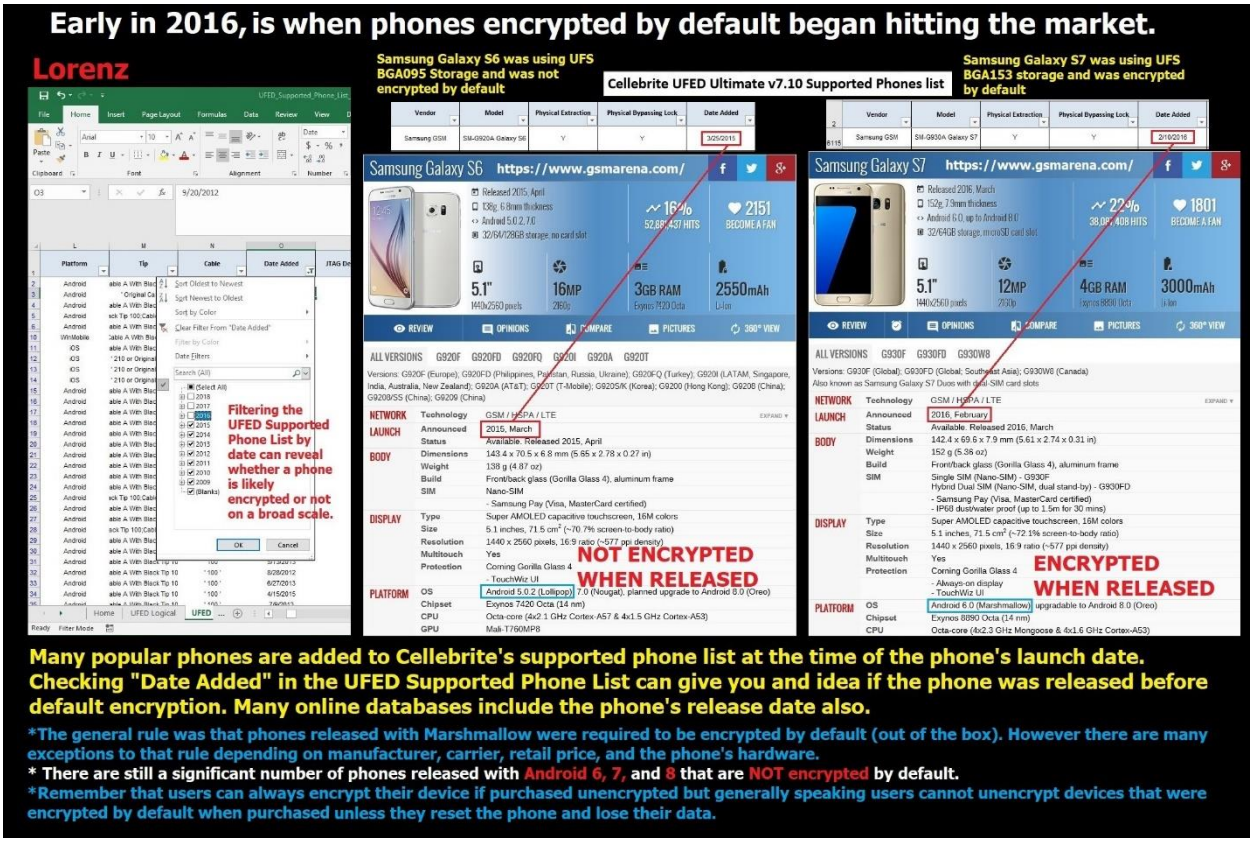


Figure 11 - Device Encryption by date

3.2.7 Device encryption based on processors

Regarding user experience, I will give you a general rule with processors that will almost always prove true with regard to encryption. I will use three very common processors with EDL support for this example: MSM8916, MSM8909, and MSM8917.

Generally speaking, of course…

- MSM8916 – not encrypted
- MSM8909 – may be encrypted or may not (refer to individual model)
- MSM8917 – is always encrypted

There can be exceptions regarding the MSM8916 and MSM8917, but statistically those assumptions above are true. There are many other processors that fall on the side of not encrypted with the MSM8916 or definitely encrypted with the MSM8917.

3.2.7.1 The MSM8909 is the center of the encryption universe

Some processors are timeless. By timeless I mean they are used on multiple generations of phones and can go forward and backward in time. When Cellebrite first announced that the MSM8909 was one of the processors that would be widely supported for extractions via the EDL method, I was very excited. The reason is because I saw the 8909 everywhere. The MSM8909 (Snapdragon 210) was launched in 2014, and is still very much in play today as it is still installed on new phones hitting the shelves. The MSM8909 fits somewhere in the middle between not encrypted and encrypted by default depending upon what phone it is mounted.

Since the MSM8909 was launched around September of 2014, it has been installed on phones running Android 5, 6, 7, and 8. It will likely be installed on many phones running Android 9 also. It has been a popular processor for phones that sell at the sub-$100 level and runs on phones as cheap as $20. For example, ZTE's Majesty Pro series of phones were sold for about $29 in stores like Walmart. The ZTE Z799VL and Z798BL were sold running Android 6.0.1 and were encrypted out of the box. The Z799VL and Z798BL began showing up in forensic labs everywhere and I had several on the shelf waiting for extraction when Cellebrite released the decrypting EDL ability in February of 2018. But before that, ZTE released the Z899VL, Majesty Pro Plus which was also running an MSM8909. The Z899VL was released with Android 7.1.1, however it was not encrypted out of the box.

The MSM8909 doesn't show any signs of going away any time soon as phones are still hitting shelves running this processor. The 8909 has proven to be inexpensive enough to mount on phones under $40, yet is respectable enough regarding performance to handle encryption on some devices. That is good news for forensic examiners.

3.2.8 Device encryption based on storage type

UFS (Universal Flash Storage) is more likely to be encrypted than eMMC (Embedded MultiMediaCard) storage. There are several reasons for this that are not directly related to the technical differences in these two types of storage. There were phones released with UFS storage that did not have out-of-box encryption. A couple of examples I list in this paper are the Samsung Galaxy Note 5 SM-N920xx and Galaxy S6 SM-G920xx. The Galaxy Note 5 and Galaxy S6 models were running Exynos 7420 processors. Both of these models were sold with UFS storage (FBGA095), usually 32GB. These devices were not encrypted when released but if they were passcode locked there was no immediate solution to bypass or extract data from some of the models to include the Verizon models – SM-N920V and SM-G920V. There was no ISP method that worked on UFS storage but eventually there was a Chip-Off solution with a particular UFS/eMMC programmer made by Dediprog.

The Chip-Off solution with Dediprog and the UFS BGA095 worked on the Note 5 and S6, which were not encrypted by default. When the Galaxy S7, was released in early 2016, it was running UFS storage (FBGA153). The S7's had out-of-box encryption. Some of the S7's were running an Exynos 8890 processor and some were running a Qualcomm MSM8996 processor. Dediprog had a Chip-Off solution with a UFS BGA153 adaptor but with default encryption for many phones running UFS storage, the adaptor is useless for many forensic examinations. See Section 4.17 for methods for shorting devices with UFS storage.

It is generally safe to assume phones running UFS BGA153 storage are likely encrypted by default. It has more to do with quality and cost of UFS storage than any technical requirement. UFS storage is faster than eMMC storage and it is generally sold in sizes of at least 32GB or greater. That means UFS storage will not be used on cheaper phones with less than 2GB of ram with slower processors. UFS will be installed on faster, high performing phones that will be encrypted by default.

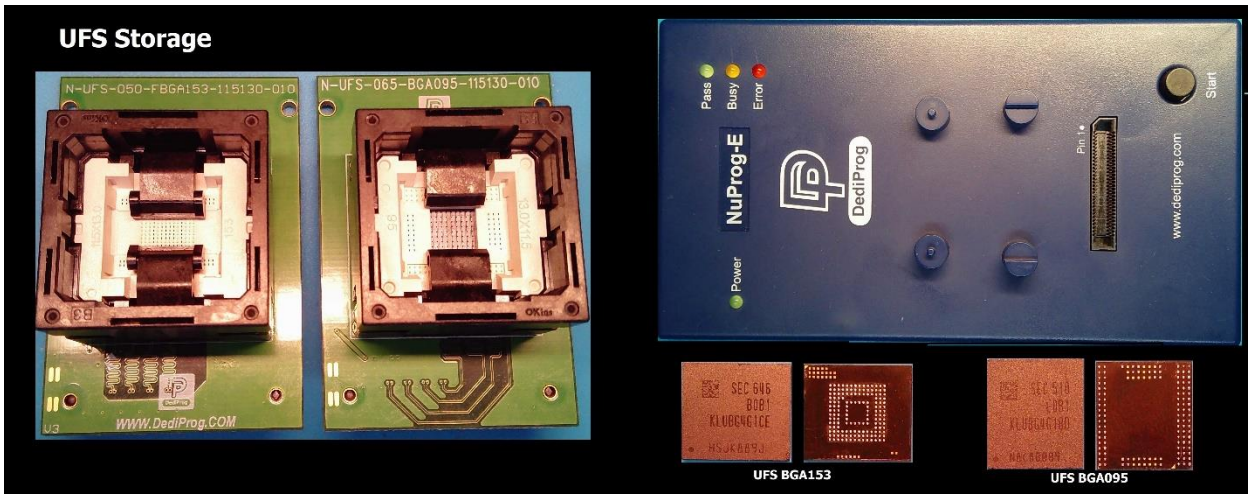


Figure 12 - Dediprog Chip-Off Solution for UFS Storage

3.2.9 Device encryption based on storage size

Storage size can be a determinant for whether devices will likely be encrypted or not. The full Android operating system takes up a lot of space. If a device has only 4GB or 8GB, of storage it is less likely to be encrypted. Smaller storage generally means older or cheaper phones. Older and cheaper mean less likely to be encrypted. But there are many newer phones running 4,8, or 16GB storage. Small storage is but one of the reasons for the development of Android (Go edition), discussed in Sec 3.2.11. Small storage is not necessarily the reason for the absence of encryption, but can be a sign that the device in your lab may not be encrypted.

3.2.10 Device encryption based on price

Are cheaper phones more likely to not be encrypted by default? Yes, but it is not absolute and not something on which you can always rely. Price is one indicator that a particular model of phone may be encrypted, but price alone is not a good indicator of whether a device is not encrypted by default. That is true with the Majesty Pro series that sold for $29 at Walmart but were encrypted out of the box – save the Majesty Pro Plus Z899VL which was not encrypted. Cheap does not mean the device will not be encrypted by default but there is a general rule you can follow for devices sold in 2016 or after. Today, not all cheap phones are unencrypted but all unencrypted phones are cheap.

3.2.11 Devices running Android (Go edition)

Android Go was mentioned by Google in May of 2017. Android Go is basically Google's recognition that there is huge market for cheap phones all over the world. Unless Google was willing to bend with Android specifications and requirements, Google could lose a large share of the smartphone OS market. Many of those inexpensive phones struggle to run newer versions of Android. Android Go was designed to be a scaled-down version to run on phones with 1GB of ram or less. It is optimized to take up less space for devices running small storage. Sagar Kamdar is the Director of Product Management, Android. The following is from The Keyword blog by Kamdar on December 5th, 2017.

> *"To make sure billions more people can get access to computing, it's important that entry-level devices are fully functioning smartphones that can browse the web and use apps. At Google I/O this year, we gave an*

early look at a project we called "Android Go" to make this possible. We're excited to announce that this software experience—Android Oreo (Go edition)—is ready, and launching as a part of the Android 8.1 release tomorrow." (Kamdar, 2017)

So with a phone running 8GB of storage, there is not enough room for the full version of Android which can take up to 11GB of space - with the full complement of pre-installed apps. With Android Go, an 8GB device can have 4GB of space left after install (Android Authority, 2018). It stands to reason that if a device is struggling to run the OS without much room for error, encryption would only slow the performance even more.

With all of the publicity and reviews of Android Go, it is actually difficult to find anyone talking about the absence of encryption or anything about encryption on devices running Android Go. That is not a surprise. Removing the requirement for or lack of support for encryption is not what Google wants to put on a brochure. Of course forensic examiners care about encryption but it is clear most Android users don't really give it much thought. I did find one article that listed NO "on-by-default" encryption as one of the bullet points in describing Android Go (Amadeo, 2018). That doesn't mean all devices running Android Go will always be unencrypted out of the box. Of course I always want to see for myself so I located and tested a few devices running Android Go. A couple sold with Android Oreo (Go Edition) were not encrypted out of the box.

3.2.12 Comparison of AT&T Prepaid Phones at Walmart

Figure 13 is a side by side comparison of AT&T phones at Walmart. Note the AT&T Axia has only 1GB of RAM compared to the Alcatel IdealXTRA with 2GB of RAM. Both devices have the same size storage (16GB) and are both priced at $49. The size of the RAM on the Axia is one factor which makes it a candidate for Android (Go edition) but another difference between the two devices are the processors. It is the ever popular MSM8909, that is found at the heart of many non-encrypted devices sold running Android 6 or higher.

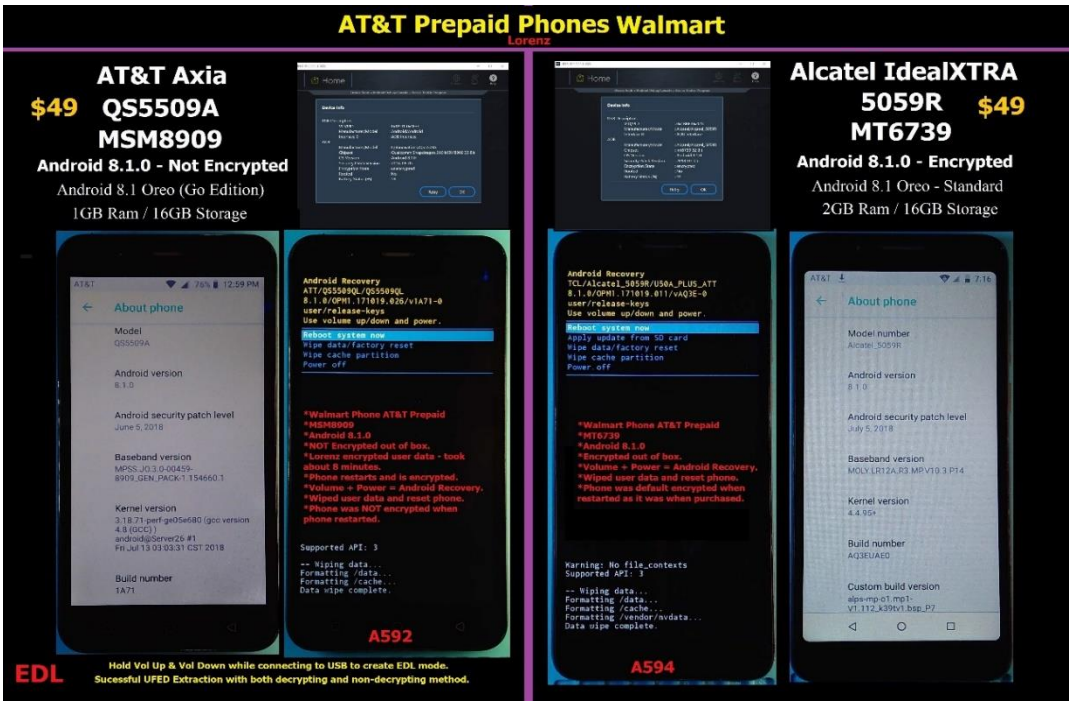


Figure 13 - AT&T Prepaid Phones Walmart

Android updates are distributed through the OEMs and the Axia will remain unencrypted despite updating to Android Pie, unless the user encrypts the device by choice, which can be done easily. As an experiment, I factory reset both of these devices and went through the setup process again. For the AT&T Axia, I first encrypted the device which only took a few minutes. This should be a reminder to examiners that just because a device is not encrypted by default, it can be encrypted by the user – even with low-end devices. After I encrypted the Axia, I wiped the data and factory reset it. The Axia was not encrypted after resetting and restoring the device, so it reverted to its original out-of-box condition – not encrypted. Of course, all of the user data is gone so this is not a forensic procedure to get around encryption. It was a demonstration that this device will likely always be unencrypted by default when purchased or when factory reset.

3.2.13 Examples of devices running Android 6 and above that are NOT encrypted

The following is a list of phones I have tested and all have not been encrypted. These are not the only ones and it is not an exhaustive list. There are many other phones, some from the same family as these phones, that are not listed here. By same "family" of phones, I mean running the same board but just renamed and repackaged for individual carriers. Notice that all of the phones have 16GB of memory or lower and most of them are running the MSM8909. All of the devices extract via EDL in the UFED using the non-decrypting generic method and some are supported under their device profile. Decrypting extractions will also work on these devices in case the device was encrypted by the user. I created this table to provide a view of the type of specifications that can indicate a lack of encryption on a particular device – **not** an absolute.

Devices Sold With Android 6 or Higher That Were Not Encrypted by Default						
Phone	**Processor**	**OS Version**	**RAM**	**Storage**	**Carrier**	**Specs**
Alcatel 5044R idealXCITE	MSM8909	7.0	2GB	16GB	ATT Prepaid	PS
Alcatel A577VL	MSM8909	7.1.1	2GB	Up to 16GB	Straight Talk	PS
ATT Axia QS5509A	MSM8909	8.1.0 (Go)	1GB	16GB	ATT Prepaid	Best Buy
Coolpad illumina 3310A	MSM8909	8.1.0 (Go)	1GB	8GB	Sprint	PS
Kyocera Cadence S2720PP	MSM8909	7.1.1	2GB	16GB	Verizon	PS
Motorola XT1609 Moto G4 Play	MSM8916	6.0.1	2GB	16GB	Verizon	PS
ZTE N9137 Tempo X	MSM8909	7.1.1	1GB	8GB	Virgin Mobile	PS
ZTE Z719DL Zmax One	MSM8909	7.1.1	2GB	16GB	Tracfone	Z719DL
ZTE Z835 Maven 3	MSM8909	7.1.1	1GB	8GB	ATT Prepaid	PS
ZTE Z839 Blade Vantage	MSM8909	7.1.1	2GB	16GB	Verizon	PS
ZTE Z852 Fanfare 3	MSM8909	7.1.1	1GB	8GB	Cricket	PS
ZTE Z899VL Majesty Pro Plus	MSM8909	7.1.1	2GB	16GB	Straight Talk	PS
ZTE Z963VL Max Duo	MSM8952	6.0	2GB	16GB	Tracfone	PS

3.2.14 Android 9 Pie (Go edition)

Android 9 Pie (Go Edition) is just coming out on newer phones. Whether or not they will be encrypted by default will likely depend on the same characteristics that have previously determined required out of box encryption. I verified the Blu Vivo Go V0390WW selling for about $79, running Android 9 Pie (Go Edition) is encrypted out of the box. It is running a MediaTek MT6739, has 1GB of Ram, and 16GB storage. Still there are some upcoming phones to be released with Android 9 (Go Edition) that will sell below $50. Some will run a Qualcomm and some will run MediaTek. Of course, devices that were not encrypted when sold will be exempted from default encryption when upgraded to Pie. The standard exemptions for performance and upgrading from a previous version of Android are still in place. The following is section 9.9. Data Storage Encryption, from the Android 9 CDD.

> *"If Advanced Encryption Standard (AES) crypto performance, measured with the most performant AES technology available on the device (e.g. the ARM Cryptography Extensions), is above 50 MiB/sec, device implementations:*

- *[C-1-1] MUST support data storage encryption of the application private data (/data partition), as well as the application shared storage partition (/sdcard partition) if it is a permanent, non-removable part of the device, except for device implementations that are typically shared (e.g. Television).*
- *[C-1-2] MUST enable the data storage encryption by default at the time the user has completed the out-of-box setup experience, except for device implementations that are typically shared (e.g. Television).*

  If device implementations are already launched on an earlier Android version and cannot meet the requirement through a system software update, they MAY be exempted from the above requirements." (Android, 2018)

According to Android, (Kamdar, Android 9 Pie (Go edition): New features and more options this fall, 2018) after announcing the first wave of (Go edition) phones in April of 2018, there are now more than 200 devices in 120 countries running Android Go. These Go edition devices can be with 4, 8, or 16GB of storage and available for as little as $30 USD. Android 9 Pie (Go edition) milks up to 500MB additional space from the flash memory compared to Android 8 (Go edition). That might not seem like a lot of space, but 500MB extra on a device running 4GB or 8GB of memory is significant. That means Android is attempting to create even more space on small storage with Android Pie (Go edition). Lower performance and less space means these devices are less likely to be encrypted than more expensive phones. It appears that Android (Go edition) is here to stay and the market for cheap devices may be expanding in the future. All of that means that forensic examiners can likely continue to see some devices sold running Android 9, not encrypted by default. To be continued…

3.2.15 Using the Android Debug Console in the UFED to determine if a device is encrypted

The Android Debug Console can be found under "Tools" in the UFED. Connecting an unlocked device with USB Debugging enabled will display several different characteristics about the phone to include whether or not it is encrypted. The ADB Console will also identify and verify that devices are in EDL Mode which is very beneficial for EDL extractions as you can actually use the console as part of the extraction process with an added benefit. Specific instructions on using the ADB Console can be found in Section 4.2.

3.3 Methods for placing devices in EDL Mode

In general, the following options are available possibilities for placing devices in EDL Mode. Fastboot and ADB require unlocked devices, unless the user's pre-seizure settings or other exploits have or can enable them.

1. Key combinations – e.g. holding Vol Up and Vol Down (locked and unlocked devices)
2. Cable 523 (Cellebrite) or homemade EDL cables (locked and unlocked devices)
3. 'adb reboot edl' (unlocked devices)
4. 'fastboot oem edl' (unlocked devices)
5. DFU and FTM mode (locked and unlocked devices)
6. Cellebrite's LG EDL method to create EDL Mode
7. Test points (on some devices) (locked and unlocked devices)
8. eMMC fault injection (shorting) (locked and unlocked devices)

These eight methods are generally listed from least to most regarding level of invasiveness to the physical condition of the device itself – meaning phone disassembly or soldering. With regard to disassembly, especially for shorting, this is an area that can be risky if not carefully done. I highly recommend training in ISP, JTAG, Chip-Off, or device repair if possible. If not, I am a firm believer in practice and repetition. I have over 600 phones in my inventory, most of which I have disassembled, repaired, chipped, used for JTAG and ISP, shorted and even cut into pieces.

I receive devices from multiple law enforcement agencies with varying degrees of training and differing operating procedures when it comes to seizing and storing digital evidence. I also receive a significant amount of damaged evidence or evidence that appears damaged but is in unknown working condition including items that have been

stored for long periods of time. If the device I receive is not encrypted and cable #523 doesn't work, I may opt for Test Points (on some devices) or eMMC shorting before powering on the device or trying anything else, especially if I don't know the condition of the device or how it was seized. I may also try FTM or DFU without a battery to ensure I don't boot the phone. If I can pull it in the off-state without booting, I will do that first – even if it means disassembly and shorting.

4 Testing, creating, and implementing EDL Mode

If the device is not directly supported in the UFED but is running a processor that is supported for EDL extractions, you will have to test what methods will likely place the device in EDL Mode. For a list of button combinations and other methods to create EDL, open the UFED, select "Smart Phones/PDAs > Qualcomm > Physical Extraction" and navigate to the "waiting for device" screen. This will give you various instructions and methods of creating EDL Mode. I recommend highlighting and copying the instructions in that window and pasting them into a text or Word document. If you are using the UFED 4PC and are going to test for EDL using Device Manager, you will need to close UFED 4PC to see EDL Mode in Device Manager.

Figure 14 - Waiting for Device Screen

4.1 How do I check to see if the device is in EDL Mode?

With most devices in EDL Mode, you will see a black screen and it may appear as if the device is off. The best way to determine if the device is in EDL Mode is by connecting the device to a PC via USB but with the UFED 4PC closed. If the device is in EDL Mode or if you place it in EDL Mode while connected to your PC, you should be able to see EDL Mode indicated in Device Manager e.g., "**Qualcomm HS-USB QDLoader 9008**…". Your PC may actually download a driver for EDL Mode if one is not available if you see, "**QHSUSB_BULK**". The UFED also provides a method to test for EDL using the Android Debug Console implemented with version 7.10.1.1080 (See Section 4.3).

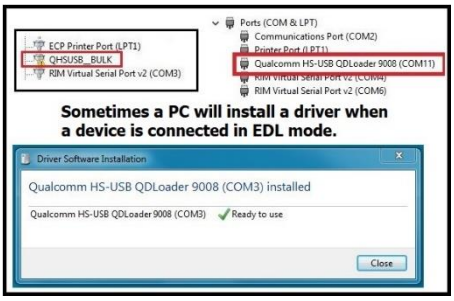


Figure 15 - EDL Mode in Device Manager

4.2 Why would I test EDL in Device Manager before extracting a device?

You don't have to test EDL on a device first. If you are sure the phone is supported, you can open the UFED, navigate to the "Waiting for Device" screen as seen in the Figure 15, place the device in EDL Mode, and then hit "Continue" when you hear the handshake and the button becomes active. If the device was in fact in EDL Mode, it will likely pull with no issues. Using UFED Android Debug Console for EDL extractions has other added benefits (Section 4.3).

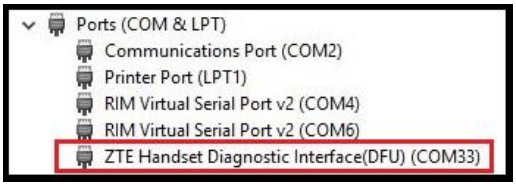


Figure 16 - DFU Mode in Device Manager

4.2.1 EDL Mode is not the only event or condition that will trigger a handshake

EDL Mode is NOT the only event that will trigger a handshake and activate the "Continue" button in UFED 4PC or create a handshake by connecting to a PC. Merely connecting a device with a battery and charging will most likely trigger this handshake, or the phone booting will trigger this handshake and activate the "Continue" button. Other modes available on phones like DFU and FTM Modes will trigger a handshake. The "Continue" button can become active for several different reasons. If the phone is not in EDL Mode or if you put the phone in FTM Mode but fail

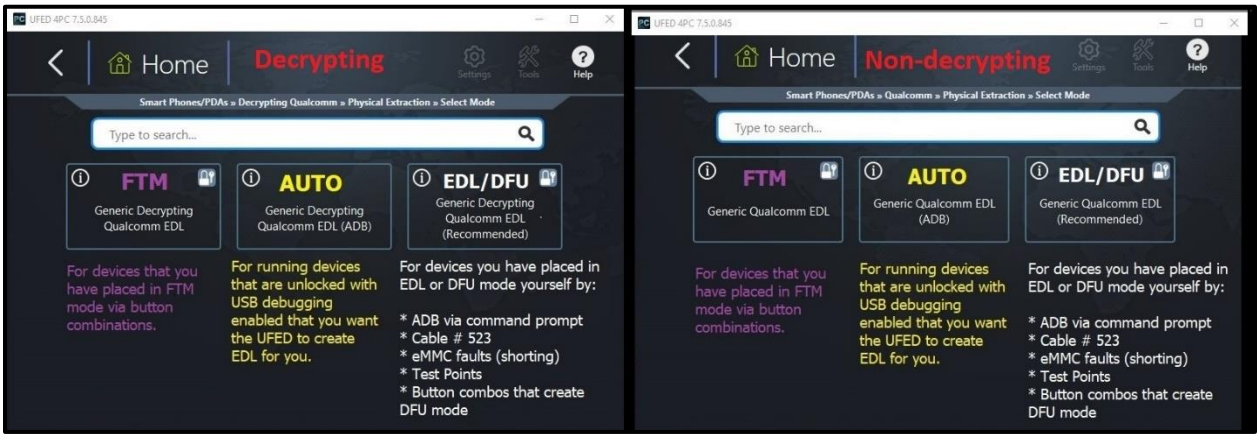


Figure 17 - Methods of creating EDL selection options

to select the correct menu option, the extraction will fail. The diagram in Figure 17 explains which menu option to choose, after you select the decrypting or non-decrypting method of extraction under the generic options.

So if the extraction fails, close the UFED 4PC, and try placing the device in EDL with Device Manager open. If your method places the device in FTM Mode, you will have to select a different menu option in Figure 17. By testing your method in Device Manager you can visually verify you are achieving EDL, DFU, or FTM Mode. You may just be connecting as an Android device. Once you verify this in Device Manager, disconnect the device, open the UFED and navigate back to the appropriate "Waiting for Device" screen. Now, perform the procedure you did during your test. If you have selected the appropriate menu options, the device should extract.

The diagram for how to extract a non-encrypted device will show you steps for testing ANY device for EDL. In other words, even if you are sure the device is encrypted but don't know how to place the device in EDL, use the steps in this diagram (**EDL Non-decrypting Pulling Steps**) to test your methods for creating EDL Mode. Once identified, employ the steps in this diagram (**EDL Decrypting Pulling Steps**) to perform the extraction on your encrypted device.

EDL Non-decrypting Pulling Steps	Step by step chart for non-decrypting EDL extraction – without ADB Console
EDL Decrypting Pulling Steps	Step by step chart for decrypting EDL extraction – without ADB Console

Typically, EDL Mode results in an almost immediate audible handshake upon connecting a device via USB. There is sometimes a delay of a few seconds dependent on the device and your PC. DFU Mode also creates an almost immediate handshake and behaves much like EDL Mode regarding the audible handshake and immediate recognition in Device Manager. You may have already placed devices in DFU Mode and pulled them via EDL in the UFED without knowing it. From the examiner's perspective, the UFED treats DFU Mode just like EDL Mode. The menu options are the same and thus a device in DFU is generally no different than a device in EDL Mode. FTM does require a different menu selection (discussed in Section 4.9.3).

4.3 UFED Android Debug Console - UFED version 7.10.1.1080

The Android Debug Console is a handy tool that can read information from a device connected to the UFED and display basic information about the device which the examiner can use to help determine the best course of action for extracting data. But the ADB Console does much more than display information. The ADB Console is a connection facilitator that alleviates some of the connection and timing issues associated with EDL extractions. The ADB Console can be used to detect EDL Mode and when detected, there is no need to create EDL again on the

device before performing an extraction. The examiner can navigate directly to the appropriate device profile or generic option to perform the extraction. Because the device is already connected and detected by the UFED, there is no requirement to create EDL Mode (including DFU or FTM) again to start the extraction.

4.3.1 Using UFED Android Debug Console to determine extraction options

Using the Android Debug Console does not relieve the examiner from decision making when it comes to extraction possibilities. Extractions other than those relying on the EDL exploit can be made after connecting through the Android Debug Console. This includes logical extractions under the device's supported profile, profiles of similar models, or other generic options.

For EDL extractions, there is still a need for the examiner to select the appropriate options after considering the condition of the phone, locked or unlocked, damaged, and whether or not the device is likely encrypted. EDL extractions beginning with the device in FTM or allowing the UFED to create EDL via ADB on unlocked devices, have separate menu options than do extractions beginning with EDL or DFU Mode. The ADB Console will help provide more information to the examiner and help fix some connection and timing issues with EDL extractions. Outside research will still be needed and fundamental knowledge about how extractions work is still necessary. New extraction diagrams are available for step by step decrypting and non-decrypting methods using the ADB Console.

ADB Console – Decrypting EDL	Step by step chart for decrypting EDL extractions using ADB Console
ADB Console – Non-decrypting EDL	Step by step chart for non-decrypting EDL extractions using ADB Console

4.3.2 Using Android Debug Console to display device information - UFED version 7.10.1.1080

Is the device encrypted? What processor is the device using? These are some of the things that help determine what extraction techniques or exploits will work or not work on a device. The ADB console will read and display this information as long as the device is unlocked with USB Debugging enabled. Some of the information detected can be found in the settings of the phone, but some things, like what processor the device is running, may not be in the settings. The ADB Console puts it all in one place on unlocked devices and devices in FTM.

So let's say I connect an unlocked device that is not listed as supported under a device profile in the UFED. This is actually a common occurrence. The device is likely supported for an extraction but there are so many variations of device models it is impossible to test and list them all. So we want to identify everything we can about the device to help determine our options. The first thing I am interested in on most extractions is what processor the device is running. If I connect an unknown device and the ADB Console tells me it is running an MSM8909, which is commonly encrypted, I know I have a good candidate for a decrypting EDL extraction in the generic options. The MSM8909 is a "widely supported" processor for EDL extractions. I then just need to find a way to get the device into EDL Mode and I can get a decrypted physical dump. Since I already have USB Debugging enabled on the device, I can select the Generic Decrypting Qualcomm EDL (ADB) option in the UFED menus and let the UFED do all of the work for me.

4.3.3 Using Android Debug Console to create and detect EDL Mode - UFED version 7.10.1.1080

Using the ADB Console as part of the extraction process means there is no need to worry about coordinating the creation of EDL with the "Waiting for device screen" in the UFED. The "Waiting for device screen" is exactly what it says. Before the ADB Console we could not create EDL Mode until arriving at that screen with the grayed out "Continue" button. Connecting and creating EDL Mode before arriving at that screen meant that the "Continue" button would be gray and we could not continue with the extraction unless we disconnected and created EDL again. But if we did that with the UFED open, there was no way to know if the method we used the first time, created EDL Mode a second time. We could assume we did when the "Continue" button activated but other conditions and circumstances can cause the "Continue" button to become active.

When using the ADB Console, we can create the EDL which will be detected and displayed in the UFED. Once we see the device is in EDL Mode, the examiner can navigate to the generic options, select decrypting or non-decrypting, and when arriving at the “Waiting for device screen” the “Continue” button is already active. Since we never disconnected the device, it is still in EDL Mode and can be extracted when we hit “Continue”. This solves a huge issue with examiners and failed extractions.

Figure 18 - UFED ADB Console - Device Unlocked & EDL Mode

4.3.4 Using Android Debug Console to create and detect FTM - UFED version 7.10.1.1080

Using ADB Console to read and display device data normally means the device has to be unlocked. The exception to that is when the device is in FTM Mode (See Section 4.9 for FTM). With a device in FTM Mode, the UFED can detect the same information about a device that was detected on an unlocked device with USB Debugging enabled, except battery status information. Although the ADB Console (UFED version 7.10.1.1080) does not explicitly indicate the device is in FTM, there are a couple of indicators that make FTM obvious. If the device is locked with no access to Developer Options, FTM is the only way to see the device information displayed in the

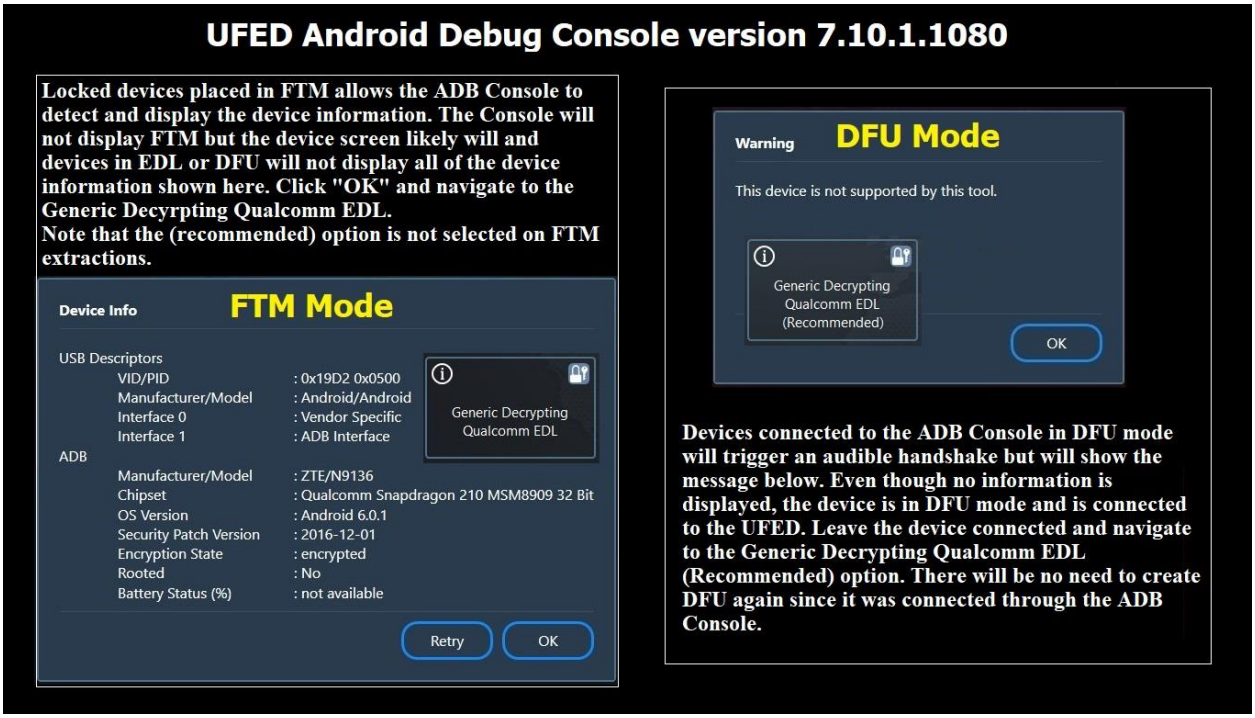


Figure 19 - ADB Console for FTM & DFU Mode

ADB Console. Another obvious indicator is that FTM is normally visible on the device screen with a window that shows FTM (Figure 24). Once the device is connected to the UFED and detected by the ADB Console, click “OK” and the Console will close. Keep the device connected and navigate through the UFED generic menus and select the appropriate options to extract the device. There is no need to create FTM again (support for FTM detection will likely be added in future versions of the UFED).

4.3.5 Using Android Debug Console to create and detect DFU Mode - UFED version 7.10.1.1080

DFU Mode is not supported for detection using the ADB Console in UFED version 7.10.1.1080. However, that doesn’t mean you can’t extract the device through the ADB Console using DFU Mode. Many examiners may not have ever realized they were extracting devices from DFU Mode even if they used generic options. This is because the UFED handles DFU the same as EDL from the examiners point of view. The menu options are the same. With DFU Mode, the benefit of going through the ADB Console is still in place even though the Console indicates “The device is not supported by this tool.” That just means the UFED won’t tell you the device in DFU Mode, but if the procedure was performed correctly and you heard a handshake, the device is likely in DFU Mode. Click “OK”, leave the device connected and in its current condition, and navigate to generic menus just as if you were going to extract a device in EDL Mode. The “Continue” button will already be active on the “Waiting for the device” screen

just as with FTM and EDL. The extraction will work and you are again relieved of having to create DFU again for an extraction. (Support for DFU detection will likely be added in future versions of the UFED)

4.4 EDL Cables

In absence of known support for your particular device, I would recommend trying an EDL cable first. Try Cellebrite's cable #523 or another EDL cable if you do not have it. The device should be powered off for this test. If possible, remove the battery and test the cable using only USB power. If that doesn't work, try with the battery in the device. Be aware that if the cable does not place the device in EDL Mode immediately, the device may fully boot once connected to USB power if you have the battery inserted. This can be an issue if you are worried about the device being exposed to the network (not placed in Airplane Mode when seized, etc). Most phones will not

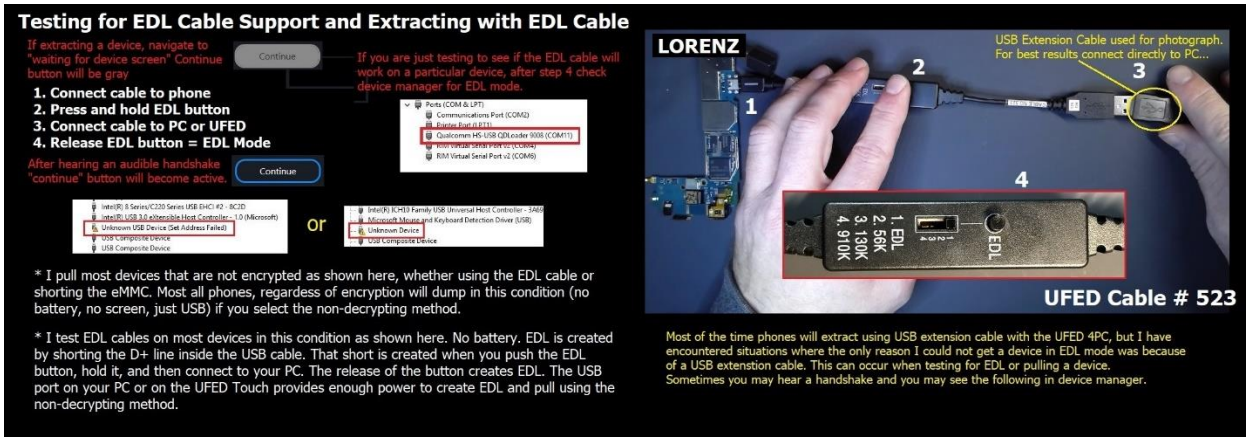


Figure 20 - Using and Test the EDL Cable # 523

fully boot without a battery but there are some that will – some models boot with just USB power or will boot-loop – start to boot, then fail repeatedly. Remember, you are doing coordinated steps. Just plugging in the EDL cable and pushing the button will not create EDL. Click on the diagram for the exact steps.

As described in Figure 21 with the ZTE Z812, not only is pulling a device via the non-decrypting method very easy and fast, it is also an effective way to test for EDL support for encrypted and non-encrypted devices with an EDL cable or eMMC shorting (covered later in Sec. 4.13). Plugging most phones into USB with just the logic board (no battery) will do nothing. There are some phones that will try to boot but most will do nothing. Short those same devices with an EDL cable and the device will immediately handshake and be in EDL Mode when you release the EDL button. If I have a device that is locked and cannot be placed into EDL Mode via button combinations, this is the method I would use to test EDL first. If EDL is not created immediately, I will try the same procedure with the battery in the phone. The procedure and order for pulling devices using the EDL cable is pretty straightforward. For a visual view of that procedure, I am including a link to a visual flow chart that covers **USING THE EDL CABLE or eMMC SHORTING** from start to finish. The process of creating eMMC shorts on a device is also covered later in this paper.

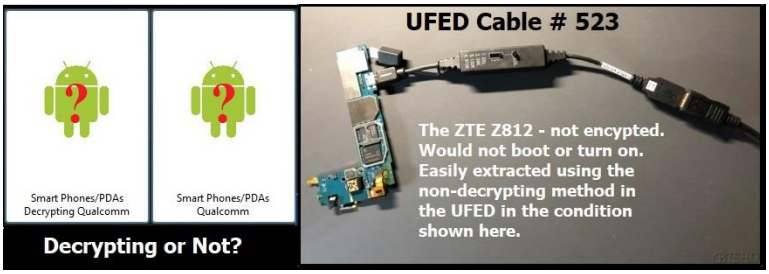


Figure 21 - Non-decrypting extraction with logic board only

Some phones will not be affected by the EDL cable at all. In those situations, pressing and releasing the EDL button may have no effect on the device. On some devices, the EDL cable may sometimes cause a device to not boot properly or may trigger some audible handshakes when pushed, but the cable will fail to create EDL. Device

Manager detects that a device is connected but pressing the EDL button with some devices caused the device to not be recognized. This means your coordination was off or the EDL cable will not work on that particular phone. The bottom line is that an EDL cable will just not work on some phones, just like button combos and ADB will not work on some phones.

4.4.1 Using the EDL Cable "shorts" the device but it is not the same as creating eMMC faults

Using an EDL cable is "shorting" the device. By pushing the EDL button, a ground is applied to the D+ line (green line inside a USB cable). With Cellebrite's cable #523, that button is pushed and held while connecting the EDL cable to USB. When the button is released, EDL Mode is detected by the PC or the UFED. Do not confuse this data line (D+) with the Data lines that connect the eMMC to the processor. For those familiar with ISP, these are Data lines 0-7 (See eMMC BGA221 & MSM8917). They are separate, different, and not directly connected with the USB data lines. I will cover that in more detail later when covering USB testing and bypass.

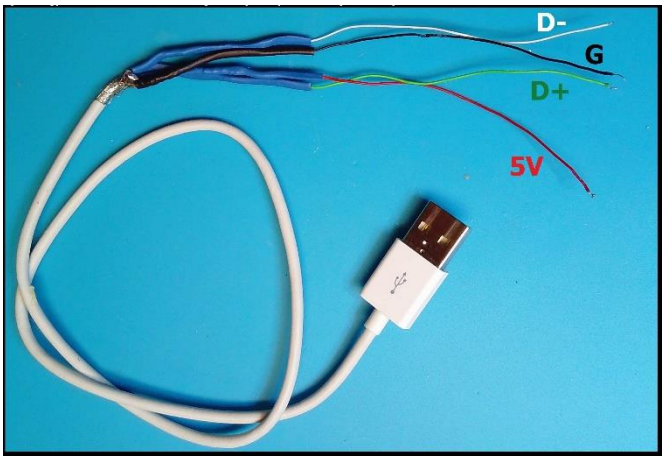


Figure 22 - Anatomy of a USB Cable

EDL cables create EDL Mode by connecting ground to (D+) during connection to a USB port. When the ground is removed, the PC or the UFED will detect a device in EDL Mode – if that method is supported by the phone manufacturer. Phone manufacturers can disable this protocol and protocols that allow EDL to be created via key combinations, Test Points, and ADB. Creating eMMC faults (shorting) cannot be disabled and will generally always work on Qualcomm processors.

The ZTE Z812 was a non-encrypted but badly damaged device that I used to demonstrate the use of Cellebrite's cable #523. That example also demonstrated the use of the non-decrypting method of EDL extraction in conjunction with an EDL cable. In that situation, no battery is needed and the phone is not required to boot during the process. It is easy and fast and will work on most devices even if they are damaged. Most devices that are not encrypted can be pulled in the same condition as seen with the photo of the Z899VL (Figure 23). This is because the non-decrypting method is low-level and does not require the device to boot.

Figure 23 - Non-decrypting Extractions without a battery

An encrypted device can also be pulled with this method. Of course, the user data is encrypted, but when I am testing for device support I will sometimes attempt to pull an encrypted device with this method just to make sure Cellebrite's EDL exploit will work on the device. I do this because the decrypting method of pulling phones can be challenging, even on supported devices. I want to make sure a failure to extract using the decrypting method is based on coordination and booting, and not because the device is just simply not supported. The non-encrypted pull is simple and easy, and if it doesn't work it usually means that processor and/or that device

is not supported. If the non-decrypting method works, then I know I have a supported deivce. I will have to repull it using the decrypting method, which will generally require a battery and some reassembly for the phone to boot. Phones that will not go into EDL Mode with the EDL cable can still be shorted into EDL via eMMC shorting or Test Points.

4.5 Button Combinations

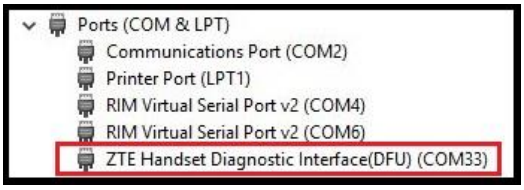
If you have a device in an unknown condition or are worried about the device not being in Airplane Mode, you should conduct this test in a Faraday environment as attempting button combinations may fully boot the device if you miss a step or are off on coordination.

I covered two basic extractions via EDL in the EDL webinar in February 2018. A PDF of that PowerPoint presentation is in the resource folder in the Mobile Device Forensics and Analysis forum. The "Resources Folder" contains many of the tools and guides needed for EDL extractions including write-ups on specific devices and ISP pinouts for many devices supported for EDL extraction. The Cellebrite EDL Webinar PowerPoint is also located there inside the EDL Extractions folder. In that PowerPoint, I cover basic extractions on the Alcatel 5044R using button combos to create EDL and the ZTE Z812 using the EDL cable #523 to create EDL.

Without specific knowledge and experience with a particular device, the easiest method to extract a device via EDL Mode is if it is directly supported under its device profile in the UFED. In the case of the Alcatel 5044R, EDL can be easily created via button combos and because that specific device is supported in the UFED, simply follow the instructions and it will walk you through the process. Even though that device was encrypted and pattern locked, the process is easy and automated with minimal user interaction.

4.5.1 What do button combinations really do?

Button combinations may not directly place the deivce in EDL Mode but instead place the device in DFU, FTM, or LAF Mode – then newest method Cellebrite uses to create EDL beginning in version 7.10.1.1080. Cellebrite then uses those modes to exploit the device and place the device in EDL Mode. Duplicating the button combo steps with the UFED closed will allow you to see the condition of the phone in Device Manager. This is important to understand because button combinations can be used generically to extract devices that are not directly supported in the UFED. Knowing what mode the button combinations create will be important so that you know which menu selection to make with generic Qualcomm methods. That will be discussed in detail in the section covering DFU and FTM Modes. Note that the UFED's LG EDL method using LAF to create EDL, is not available as a generic option. It will only be found on specifically supported devices under the device profile in the UFED.



4.6 Automated ADB via UFED's direct support or generic support with unlocked devices

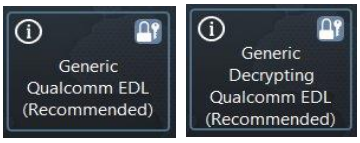
Devices that are unlocked or the passcode is known, means the examiner can enable USB Debugging, Stay Awake and Unknown Sources under Developer Options. With all of those settings enabled, Cellebrite can then use ADB to place the device in EDL Mode in order to take advantage of their EDL exploit on supported Qualcomm processors. Cellebrite can pull devices running processors supported for EDL under the generic Qualcomm method for both decrypting and non-decrypting methods. I recommend using these methods (decrypting or non-decrypting) if you have an unlocked device.

When using the automated ADB method, as long as you have enabled USB Debugging, Stay Awake, and Unknown Sources on the device, Cellebrite will initiate EDL for you. Cellebrite will also pull the device with the user lock in place. In other words, you can unlock the phone with a known passcode and enable USB Debugging and other options, but it was not necessary to disable or remove the user lock on devices I tested. By the time the UFED reboots the device to extract, the exploit is already loaded. You will see the device boot to the user lock screen, but no further action or unlocking is needed.

For the automated ADB method, just follow the instructions in the UFED prompts. You may be required to disconnect and reconnect the USB cable during this process for some devices but the UFED will handle the process of creating EDL via ADB. If you have a device seized in working order with access to Developer Options, and it is running a processor supported for EDL, I would recommend this method as a first try, especially for encrypted devices.

Using the automated function means that you don't have to worry about coordinating Command Prompt with the UFED 4PC, or moving your device from your PC to the UFED Touch after creating EDL via Command Prompt. For a more detailed explanation of those two methods, see "EDL Mode Via ADB with UFED 4PC Generic and Manual". That PDF is located in the resources folder on the Mobile Device Forensics and Analysis forum. I use the non-encrypted selection in that paper, but the steps are the same for the decrypting method - just a different menu selection.

4.7 Manual ADB or Fastboot used by the examiner to create EDL



For some users who just like to do things for yourself, you can use ADB via Command Prompt to create EDL Mode using "**adb reboot edl**" command. Users may want to verify a device is actually entering EDL Mode if an automated extraction fails or to verify this command will work on your PC before attempting this extraction with the UFED Touch.

You can also create EDL manually using ADB via Command Prompt and extract the device via the UFED. This is a different procedure than allowing the UFED to automatically create EDL via ADB. That means different menu selections. If you choose to create EDL yourself via Command Prompt or Fastboot, **DO NOT** select the UFED's ADB method of extraction. The **generic decrypting or non-decrypting method** of extraction should be selected for any method that involves you creating EDL outside of the UFED. That includes button combinations, shorting to create eMMC faults, Test Points, EDL cables, or EDL created by ADB or Fastboot using Command Prompt. When you create EDL yourself, don't tell the UFED to do it via ADB – it will fail. I have a specific EDL write-up for those control freaks who just want to control the world via Command Prompt (EDL Mode Via ADB with UFED 4PC Generic and Manual).

****Note** – Some devices will immediately trigger a handshake sound when rebooting to EDL after the "**adb reboot edl**" command and you will see the device in EDL Mode in Device Manager. If the device reboots to a black screen but does not create a handshake or appear in Device Manager, disconnect and reconnect the USB cable from the PC. The phone is likely in EDL; the PC just can't see it. This should trigger the handshake and refresh Device Manager. The phone remains in EDL Mode even after disconnected from USB because you presumably have a battery in it if using ADB*. Devices placed in EDL Mode will generally remain in EDL Mode until all power is removed (battery included) or they are forced to restart*.

4.8 Devices not supported for entering EDL via ADB or Fastboot commands

Some devices running processors supported for the EDL exploit, will not go into EDL Mode via an ADB command – either by your command or Cellebrite's automated process. On those devices, the **"adb reboot edl"** command will reboot the device, but instead of rebooting to a blank screen (EDL), the device will just fully boot as normal. You can test this by closing the UFED 4PC, connecting your unlocked, enabled, and booted device and then giving the command. If the device reboots to a black screen, it will likely be in EDL Mode. Check Device Manager. Failing to reboot to EDL does not mean the device cannot be extracted with the UFED. It just means you will have to find another method to create EDL.

4.9 FTM and DFU Mode to create EDL

Some devices can be placed in Field Test Mode (FTM) or Handset Diagnostic Interface (DFU) Mode with button combinations and the UFED uses those conditions to exploit them. This is still the EDL exploit at work, it is just a different path to achieve EDL. The ZTE Z835 Maven 3 is directly supported for the EDL exploit in the UFED. When navigating to this device profile, you will see instructions for button combinations to perform the extraction. However, there are several different paths that can be used to extract the Z835. The Z835 Maven 3, was sold as an AT&T prepaid phone and was running Android 7.1.1, but it was not encrypted by default. The UFED pulls the Z835 under its supported profile using the decrypting method. We know this because the extraction process used 6 steps and an execution phase. The device is also required to fully boot. Because we know the Z835 is not encrypted by default, we can also extract a fully unencrypted physical using several non-decrypting methods available in the UFED.

*Using the **decrypting method to pull phones that are not encrypted** will not harm the evidence and is forensically sound.*

4.9.1 ZTE Z835 using DFU Mode

ZTE's Handset Diagnostic Interface is normally achieved by pressing Volume Up and Volume Down while connecting the device to USB. This is the same instruction given under the Z835 supported device profile in the UFED. On devices I tested and including the Z835, DFU is created immediately with an audible handshake very much like eMMC shorting and Test Points. On devices I tested, devices can enter DFU without the battery connected and using the non-decrypting EDL method, the device can be extracted without the battery. The extraction is identical to shorting a device that is not encrypted and pulling it without the battery. So even though the ZTE Z835 is directly supported in the UFED under its device profile for a decrypting EDL extraction, my experience with the device tells me that it is not encrypted by default and can be pulled under generic options using the non-encrypted method. I use the same button combos described under the Z835 profile instructions – hold Volume Up and Volume Down while connecting to USB. DFU will result almost immediately and trigger an audible handshake. Because we are selecting a non-decrypting pull, the device will begin to dump after 2 steps and, most importantly, will not boot during the extraction.

*Pulling an **encrypted device with a non-decrypting extraction** will not harm the phone or the evidence, but the data extracted will be encrypted. If that happens, pull the device again using the decrypting method.*

4.9.2 How do I know if my device is in FTM or DFU Mode?

Checking to see if your device is in FTM, DFU, or EDL is all done the same way – by checking Device Manager. The device screen will most always be black for DFU and EDL Mode. FTM mode will generally have a screen that identifies it is in FTM. But each of these modes looks different in Device Manager. Most phones can be placed in DFU and EDL Mode with no battery in the device. FTM requires a battery for most devices. DFU and EDL Mode are similar in many ways, from the examiners perspective. DFU and EDL Mode will generally be created almost immediately when the condition designed to create EDL and DFU is applied and the device is connected to USB.

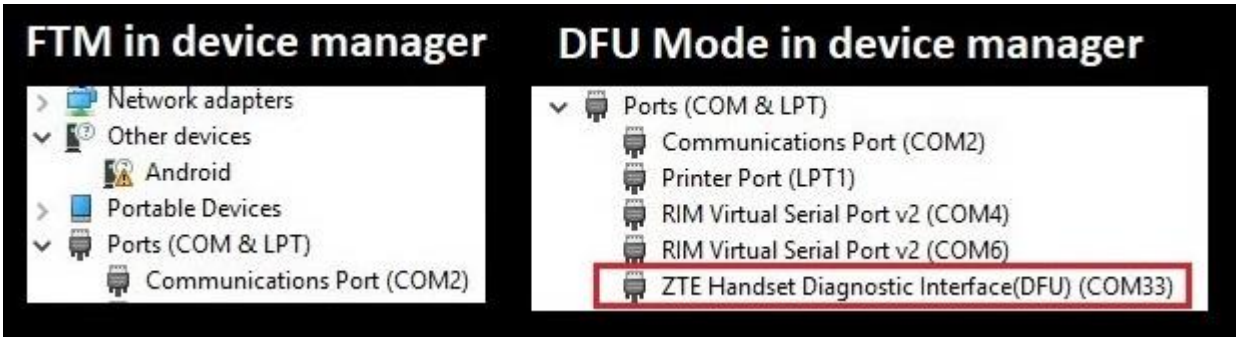


Figure 24 - FTM & DFU Mode in Device Manager

With EDL for example, applying an eMMC short will create an immediate handshake when connecting to a PC. The device will immediately appear in Device Manager as in EDL Mode. The same is true with DFU Mode on devices I have tested. With the device in the off-state (with and without a battery), holding down Volume Up and Volume Down together and connecting to USB at the same time will create DFU almost immediately. You will hear an audible handshake and Device Manager will show the device in DFU Mode. So if you weren't looking at Device Manager, the reaction of your PC (audible handshake) and the appearance of the device, are the same with EDL and DFU Mode.

Not only do EDL and DFU cause your PC and the device to behave the same way, the generic menu selections for devices in EDL Mode and DFU Mode are exactly the same in the UFED. In other words, your menu choices for selecting the Qualcomm (recommended) selection for either encrypted or non-encrypted devices will be the same. From the examiners point of view, it is really not even necessary to know if you have the phone in DFU or EDL Mode because the UFED will ultimately use both of those modes to achieve the same result. Examiners need to know this because those unfamiliar with DFU and how it works may not start an extraction thinking something is wrong. An examiner may think they have done something wrong or think the phone will not extract if they check Device Manager expecting to find EDL Mode and instead find DFU. For the decrypting and non-decrypting method of extraction in the UFED, they are both the same.

FTM can also be achieved by holding Vol-Down or Vol-Up while holding he power button with the battery inserted. This is before connecting to USB. The phone may start to boot once and restart once before success.

4.9.3 FTM behaves differently and requires separate menu selections than DFU and EDL

With FTM, a battery is needed, but exceptions exist. FTM is reached by holding either the Volume Up or Volume Down buttons while connecting the device to USB. The behavior of the device is different with FTM as opposed to DFU or EDL. With DFU and EDL, the examiner gets immediate feedback and a result. When attempting FTM, starting with the device in the off-state with a battery inserted, press and hold either the Volume Up or Volume Down button (not both) and connect the device to USB. The phone will appear to begin to boot as you will see the ZTE booting process begin. Continue to hold the volume button you chose. You may have to hold that button for up to 20 seconds on some devices I tested. If supported and you selected the correct volume button, the phone will display FTM as shown in the example of the ZTE Blade Spark created by holding the Volume Down button. Device Manager will also indicate the device is in FTM on devices I tested. I also checked Command Prompt and it showed a device with a question mark where you would normally see the device recognized by serial number.

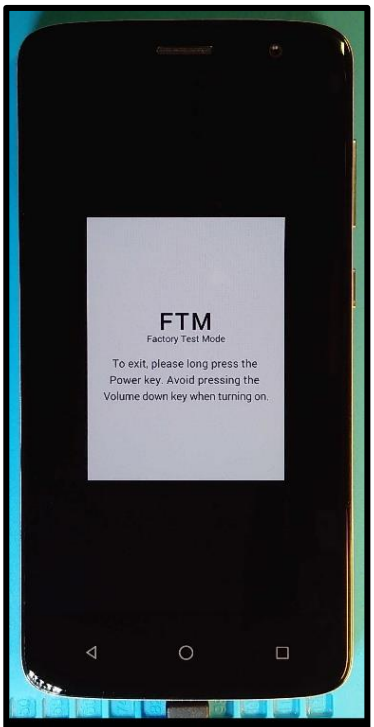


Figure 25 - Device in FTM

4.9.4 FTM method for generic EDL extractions

Unlike DFU and EDL, which are seamlessly interchangeable with menu selections in the UFED, FTM requires the use of a rarely used button in the UFED workflow of extraction steps. So when placing a device in FTM Mode and attempting an EDL extraction of a device under the generic Qualcomm options, the correct button must be selected (See the diagram on generic menus options in the UFED). Note that these options are the same and you must make the correct selection under decrypting or non-decrypting method. Selecting the Generic Qualcomm (recommended) method for a device that you have placed in FTM will fail or have issues. Instead of going to steps and 1 and 2 and then to a binary dump, you will receive a message to enable USB Debugging. That message is incorrect, given what you are trying to do and what just occurred. For FTM extractions, it means you selected the wrong menu option (See Section 5.4 Common Error Extraction Messages).

4.9.5 Why doesn't Cellebrite just tell me what button to push all the time?

Cellebrite is probably the most user friendly mobile forensics programs on the planet (my opinion). Most of the time the UFED does tell us what button to push. But there are some things that are out of Cellebrite's control and must be left to user discretion and selection. There are so many ways to create EDL Mode and many of those methods must be done outside of the UFED and involve significant, physical manipulation and disassembly of the device. If the choice is to only include directly supported devices with explicit instructions on each device that leaves no room for error or a more flexible tool to that allows experienced users to find their way – I choose the latter.

I created the Generic EDL options diagram to help encapsulate the menu selections based on the different methods of creating EDL, either by you or the UFED. Most of the EDL extractions I do are done under the Generic Qualcomm EDL menus, because I prefer to select my method of EDL and I am comfortable with all of the Generic Qualcomm solutions and options. There are so many devices running Qualcomm processors that it would be impossible to field test each one. Note that those menu choices are true for both all Generic Qualcomm EDL extractions to include the decrypting and non-decrypting methods. So after you select decrypting or non-decrypting, the menu choices are the same after that. The UFED can use its decrypting EDL tool with devices placed in EDL, DFU or FTM by all of the techniques in that diagram.

4.9.6 Shorting a device into DFU Mode

There is one more issue I will cover in this section, even though it can be covered under eMMC faults as well. I have discussed using eMMC faults to short devices into EDL Mode and various strategies for doing that. There are other things that can occur when shorting devices that are not EDL but may go unnoticed or may cause an examiner to believe they have done something wrong. Later in this paper, I go into which ISP points can be shorted to create eMMC faults. I have put out some diagrams in the past that indicate shorting all ISP points create EDL Mode. That included D0-D7, CMD and CLK. There are some exceptions I encountered a while back and explored more when testing devices for this paper. Shorting the CLK pin on some phones will cause the device to go into DFU Mode instead of EDL Mode. When that occurs it is true for both sides of the CLK pin. It only happens on some devices and I have not tested the CLK pin on enough devices to put a percentage on it but I have verified it repeatedly.

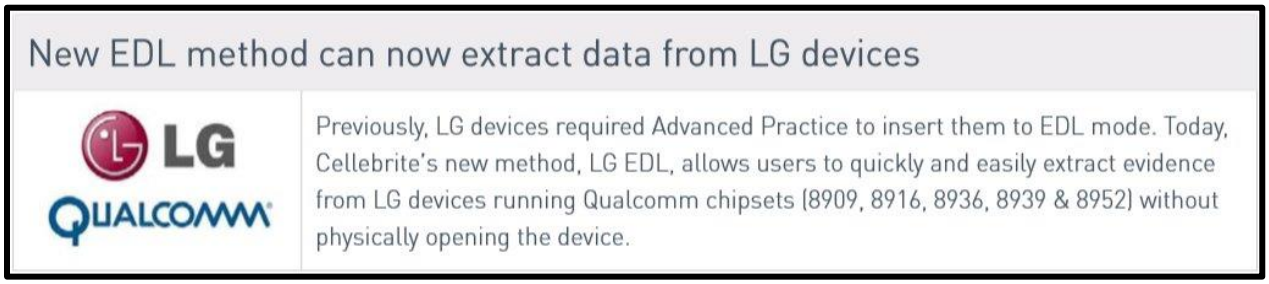
So the good news is that it will not affect an EDL extraction. As we know now, from the point of view of the examiner, the UFED will handle a device in DFU Mode the same as a device in EDL Mode. Of course the UFED is doing more behind the scenes, but an EDL extraction will occur with the exact same menu selections. It is worth mentioning because I have advocated that some examiners short the CLK pin on some devices simply because it was more accessible given the board layout and heat shields. A user who wants to test his or her shorting result on the CLK pin during an extraction, might be confused when they look in Device Manager and find that they have shorted the device into DFU Mode. No worries, it's normal and it will extract.

As an academic exercise, I tested CLK shorting on the following devices and you see the results below. Other ISP pins on these devices produced EDL. Only the CLK produced DFU. There is a link attached to each device with access to the diagram. It doesn't affect an EDL extraction so it is just FYI.

Phone	Processor	FTM	DFU	CLK short
ZTE Z971 Blade Spark	MSM8917	Vol Down - connect USB	Vol Up & Vol Down - connect USB	EDL
ZTE N9136 Prestige 2	MSM8909	Vol Up - connect USB	Vol Up & Vol Down - connect USB	DFU
ZTE N9560 Max XL	MSM8940	Vol Up - connect USB	Vol Up & Vol Down - connect USB	EDL
ZTE Z839 Blade Vantage	MSM8909	Vol Down - connect USB	Vol Up & Vol Down - connect USB	DFU
ZTE Z835 Maven 3	MSM8909	Vol Down - connect USB	Vol Up & Vol Down - connect USB	DFU
ZTE Z852 Fanfare 3	MSM8909	Fail	Fail	DFU

4.10 Cellebrite's LG EDL method to extract data from LG devices

Do you ever get the feeling Cellebrite doesn't want us to disassemble phones? Leave it to Cellebrite to chip away at one of the few simple pleasures I have in my life. There are several ways to place/force devices into EDL Mode. Some of those methods, like EDL cables, ADB commands, or button combinations, can create EDL without having to break into the device. When those methods fail, it may be necessary to disassemble the device to access test points or create eMMC or UFS faults (shorting). Of course that means cracking open the device, using tools or heat, and soldering, in some cases. Things can go wrong with disassembly of a device and with shorting. It also takes time to do that or to research how to disassemble a particular phone. Then, there is the issue of locating test points or searching for an ISP pinout. Of course not everyone is trained or equipped to perform such procedures or may not have permission to perform such an operation on a victim's phone or phone obtained via consent. So it is logical for Cellebrite to continually search for non-invasive methods of exploitation to avoid physically opening a device. Still I can't help but feel some of the magic is lost.



4.10.1 The UFED taking advantage of LG Advanced Flash (LAF) to create EDL Mode

With this method, Cellebrite is using LG Advanced Flash (LAF) as a means to create EDL Mode on LG devices. LAF is normally created by pressing and holding the Volume Up button while connecting to USB. The phone will display download mode and then "Firmware Update" on the device screen. Cellebrite also provides another method of getting to the Firmware Update screen. Using Cable #523 (Cellebrite's EDL cable) you can set the switch to position 4, connect the cable to the phone, and then connect the phone to USB. This method worked for me on several phones but if it doesn't work, try the traditional method also. Just remember to move the selector switch back to position 1 before continuing the extraction if you use the EDL cable. When the selector switch is on position 1 on

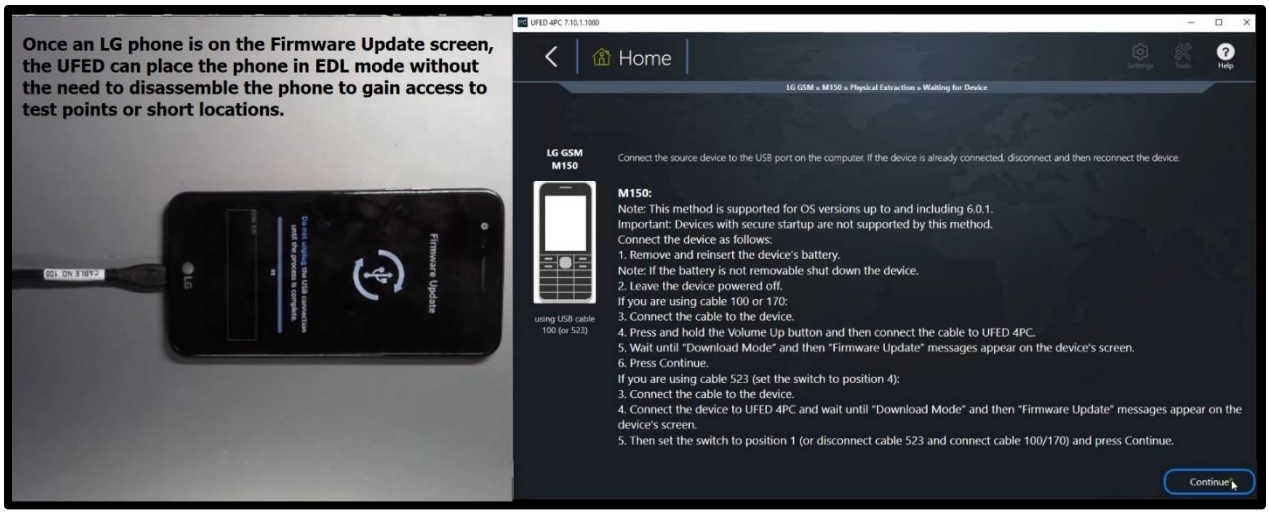


Figure 26 - UFED uses LG LAF to create EDL Mode

Cable #523, it performs just like a standard USB cable. When the phone is placed in Firmware Update at the "Waiting for device" screen, the continue button will be active. Press "Continue" and the UFED will take over from there. The UFED will then take advantage of the LG phone being in Firmware Update and cause the phone to go into EDL Mode. So some LG phones that I had to take apart to access the test points, can now be extracted with this safer, quicker but somehow less satisfying method.

4.10.2 Why is it necessary to move from LAF to EDL if LAF will dump the phone?

If I place the LG M150 in Firmware Update mode, why can't Cellebrite just use LAF to extract a physical image of the device as the UFED does with many LG phones? What is the point of moving from LAF to EDL Mode? Because using LAF to dump the physical storage is a low-level process. If the device is encrypted, the user data extracted via LAF will be encrypted also. That means it will be useless. With Android encryption there is no way to decrypt a physical image "after" it has been extracted from a device via low-level methods. So while LAF is excellent for bypassing user locks, if the phone is encrypted, our extracted user data will also be encrypted. Therefore Cellebrite uses LAF to shift to EDL Mode, and uses its decrypting bootloader to extract a physical image. The phone will boot as it normally does with an EDL decrypting extraction. So this method of getting devices into EDL Mode will be found under buttons labeled with "Decrypted Bootloader (Recommended)". LAF, in these extractions, is a means to and ends.

LG-M150 EDL Mode via LAF	UFED's new method of creating EDL Mode from LAF – Decrypting extraction
LG-LS676 EDL Mode via LAF	EDL Mode from LAF on device with non-functioning screen – Decrypting extraction

4.10.3 The new EDL method for LG phones is not found under generic options

This new method will not be found under generic options and is only used on devices running selected processors. As stated in the UFED release notes, *"Previously, LG devices required Advanced Practice to insert them to EDL Mode. Today, Cellebrite's new method, LG EDL, allows users to quickly and easily extract evidence from LG devices running Qualcomm chipsets (8909, 8916, 8936, 8939 & 8952) without physically opening the device."*

You will notice that the processors listed are all from the widely supported list. This new EDL method will only be found under specific LG device profiles and not under generic options because Cellebrite does not want users to attempt this method on devices running processors not listed. As I have mentioned several times in this paper, EDL Mode will not harm devices but methods used to create EDL Mode can harm devices. That warning does not just apply to physical methods of creating EDL Mode. Attempting to force devices into EDL Mode via software exploits can disable or brick devices. So it is recommended to only deploy this method only when it is presented as an option for the specific device supported under the specific device profile in the UFED.

4.11 LG EDL Recovery tool

Ending the last section by discussing things that can go wrong when EDL Mode is created via software exploits, provides a convenient segue to discuss the LG EDL Recovery tool. Sometimes LG devices can get stuck in EDL Mode even if you have made no attempt to disassemble or solder or otherwise crack open the device. I have done this before accidentally and, when preparing to write this section, I did it on purpose in order to create a demo video. In the past when I have done it accidentally (got devices stuck in EDL Mode), I managed to get devices working again only after a long time messing with them and applying power and holding buttons for long periods of time. When I did get them functioning again, I couldn't tell you how I did it except to say it involved cursing, pushing every button on the phone, and abusive use of power sources. I also didn't know that it was a relatively common occurrence until I saw Cellebrite added a tool to fix the issue in the UFED.

Getting devices stuck in EDL Mode would usually happen to me when I would spend hours testing devices trying to get decrypting bootloaders to work. When I would see the extraction failing I would just yank the USB cable out, kill the UFED, and impatiently start again. Occasionally, I would end up with a device that was in EDL Mode when it shouldn't be. Of course these were test devices and I would never be so careless with evidence.

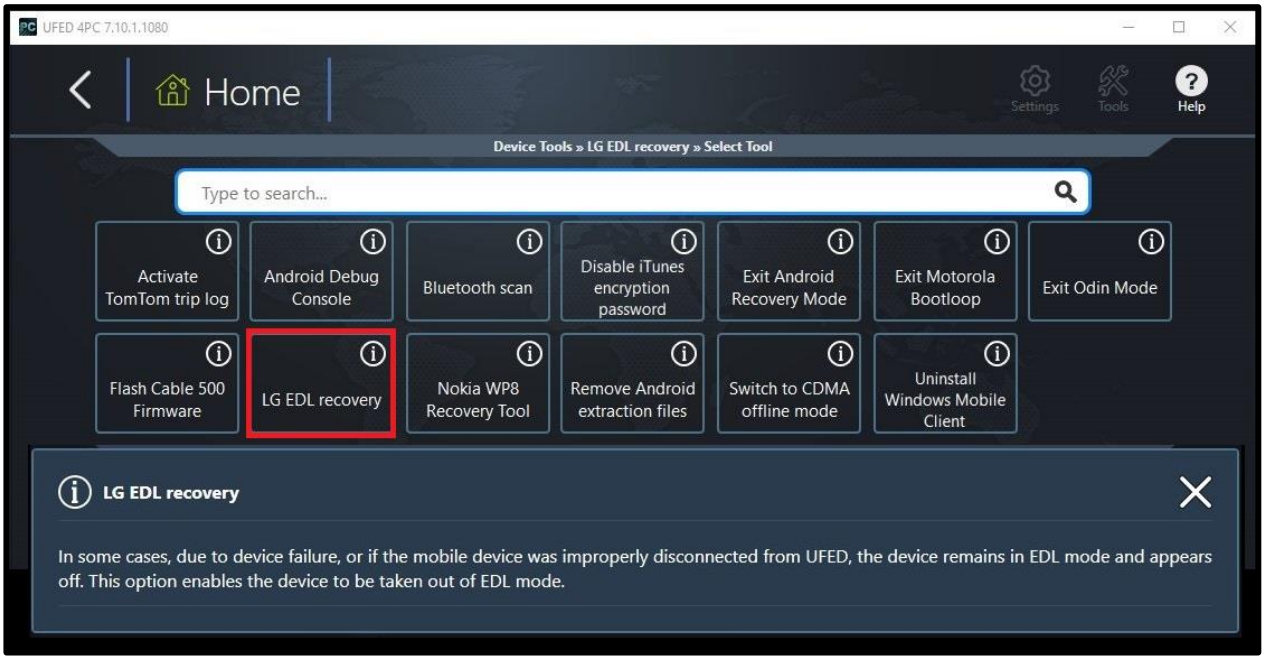


Figure 27 - UFED LG EDL recovery tool

I have discussed the issue of permanent EDL Mode with examiners many times in the past and present and have written about it in this paper in several locations. EDL Mode is what we are shooting for, but permanent EDL Mode is bad. One of the most common errors with EDL extractions failing when using eMMC faults, is when examiners fail to remove the short they created, before pressing "Continue". It is because the device needs to come out of EDL Mode for the extraction. Thinking back, I suspect getting devices stuck in EDL Mode via improper connections issues has happened to me on some ZTE devices but out of the hundreds of phones I have tested, I can't remember and I will have to see if I can create it again on devices other than LG. Best practices should be to avoid this situation entirely by allowing extraction processes to complete before disconnecting and when extractions fail, follow the instructions given in the UFED and wait until the UFED finishes before disconnecting. But it is good to have this tool if it does happen and the tool does work. See the video below.

LG EDL Recovery Tool	Demo of LG device stuck in EDL Mode and removal process with UFED

4.12 Test points to create EDL Mode

Note: In addition to the information found in this section, also see Mastering EDL Test Points. This new document covers all aspects of locating and explaining the orgins of EDL test points.

Test points are installed on some boards for the purpose of allowing access to EDL Mode. Although I have this method listed under shorting, actual Test Points are not shorting one pin to a ground on the phone as is the case with the EDL cable or eMMC shorting, which both create EDL by shorting data, CLK, CMD, or D+ to ground. On phones running Qualcomm processors I have researched and located Test Points, neither of the points are ever ground. Either one or both test for voltage (usually 1.8 volts).

As an example, I used the Xiaomi Mi 5X running the Qualcomm MSM8953. This device is supported for Smart ADB in the UFED if unlocked and that method resulted in a successful physical extraction. A quick search of the internet reveals that many Xiaomi devices have Test Points that allow them to be shorted into EDL Mode. Most of the time shorting Test Points can be done with a pair of tweezers. Using the tweezers to create a bridge between the two points and then connecting the device to USB while holding the tweezers on each point (See Figure 28) The device will almost immediately go into EDL Mode just as if you used the EDL cable, eMMC shorting, or DFU. My tests on the Xiaomi worked with and without the battery connected. Neither of the Test Points is a ground pin. One of the pins registers 1.8 volts with the battery connected and the other registers no voltage. *Note that Test Points create EDL Mode due to instructions provided in the bootloader code. Thus, they can be turned off or blocked (unlike eMMC faults) just as ADB and Fastboot commands to create EDL can be turned off.

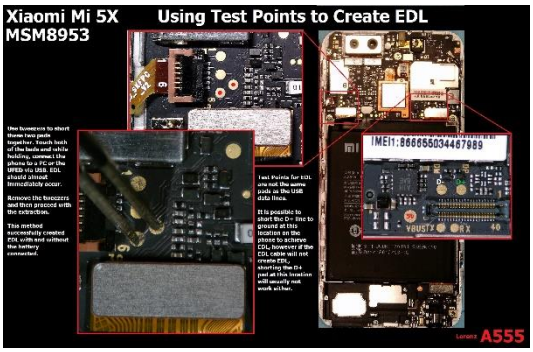


Figure 28 - Xiaomi Mi 5X EDL Test Points

4.12.1 Looking and probing for Test Points

On many brands, Test Points look like every other pad on the phone. Sometimes the presence of two pads near each other in a particular location can be a hint, but there are many round pads on all types of phones all over the board. An example of some typical Test Points can be found on the Motorola XT1922-7 moto g6 Play (MSM8920). You can click on the diagrams for a blown-up image. Note the Test Points I have marked (Figure 29) I picked to test because of their proximity to each other and location. I first tested them for voltage and ground. I look for pins that are not ground and may or may not have voltage. I test ground points with the phone in the off-state with no battery connected. I test for voltage on Test Points with the battery connected. Once I have my suspected pins, I disconnect the battery and USB. With no power to the phone, I place the tweezers on my suspected pins as shown in the Xiaomi diagram. With tweezers in place, I plug the phone into USB. If you have the correct points and your tweezers haven't slipped, EDL Mode will be almost instant. You will hear an audible handshake and Device Manager will register EDL.

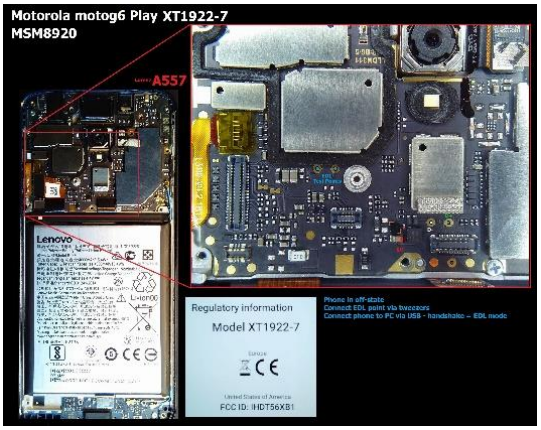


Figure 29 - Motorola motog6 Play XT1922-7 Test Points

So while proximity gave me a hint on the XT1922-7, another Motorola phone did not position the Test Points so obviously. The Motorola Moto E5 Play XT1921-5 (MSM8920) had me puzzled for a bit. The Test Points are positioned unusually far from each other (Figure 30). My reason for trying those two pins was that neither was ground and one was voltage. Click on the diagram for a blown-up view.

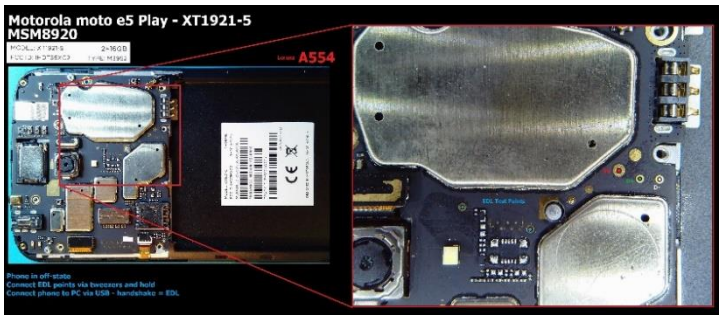


Figure 30 - Motorola moto E5 Play XT1921-5 Test Points

4.12.2 USB Taps are not EDL Test Points

If you notice, I mark the USB TAPs on many of my diagrams to include both of the Motorola diagrams here. There are a couple of reasons for that. One reason I began marking USB TAPs is for USB bypass operations on damaged

devices (discussed later). The other reason is to help distinguish between USB taps and EDL Test Points. I have seen diagrams on the internet marking USB TAPs as EDL Test Points. While USB TAPs can create EDL, they are very different. EDL created via USB TAPs is accomplished by shorting the D+ tap to the ground tap. When you do that with the USB TAPs on the logic board, you are essentially becoming an EDL cable. EDL cables do the exact same thing, they just short D+ to ground inside the USB cable. If the EDL cable doesn't work, shorting the USB TAPs won't work either (on all phones I have tested). EDL cables don't work on either of the Motorola phones in Figure 29 & 30, but they both have Test Points that create EDL. They can also both be forced into EDL via eMMC faults like most phones.

4.12.3 LG phones and Test Points – look for the sign of the cross

Many LG phones make finding Test Points easy. You will likely be stunned on how many LG phones have Test Points – old and new alike. All of them I have found have all been in the shape of a cross. (Figure 31) No other pins on the phone look like this. So the proximity and shape make them a dead give-away. All EDL TAPs work the same

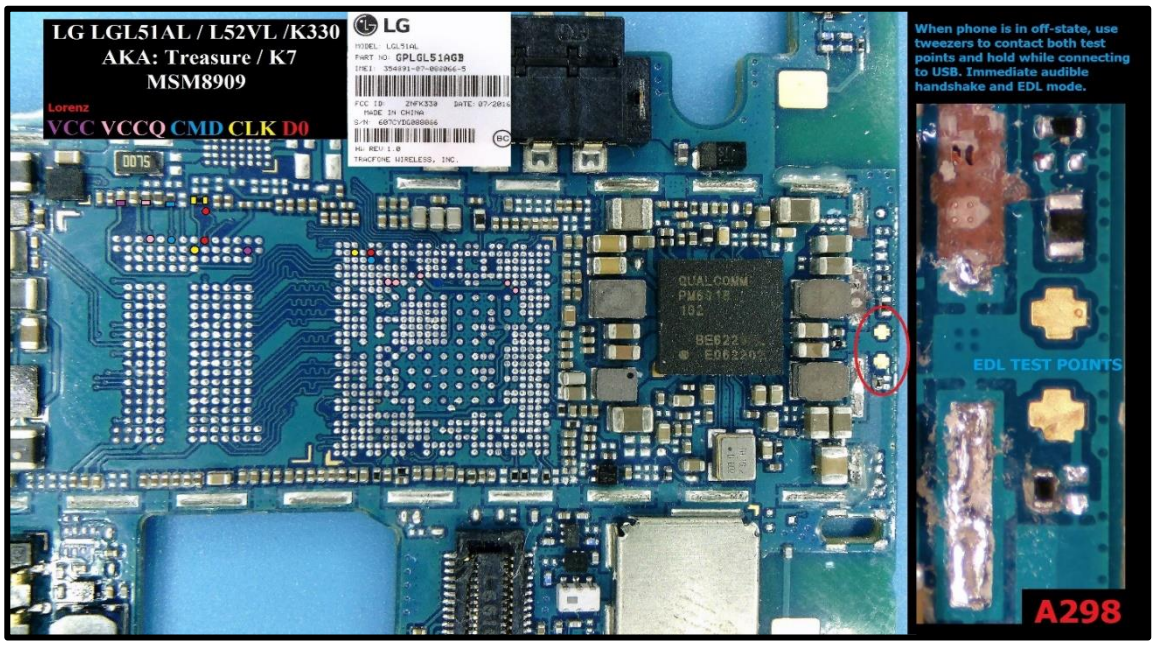


Figure 31 - LGL51AL - K7 Test Points

way, connect the two points (tweezers) and then plug into USB. I have listed many LG phones in a table coming up. You can follow the links to those diagrams for examples of LG Test Points.

4.12.4 Test Points are not grounded

EDL Test Points on LG phones not running Qualcomm processors will not produce EDL Mode. EDL Mode is a function of Qualcomm processors. In the world of phones, it is common to find many different phone models using the same logic board. This can create some confusion when trying to identify devices and determine what hardware the device is running. It is not uncommon for phones with the same name to run different processors. For example, some Galaxy S7's run Qualcomm processors and others run Exynos processors. LG phones are no exception. Some LG phone models with the same board run Qualcomm processors and some run MediaTek processors.

4.12.5 Non-EDL Test Points

Many LG phones running Qualcomm processors will have test points to create EDL Mode. When that series also runs a MediaTek processor, those test points might be missing, but sometimes they are still there. This can be confusing as LG test points really stick out and the processor may be obscured under a heavy welded heat shield. So an examiner may see test points and assume the processor under the heat shield must be Qualcomm. Just

remember the rule that EDL Test Points are not ground. Meaning that, with real EDL test points, neither of the crosses so easily recognizable on LG phones are grounded. It is easy to test with a meter but can also be seen with the naked eye. Some LG devices will remove the test points when they mount a Mediatek processor on the same board that may have been originally designed to run a Qualcomm processor. But that is not always the case. So if you fail to produce EDL using what looks like EDL test points on an LG phone, the test points may not be for producing EDL and the processor is likely not a Qualcomm.

4.12.6 Compare similar LG boards that run Qualcomm and MediaTek Processors

As you can see from these two series of LG devices that run either Qualcomm or MediaTek, sometimes the test points are removed (as on the SP320) and sometimes they are present but don't produce EDL. Without seeing the processor on the SP200, the clue that these are not EDL test points is that one of the crosses is grounded.

LG Phoenix 4 / Risio 3 / K8 (2018) / Tribute Dynasty	Processor	Test Points
LG LM-X210APM ZNFX210APM / ZNFX210FM / ZNFX210ULM / ZNFX210VPP	MSM8917	YES - EDL
LG SP200 Tribute Dynasty	MT6750	Non-EDL Test Points - No EDL
LG X Charge / X Power 2 / Fiesta	**Processor**	**Test Points**
LG L64VL / ZNFM322 / ZNFM320G / ZNFL63BL	MSM8917	YES - EDL
LG SP320 / SP320H	MT6750V	Test points removed

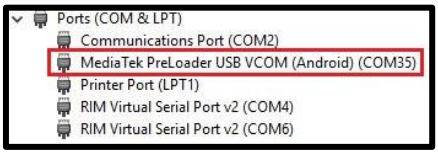
4.12.7 LG Test Points on phones with the same name and different Qualcomm processors

Again, this can be confusing because it is sometimes difficult to figure out what processor a phone is running based on online databases. There can be many phones with the same name but with different model numbers. Just from my homemade database I can query the Stylo 3, and get several hits with different model numbers. The phones have the same logic board and the EDL test points are located in the same place but they can either be running the MSM8917 or the MSM8940. Both of those processors currently have limited support for EDL extraction in the UFED and none of the Stylo 3 phones are extracting via EDL as of UFED 7.10.1.1080.

LG	L83BL Stylo 3 LTE	GSM	ZNFL83BL	MSM8917	EDL L83BL
LG	L84VL Stylo 3 LTE	CDMA	ZNFL84VL	MSM8917	EDL L84VL
LG	LGMP450 Stylo 3 Plus	GSM	ZNFTP450	MSM8940	EDL LGMP450
LG	LGMP450 Stylo 3 Plus Titan		ZNFTP450	MSM8940	EDL LGMP450
LG	LS777 Stylo 3	CDMA	ZNFLS777	MSM8940	ISP-LS777
LG	M400F Stylo 3	GSM	ZNFM400F	MT6750V	FCC M400F
LG	M400MT Stylo 3		ZNFM400MT	MT6750V	FCC M400MT
LG	M430 Stylo 3			MSM8917 / MSM8940	ISP-LS777
LG	TP450 LG Stylo 3 Plus	GSM	ZNFTP450	MSM8940	EDL TP450

4.12.8 LG Test Points on phones running MediaTek processor, what do they do?

They look like EDL test points, but they don't produce EDL Mode because the processor is not a Qualcomm processor. The test points on the LG SP200 creates Preloader mode on the Mediatek processor. This will trigger an immediate handshake and cause the "Continue" button to activate but of course this not EDL Mode and that is because you have connected a device with a MediaTek processor. If you have a locked MediaTek device but you think it is running a Qualcomm processor, the test points will serve the purpose of identifying the processor as a MediaTek device if you connect with Device Manager open. So at least you know you have a MediaTek processor not suitable for the EDL exploit.



4.12.9 Which is easier – eMMC shorting or Test Points?

EDL Points are easier than eMMC faults because no soldering is needed, the targets are bigger, and Test Points are generally found in accessible locations. Many times, you will not have to flip the logic board to reach the storage or processor. As with anything else, there are exceptions to that rule. If the Test Points are located on the flip side of a board, you may have to get creative when deploying the decrypting bootloader as you will need the battery in place for the phone to boot. So there are a few phones that eMMC shorting may be preferable, logistically, even though Test Points are available.

4.12.10 List of sample phones with Test Points – links to diagrams included

I am including a list of some of the phones I had in my inventory with Test Points. Most of the phones on the list are LG. Many phones don't have Test Points so with some brands, you may never come across Test Points. Each of these phones has a link to the diagram with Test Points and ISP points. As you can see from the list of phones, the LG phones have Test Points on older phones and newer phones. Some of these phones are not supported for EDL extraction by the UFED – yet – but likely will be. Many of the processors shown are on the potential up and coming list of limited support by Cellebrite.

Phone *(Some devices listed not supported for extraction via EDL in UFED)*	**Processor**	**Test Points**
LG L64VL X Charge	MSM8917	Yes
LG LS770 Stylo	MSM8916	Yes
LG H811 G4	MSM8992	Yes
LG-H860 G5	MSM8996	Yes
LG H918 V20	MSM8996	Yes
LG L52VL K7 / Treasure	MSM8909	Yes
LG L61AL K10	MSM8909	Yes
LG K428 K10	MSM8909	Yes
LG K120 K4	MSM8909	Yes
LG K371 Phoenix 2	MSM8909	Yes
LG L33L Sunset	MSM8916	Yes
LG L44VL Rebel	MSM8909	Yes
LG LS676 Tribute	MSM8909	Yes
LG LS777 Stylo 3	MSM8940	Yes
LG LS992 G5	MSM8996	Yes
LG M150 Phoenix 3	MSM8909	Yes
LG MS210 Aristo K8	MSM8917	Yes
LG L82VL Stylo 2	MSM8916	Yes
LG M257 Harmony K20	MSM8917	Yes
LG LS665 Tribute 2	MSM8916	Yes
Motorola XT1922-7 Moto G6 Play	MSM8920	Yes
Motorola XT1921-5 Moto E5 Play	MSM8920	Yes
Xiaomi Mi 5X	MSM8953	Yes

4.13 Creating eMMC faults (shorting)

Cellebrite always attempts to make the extraction process as automated as possible for any method used via the UFED. However, despite their best efforts, a certain degree of examiner interaction with a device is necessary with many extractions and especially with certain EDL extraction techniques. In order to take full advantage of Cellebrite's EDL exploit on as many devices as possible, it is necessary for the user to do more than just button combinations, EDL cables, or navigating menus in device settings on the target device. In fact, with EDL extractions it may be necessary to become intimately familiar with the device. Some methods require a significant amount of research, testing, disassembly, and some soldering.

Initiating eMMC faults, also referred to as “shorting”, is one of the most reliable methods to create EDL. To be clear again, using an EDL cable is shorting a device. EDL cables simply create a short in the USB cable by grounding the (Data +) line. However, do not confuse the (Data +) line in a USB cable with Data lines connecting the eMMC to the processor. They are not the same.

Creating EDL by shorting the exposed CMD, CLK, or Data locations on a device will normally always work whether or not other methods of creating EDL on that same device have failed. So if another method of placing a particular device in EDL works, eMMC shorting will also work on that device. If no other method of placing a particular device in EDL works, shorting will most likely still work. Most devices running Qualcomm processors can be placed in EDL Mode via eMMC shorting. Whether or not they can be extracted once in EDL Mode is a separate issue. Most “modern” phones with Qualcomm processor can be placed in EDL Mode via shorting.

Figure 32 - Motorola XT1650-02 Shorting

I have come across some devices in which locating places to short was very difficult and even more difficult to short once I found them. Click on the Motorola XT 1650-02 as an example. That phone is running an MSM8996, and locations that worked on other phones running that processor were not creating EDL for me. The phone was also running Universal Flash Storage (UFS), which doesn’t prevent shorting but it is not like eMMC storage (discussed in Section 4.17).

4.14 Removing the short before extraction

Shorting the flash storage (eMMC or UFS) is the only method that requires the examiner not only to create the condition that produces EDL, but then **requires the examiner to remove that condition before proceeding with the extraction**. With other methods requiring the user to create EDL, the action that created EDL (button combos, cable #523, or Command Prompt) creates a temporary or momentary condition that creates EDL, which Cellebrite can remove during its extraction process. Cellebrite’s cable #523, has a spring-loaded pressure release button to create the short that produces EDL. EDL is visible when the button is released, thereby allowing the UFED or Windows to detect a device in EDL Mode connected via USB. The phone is in EDL Mode, but the short is removed by releasing the button.

With eMMC shorting, you must create the short, apply power that creates EDL Mode to the device, and then remove the short from the device after EDL is created. The device will not extract while shorted even though it is in EDL Mode. I have only encountered one exception to this rule. Thus devices that are damaged and are permanently in EDL by accident or intentional destruction, will not be extracted until that short or damage is located and removed or repaired. ***A common error for EDL extractions is that examiners either forget to remove the short that created EDL, or have accidentally damaged a device or created a bridge when soldering on the device. When this happens and you hit “Continue”, the device will generally not proceed past Step 1, and will eventually fail with an extraction error***. Failing to remove the short has never permanently damaged a device for me, but you will have to start the process again. Disconnect the device, remove battery or reset, and then short again. See these PP slides (Shorting eMMC – Solder & Clip Method).

4.15 Testing for solder bridges or other damage that creates permanent EDL Mode

Sometimes and examiner may not be aware that damage has occurred or a solder bridge has been created. This typically happens during disassembly or soldering to an ISP location. This can create a frustrating issue for hours or days for examiners. The reason is that when the bridge or damage creates a short, the result will be EDL Mode. EDL Mode just happens to be what the examiner is trying to achieve. So if an examiner creates a solder bridge during the course of soldering a wire to the CMD or CLK pin, the examiner will then use the wire to short the device.

When the examiner connects the device to USB, it will be in EDL Mode. The examiner then removes the short by unclipping the wire from ground and begins the extraction. The extraction will fail because the examiner is unaware that the device is still in EDL Mode. It is still in EDL Mode because of the solder bridge that was unknowingly created. So the examiner will typically connect the ground wire again to create EDL and go through the process again with the same failing result. The phone is never coming out of EDL Mode.

So when you have a device with a widely supported processor or one that you know is supported for an EDL extraction that fails to extract, the examiner should test the device to make sure it is not in permanent EDL Mode. Do this by removing the battery and of course removing the ground wire or whatever you have used to create EDL Mode. Connect the device to USB. With no battery and with your short removed, the device should not be in EDL Mode. If it is in EDL Mode, you know there is something else causing the eMMC fault. You must locate and repair the device or remove the short before you can extract the device.

4.16 Phone disassembly

Depending on the model of the device, there can be significant disassembly involved with this method of creating EDL. With some devices, it may be necessary to use heat to remove screens or back covers of devices, in order to access the logic board. The heat shields on some devices can also be robust and require heat and/or destruction to remove. To illustrate this further, watch Moto Heat Shield Removal Video and Moto Heat Shield Removal Close Up (See Moto Heat Shield Removal Video). This is where damage to a device can occur. Resistors and capacitors can be accidentally dislodged when removing heat shields or unseen shorts can occur. On some devices, Data, CLK, and CMD lines run just below the surface of the logic board under a thin coating. These lines can be severed by careless use of a razor or other tool used to pry off heat shields.

Figure 33 - Soldering to ISP locations to create EDL Mode

Experience with phone disassembly and repair, ISP, JTAG, and Chip-Off is helpful for this method. I recommend using a microscope for this procedure to ensure accuracy and to avoid inadvertently damaging the device.

4.16.1 Shorting ISP points

Many examiners have not had ISP training so I understand the confusion and questions I get regarding “shorting”. So generally, when the term “shorting” is used, it is referring to shorting the same locations examiners use when pulling a device via In-System Programming (ISP). These are not the same locations used for JTAG proceedures. JTAG taps will generally not create EDL Mode when shorted. That may also be confussing because JTAG taps look like Test Points, found on some devices, that can be used to create EDL Mode.

Shorting the Data lines connecting the processor to the eMMC or shorting the Command or Clock line is the most effective, reliable way to create EDL on devices running Qualcomm processors. When EDL cables, button combinations, and ADB commands fail to place devices in EDL Mode, eMMC shorting will most likely always work. Whether supported for extraction or not, most Qualcomm processors can be forced into EDL Mode via eMMC shorts. And by the way, that also includes shorting devices with Universal Flash Storage (UFS) chips found on newer phones.

4.16.2 Why ISP, JTAG, and Chip-Off training and techniques are still relevant

With more and more devices encrypted by default, fewer examiners receive training on ISP, JTAG and Chip-Off. Also, as a result of encryption, fewer pinout diagrams of devices are created because pulling the data via ISP and

Chip-Off will result in an encrypted extraction if the phone is encrypted. However, there are good reasons why training in these areas is very useful and relevant today:

- Many phones released with Android 6, 7, and 8 are still released without being encrypted by default.
- Feature phones that can be exploited by JTAG are still manufactured and sold today.
- The world is full of older devices that are not encrypted and still in use or are not in use but contain evidence of a crime. These phones can be exploited by JTAG, ISP, and Chip-Off.
- Many EDL extractions require phone dissassembly and shorting ISP points to create EDL Mode.
- Certain flasher boxes require an ISP connection to a device to remove FRP locks from devices and other functions.
- Chipping and pinning modern processors will allow you to access badly damaged devices using exploits like EDL – even if the devices are encrypted (examples to follow).
- The skills and equipment for ISP, JTAG, and Chip-Off are necessary to take full advantage of exploits like EDL.

The Z799VL aka Z798BL or Z899VL is a phone that was challenging and generally untouchable (if locked) for a while before Cellebrite's release of EDL decrypting extractions. These are the Majesty Pro and Majesty Pro Plus devices. The Z799VL and Z798BL were encrypted by default, while the Z899VL was not encrypted by default. They all have the same logic board and pinout and all must be shorted into EDL Mode because no other methods work to create EDL with these three models. Before the EDL exploit, I was able to access data on the locked Z899VL via ISP or Chip-Off. Having the ISP pinout for these devices and knowing how to dissassemble and short them meant I could take immediate advantage of the EDL exploit released by Cellebrite.

4.16.3 Pinning devices and locating EDL short points

To get a good idea of how devices work behind the scenes I always find it beneficial to understand where everything is located and how it is connected. To do that, I like to take test devices and chip and pin them, going beyond the usual ISP pinouts. I have done that with Majesty Pro series of devices. Pinouts showing the locations of ISP points have proved useful for EDL extractions. EDL cables, button combinations, and ADB commands will not work on all phones. When those options do not create EDL Mode, "shorting" or creating eMMC faults will be necessary.

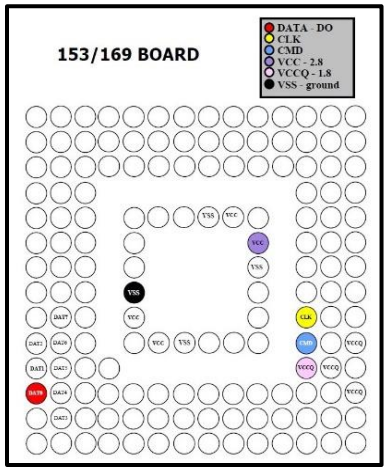


Figure 34 - eMMC Chip pinout

Pinning devices for ISP can be done by chipping off the memory (eMMC) of a test device and using a multimeter to trace known locations under the eMMC to exposed locations on the logic board, so that direct access to the eMMC can be achieved without removing the chip on the target device (the evidence). These pinouts are to what examiners are referring when asking for "pins" or a "pinout" of a particular device, either for an ISP extraction or locations to "short" for EDL extractions. There are many different types of storage and BGA layouts. There are three primary BGA patterns found on the vast majority of devices. Clicking on the BGA diagram (Figure 34) will take you these diagrams in the MDFA resource folder.

The pins needed for an ISP pinout include the Data0 or **D0**, Command (CMD), Clock (CLK), VCCQ (1.8 volts) or VCC (2.8 volts). These same locations are present under the processor with the exception of the VCC, which is usually not found under the processor. For purposes of EDL, VCC and VCCQ should be avoided. Shorting voltage pins can produce EDL Mode, but it can be dangerous as well. For extracting devices via ISP, Data0 (D0) is the only data line that is needed, however there are eight (8) data lines (D0 – D7). There are some flasher boxes that can take advantage of the other data lines to extract data faster from a device via ISP. The vast majority of pinouts available today, only include the Data0 (D0) pin locations.

4.16.4 Finding ISP points without chipping a test device

Kim Thomson wrote an excellent paper on ISP Recognition. That is, locating ISP pins without chipping a device (**ISP Recognition – Kim Thomson**). Most of the time I can find a location to short devices into EDL without chipping or seeing a pinout. I do that using some of the techniques Kim talks about in his paper and just recognizing patterns I know exist on particular phones and processors. Many devices that run the same processor have similar patterns regarding board layout and ISP locations. I created a couple of diagrams that show patterns that exist on devices running the **MSM8909** and the **MSM8916**.

4.16.5 Which ISP points create EDL?

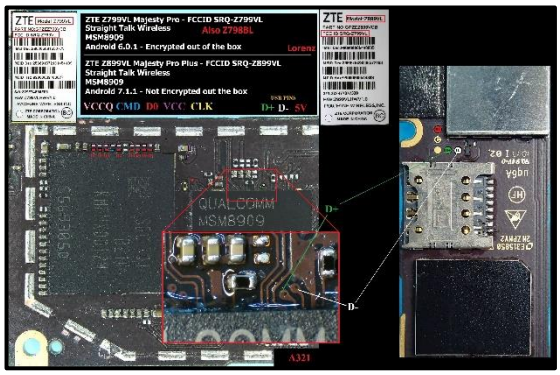


Figure 35 - ZTE Majesty Pro Series ISP Pinout

All of them can. When looking at an ISP pinout, you will generally have a choice of possible locations to short to create EDL Mode on the phone. All ISP pin locations (D0-D7, CMD, and CLK) can create EDL when shorted. The CLK pin can create DFU Mode on some phones as I discussed under the FTM and DFU section of this paper. In many of my previous posts and papers, I often use the command (CMD) location to short to create EDL, but it is not necessary to restrict yourself that that pin. You can choose a pin that is easiest to access and solder. I have not tested every pin on every phone but I purposely marked and tested all of them on ZTE's Majesty Pro series of phones. All of the locations (D0-D7, CMD, and CLK) produced EDL when shorted. As you can see by the diagram, the data pins are easier to access and solder than is the CMD pin. When it comes to EDL Mode, there are no "degrees" of EDL. EDL created by the CMD pin is no different than EDL created by the CLK or Data0 pin.

4.16.6 What is the difference between the ISP data lines and data (D+) line in a USB cable?

As shown in the Majesty Pro diagram, I have marked the USB taps. Regardless of the type of USB cable, whether it is micro USB or USB-C, there are only 4 lines of concern if I want to exploit or bypass USB for device extractions. The D+ and D- line communicates with the processor and runs from the USB port to the processor. The D+ line is the green line inside a USB cable and I mark it as green in my diagrams. The D- line is white and I mark it accordingly in my diagrams. The D+ line is the line that is shorted to create EDL Mode on devices that support that protocol. But that protocol can be turned off by manufacturers and that is why EDL cables don't work on all devices.

The Data lines D0 -D7, run between the device storage and the processor. So when the processor receives information from the device storage (eMMC or UFS for example) that data is sent out the USB port through the USB Data lines (See the eMMC BGA221 & MSM8917 diagram). Information from the device storage goes through the processor and out through the USB data lines to the UFED. Shorting the Data lines between the processor and the storage is referred to as creating eMMC faults or shorting. Shorting those lines, along with the CLK and the CMD, are the methods most reliable for creating EDL.

Shorting procedures and equipment

I think there is a lot of confusion and misunderstanding regarding eMMC faults and the procedures surrounding shorting. Most of the questions I get involve this issue or some problem resulting from attempting to short devices. Shorting the ISP points on a device is fairly simple compared to actually pulling the device via ISP, which require multiple connections. By shorting the Data, CMD, or CLK locations, you are simply connecting those points to ground and then plugging in a USB cable or holding the power button. The Qualcomm processor then detects an issue caused by the ground and immediately diverts to EDL Mode. As long as the device stays powered at that point, it will remain in EDL Mode. But before Cellebrite can exploit EDL Mode that you created by shorting the device, **you must remove the short that created EDL Mode**. I think this is where some confusion occurs. By

removing the short, you are not taking the device out of EDL Mode. You are simply removing the condition that put the device in EDL Mode. Behind the scenes, the UFED will need to control the device and it cannot do that if the device remains shorted.

When creating eMMC faults by shorting, you are connecting a wire to one of the ISP locations already known or mapped out on that device. The other end of that wire is then connected to any part of the phone that is grounded. Heat shield frames are what I use frequently for this. If I am using the clip method, I will clip the wire to the heat shield frame after soldering the other end of the wire to an ISP point. Of course, you do all of that with the phone in the off-state with the battery removed and disconnected. With the short created, applying power to the phone will send the device into EDL Mode.

4.16.7 Is soldering necessary?

When creating eMMC faults, I frequently solder. It is easier to control and I am comfortable with my soldering skills. It is also easy to cause damage using a needle. Too much pressure with a needle can dislodge or damage the resistor. If you have never soldered anything before, I recommend practicing on something that doesn't matter

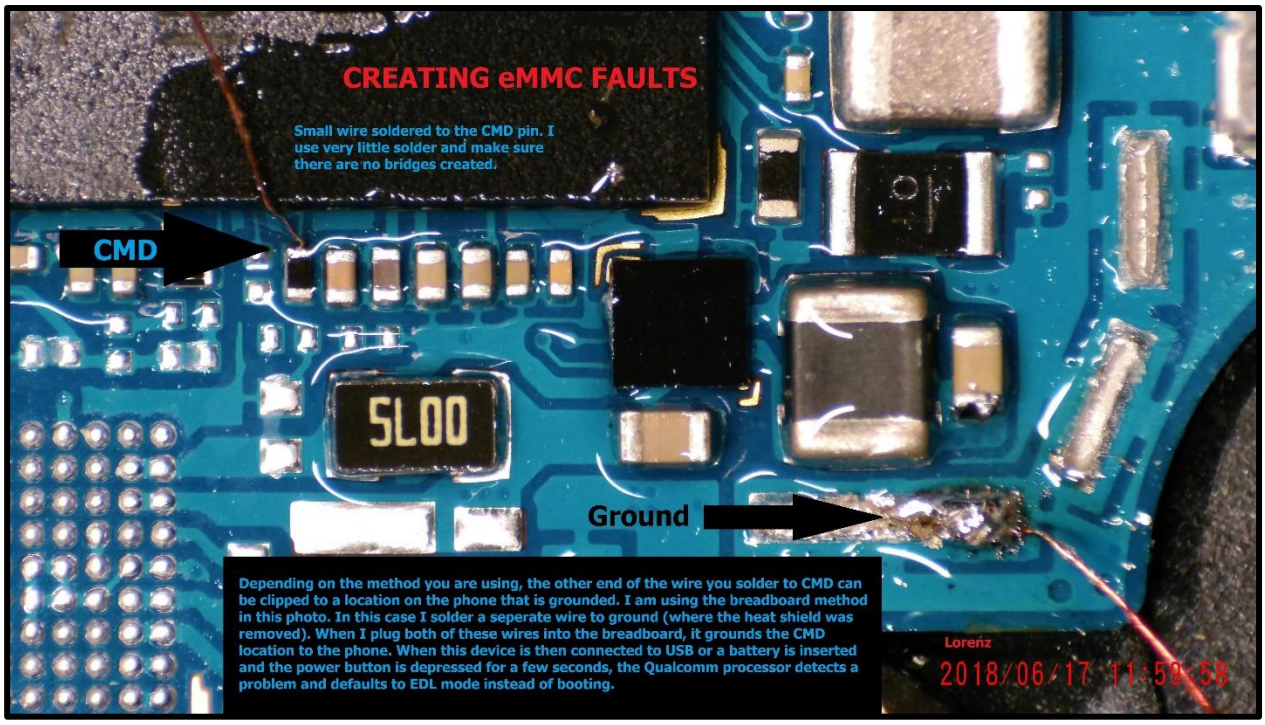


Figure 36 - Creating eMMC Faults via Soldering to ISP Points

first. Most ISP locations are very small and a microscope will be needed. It is possible to operate without a microscope in some situations where large pads have been identified and mapped, but errors can occur during soldering which can affect the ability of the UFED to extract the device. Bridges can be created accidentally which keep the device permanently shorted and thus will fail to extract until that bridge is removed.

There are three general methods of shorting that I will cover here. Let me be clear that these are not the only methods and may not be the best. I have my preferences based on the equipment and experience I have. Other examiners may have a different method that works well for them. I am including a link to the diagrams for these methods. One of the diagrams (breadboard method) does a visual step-by-step walkthrough of an extraction demonstrating when and how the short is applied relative to the extraction process and requests in the UFED. All of these diagrams will be in the resources folder on the Mobile Device Forensics and Analysis forum.

4.16.8 Shorting Methods - Diagrams:

- Needle Short Method
- Clip Method
- Breadboard Method
- eMMC Shorting with Tweezers

4.17 Shorting and pinning devices with Universal Flash Storage (UFS)

Shorting devices with Universal Flash Storage (UFS) is procedurally the same as shorting devices running eMMC storage. What makes this endeavor more challenging is that there are really no pinouts created for phones running UFS storage. So without reliable pinouts for locating ISP points as with eMMC chips, it means fishing for those points is necessary.

UFS is an advancement from eMMC chips. Because UFS chips were designed much differently that eMMC chips, ISP was not possible with them like it was with eMMC. Therefore, there were no pinouts created that tell us the location for ISP. I remember the first devices, of which I was aware and cared about, that took advantage of UFS storage was Samsung Galaxy - specifically the Galaxy Note 5 and the Galaxy S6. Those devices were running UFS BGA095 chips. Dediprog manufactured a chip reader and adaptors for UFS chips and I purchased one in late 2016 with a UFS BGA095 and BGA153 adaptor. The Galaxy Note 5 and S6 were not encrypted by default and thus we could chip those devices and obtain a physical as with eMMC chips. I used my UFS BGA095 adaptor many times on the Note 5 and S6 before there was a non-chipping exploit developed.

There was no ISP solution for UFS chips that developed and soon after UFS emerged, default encryption began to take hold. The few times I used my UFS BGA153 adaptor, was on test devices and they were all encrypted. Whether ISP was possible with UFS chips or not, there was no point with default encryption. Fast forward to today and we want to short ISP locations to take advantage of Cellebrite's EDL exploit, but we don't know where to short devices running UFS chips. But those locations exist and can be located.

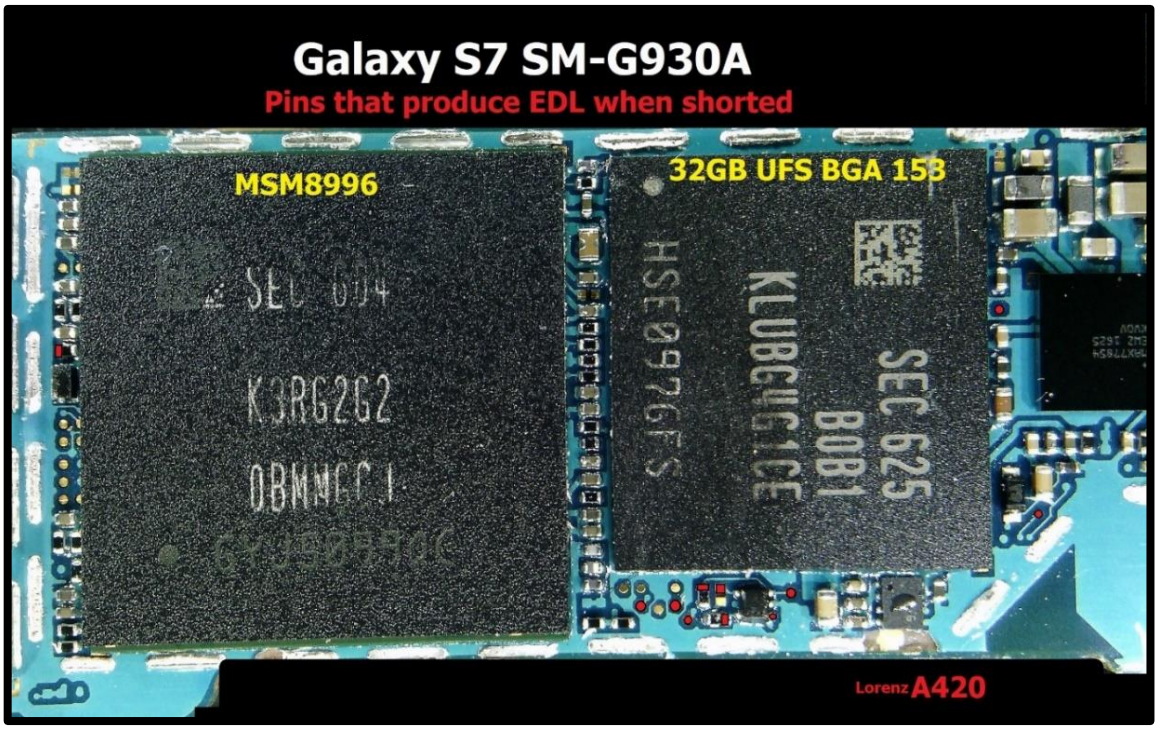


Figure 37 - Galaxy S7 SM-G930A - EDL Short Locations

4.17.1 Samsung Galaxy S7 UFS BGA153 with MSM8996

Except for the older phones running UFS BGA095, nearly all phones you encounter today running UFS chips will be encrypted by default (See Section 3.2.8). Most of the UFS chips I run across are all BGA153. Many Samsung Galaxy S7 devices run the Qualcomm MSM8996 processor and the UFS BGA153 storage. Just like phones using eMMC storage, there are numerous locations that, when shorted, create EDL Mode. Cellebrite can use the EDL exploit to obtain decrypted physicals from some S7's running the MSM8996, via ADB. That means the device must be unlocked and/or USB Debugging enabled. At the writing of this paper, Galaxy S7s shorted into EDL were failing to extract using the UFED decrypting method. I am including a diagram of the Galaxy S7 that shows all of the short locations I located that produce EDL (Figure 37).

4.18 Reverse pinning processors and UFS to locate short points

I have pinned many processors on test devices for ISP and JTAG in the past for purposes of helping me get into other devices running that same processor. Knowing the location of JTAG and ISP points on a processor is very useful. Is that still true with devices running UFS storage? Yes, in some situations it is. There are some devices I have encountered in which I had a hard time finding short locations just by fishing and probing. Knowing points under a processor or a UFS chip that have been pinned and marked as connected to short locations on other devices can help when trying to locate short locations on a device running that same storage or processor. As I mentioned earlier, there has been much less work on pinning newer phones running eMMC storage due to encryption because an ISP extraction of an encrypted device would be useless for collecting user data. The same is true for devices running UFS storage, which will almost certainly be encrypted and because of their architecture, ISP will not work on UFS – that is, it was never researched very much because of encryption.

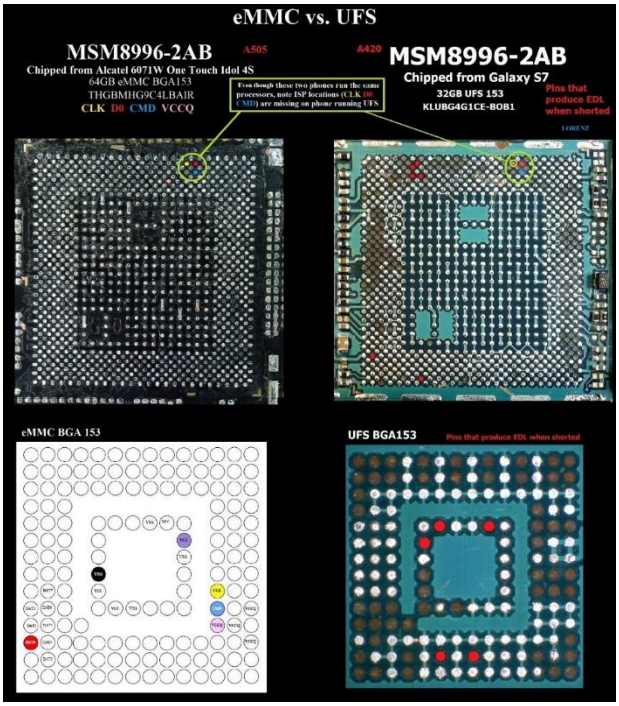


Figure 38 - eMMC vs. UFS - Pinouts

I am including a diagram comparing the eMMC BGA153 to the UFS BGA153 (Figure 38). They look identical when chipped but they are completely different. This diagram also provides a visual of how the same processor can be differently configured if that processor is used on a device running eMMC and a device running UFS. So I can still pin UFS storage, but I am not identifying the pins as CLK, CMD, or Data. I am only identifying the pins as creating EDL Mode when shorted. That is all that is really useful at the moment with UFS. The same is true of processors mounted on phones running UFS storage. The MSM8996, is what I use as an example to show that pin configuration is based on the storage when it comes to locating short locations for EDL.

5 Troubleshooting EDL extractions and advanced methods made possible by the EDL exploit

There are many things that can go wrong and cause an EDL extraction to fail. Most of them are minor and harmless. There are some things that can go wrong and destroy the device and thus the evidence will be lost. I think the solution to both major and minor issues are both the same – attention to detail and preparation.

Some issues related to extraction failures are not the fault of the examiner. Examiners will receive devices that range from virtually new to completely destroyed. Sometimes working devices may have an unseen issue that

prevents extraction. The failure can be misdiagnosed as not supported when in fact it is something that can be corrected or bypassed all together.

5.1 Attention to detail

I am confident this is an issue because of the questions I get and because I have done it myself. If you get in a hurry and try to rush through, sometimes minor details can be overlooked and the extraction will fail. When the extraction fails, we often make assumptions that may not be correct.

5.1.1 Check for EDL and check that EDL can be removed

The same is true with shorting devices into EDL Mode. Remember that with eMMC faults, you must be able to remove the short that creates EDL in the first place. I get many questions from examiners who do everything correctly but the device will just not extract. Because I have been there myself, I can usually give them some ideas. A common error occurs when an unseen bridge is created. This occcurs via some overzealous soldering or the wire used to create the short becomes damaged and does not create a short or gets pinched when reassembling a device for an extraction and creates a permanent short that remains even after the clip is removed.

When an extraction fails in Step 1 in the UFED when using eMMC shorting, close UFED 4PC if that is what you are using. Connect the shorted device and see if EDL is appearing in Device Manager. If it doesn't appear, you are not creating EDL in the first place and the extraction will fail. If EDL appears in Device Manager, remove the short and the battery and then connect the device to your PC again. This time you should not see EDL Mode. If you do, you have a bridge or the wire with the clip is pinched or exposed. Many times examiners will check for EDL, but will not check to make sure EDL can be removed. A permanently shorted phone will not extract.

5.2 Preparation

Do as much research on a device as you can before attempting an EDL extraction. Especially if it is one that requires dissassembly or shorting. Determine what processor it is running and make sure shorting is the only method to create EDL on the device. Is the device encrypted? An EDL extraction on a device that is not encrypted is much easier than using the decrypting method on some devices, as was discussed in this paper. Contact others on the forum who can provide some insight on everything including, encryption, short locations, methods, tools, and dissassembly.

There are things that can go wrong during dissassembly if not done carefully. I think this area is sometimes overlooked. I specifically created a video on removing heat shields a few months ago after some requests for that. Sometimes getting access to the locations to be shorted can be the most challenging part of the extration. Research and watch teardown videos on the internet. Beware that sometimes people can leave out steps so there's nothing wrong with watching multiple videos on the same device. Most teardowns videos will not include removing heat shields and other things that only forensic examiners would want to do.

5.3 Get test devices

It is sometimes difficult to convince supervisors, adminstrators, and prosecutors that you need to buy a phone so you can destroy it and/or practice what you are about to do. I have many people who will just give me phones because they know what I do with them. I get a lot of donations to my cause from people who know and trust me and also receive old evidence to be destroyed.

Andrew Rathbun provided a good idea that may provide a way to feed test phones to the forensic examiners at police departments. In his particular example, abandoned phones are brought as lost and found and must be stored for 90 days before they can be turned over to either the local domestic violence shelter or to the digital forensic unit to be used as test devices. Most departments have general orders or standard operating procedures for disposal of abandoned property that could be modified to benefit the digital forensics unit. Reach out to local libraries or any other institutions that may have a lost and found and ask if you may included in their workflow once a phone has gone unclaimed for an extended period of time.

5.4 Common extraction errors and error messages (10-12-18)

Like any other type of extraction, EDL extractions can and will fail. There are a variety of different reasons and causes for these failures. Some of the failures are because the device is not supported for an EDL extraction. Some of the failures are caused by user error.

There are frequent questions about why extractions fail and questions about the meaning of messages in the UFED. So I decided the best way to test, explain, and troubleshoot error messages is to intentionally create errors to demonstrate issues caused by improper procedures and connection of unsupported devices.

10-11-18

Error	Device	Method	Intentional Error Created	When Failed	Error Message in UFED
1	ZTE N9136 / MSM8909	Generic Qualcomm EDL (Recommended)	**Device Not in EDL Mode When Started**. Connected in off-state, charging indicator enabled "continue" button	Connecting	Verify that USB Debugging is enabled on the source device and then press Continue. If USB Debugging cannot be verified press Abort
2	ZTE N9136 / MSM8909	Generic Qualcomm EDL (ADB)	**Failed to enable USB Debugging.** Connected unlocked device in on-state. "Continue" button enabled when connected	Connecting	Extraction Error / Cannot read phone memory (9)
3	Motorola XT1922-7 / **MSM8920**	Generic Qualcomm EDL (Recommended)	This device was known as **not supported for EDL extractions**. Verified connected in EDL Mode.	Step 1 out of 2	Process Failed / There are no additional bootloader methods available
3	Motorola XT1922-7 / **MSM8920**	Generic Decrypting Qualcomm EDL (Recommended)	This device was known as **not supported for EDL extractions.** Verified connected in EDL Mode.	Step 1 out of 6	Process Failed / There are no additional bootloader methods available
4	Oppo R9S / MSM8953	Generic Decrypting Qualcomm EDL (Recommended)	**Improper method of creating EDL.** Failed to connect device in off-state and wait for charge indicator to appear.	Step 6 out of 6	Extraction Error / Cannot read phone memory (9)

5.4.1 Error # 1 - Device not in EDL Mode when the extraction starts (10-11-18)

Of course, the first step in a successful EDL extraction is to actually get the device in EDL Mode. Either the examiner will do this themselves or, for unlocked devices, allow the UFED to place the device in EDL Mode – if that action is supported on that device. There are times when we think the device is in EDL Mode but it is really not. We navigate to the "Waiting for device" screen, connnect the phone, and the "Continue" button becomes active. For this scenario, I selected the **Generic Qualcomm EDL (Recommended)** option and navigated to the **"Waiting for device"** screen.

I then used a ZTE N9136 and connected it to the UFED in the off-state with the battery inserted. I intentionally made no attempt to put the device in EDL Mode. Like many phones, the charging indicator will appear on the screen and then the device will be recognized by the UFED 4PC or the UFED Touch and a handshake will occur. This will be audible on the UFED 4PC and this handshake will cause the "Continue" button to become active on both the UFED 4PC and UFED Touch. The device is NOT in EDL Mode. What happens if we press "Continue"?

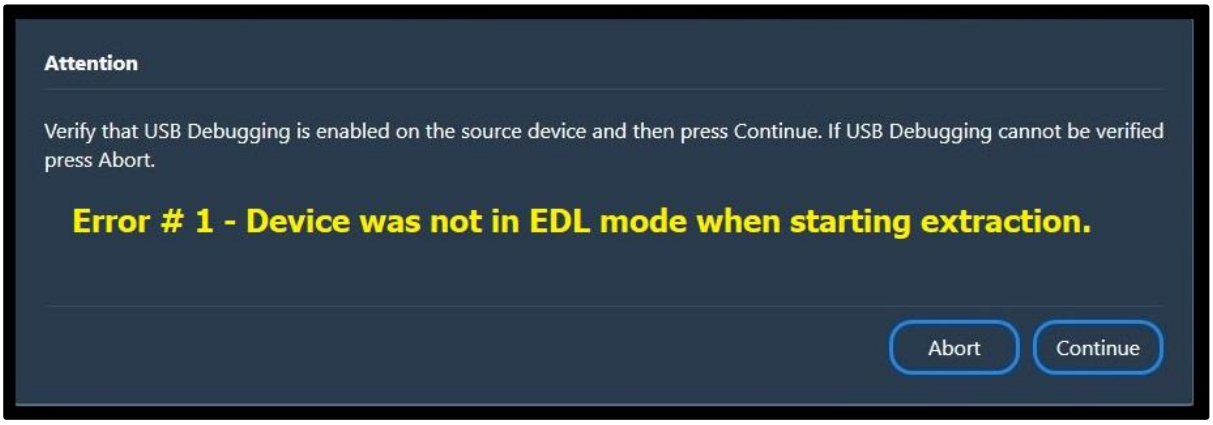
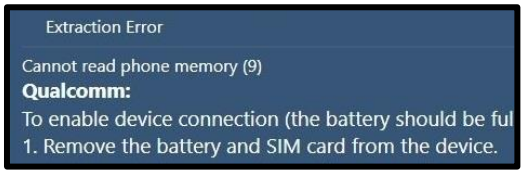


Figure 39 - Verify USB Debugging Error

We will see the "Connecting" screen and hear a series of handshakes but we will end up with the message above, ***"Verify that USB Debugging is enabled on the source device and then press Continue. If USB Debugging cannot be verified press Abort."*** We selected the (recommended) EDL extraction and that means we must connect a device in EDL Mode. We failed to do that so the extraction fails. What happens if we hit continue again here? Extraction Error Cannot Read Phone Memory (9).



It is important to note that many things can cause the "Continue" button to become active. As an absurd example to prove that point, I selected a non-decrypting Generic Qualcomm EDL extraction in the UFED and navigated to the "waiting for device" screen. I then plugged in my iPhone X in the on-state. The "continue" button became active and after a short time a message appeared asking me to "Verify that USB Debugging is enabled…"

The message, ***"Verify that USB Debugging is enabled…"*** is misleading from the user's perspective, especially if you are trying to extract a device that is locked and the UFED button selected clearly indicates it supports a lock bypass, which means it will extract devices that are locked with no user access to Developer Options. There are two lessons to take away from this experiment.

1. If you select an extraction method with Lock Bypass ability, and you get a message to enable USB Debugging, something is wrong. In this case I know the reason there was an error because I intentionally created it. The device was never in EDL Mode.

2. Just because a connected device triggers the "Continue" button, that does not necessarily mean the device is in EDL Mode.

5.4.2 Improper use of Cable #523 fails to create EDL Mode and causes the same USB Debugging message (10-12-18)

Remember that I intentionally connected a phone not in EDL Mode in the first example. But there are other issues that can fool examiners into thinking they have a device in EDL Mode when they actually don't. Improper use of

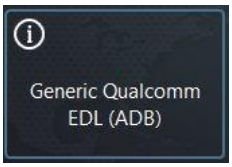
cable #523 will cause the same issue and the same error message. Simply plugging in an EDL cable and then pushing the EDL button may actually cause a handshake and cause the “Continue” button to become active, but the device is not in EDL Mode.

Normally the EDL button on cable #523 must be depressed before connecting the device to the UFED. Connecting the device with the D+ line shorted is what creates EDL Mode. Releasing the EDL button is when the handshake occurs because now the device can say hello and be recognized as in EDL Mode. You can connect a device with an Exynos processor and use the EDL button to repeately handshake with the PC. Aside from creating a bunch of handshake beeps, you are doing nothing because only Qualcomm devices can be placed in EDL Mode. The point is that pushing the EDL button, hearing a handshake, and seeing the “Continue” button become active does NOT mean your device is in EDL Mode. Check your procedure and timing for button pressing and cable connecting. Practice with the UFED closed and device manager open.

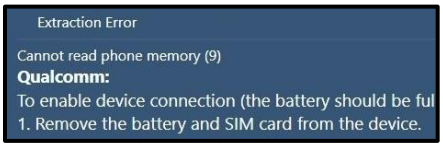
5.4.3 Error # 2 - Asking the UFED to create EDL Mode but failing to enable USB Debugging (10-12-18)

Lets take an unlocked device with a Qualcomm processor that is supported for an EDL extraction. If the device supports allowing ADB commands to create EDL Mode, why not let the UFED put the device in EDL Mode for us. This is actually the easiest EDL extraction to perform. Whether the device is encrypted or not, allowing the UFED to control every step from creating EDL Mode to the dump is better. This is because we don’t have to worry about timing and connection issues that can be tricky if the examiner is involved in the process. This is especially true when selecting decrypting extractions that require coordination and booting. Unlocked devices means that the examiner can turn on Developer Options and enable USB Debugging and other settings as instructed by the UFED. But what happens if I intentionally fail to enable USB Debugging and then ask the UFED to create EDL Mode for us?

I used the same ZTE N9136 and turned on Developer Options but intentionally failed to enable USB Debugging. I then used the Generic Qualcomm EDL (ADB) option in the UFED menus – the middle button. Remember that this is the button we select when we want the UFED to put the device in EDL Mode for us. The instructions clearly direct us to “Enable Stay awake and USB deugging modes” but I failed to do that. What happens when I connect the device in the on-state and press “Continue”?



After being warned that a device restart may be needed, we are prompted to press ”Continue” again, which I did. The UFED then tries to command the device to reboot into EDL Mode. If using UFED 4PC, we will hear the UFED continually issuing this command but because USB Debugging is not enabled, the extraction will fail. However, we DO NOT get a message that tells us to “Enable USB Debugging…” we just get the standard “Extraction Error Cannot read phone memory (9)” message.



Remember that you are connecting a device in the on-state, so if the device is still in the on-state and never rebooted, USB Debugging was likely not enabled. Had it been enabled, you would see the device shut down and then reconnect in EDL Mode. The screen will be black. So if you have asked the UFED to create EDL for you on an unlocked device and you get this message (Cannot read phone memory (9)), make sure USB Debugging is turned on, disconnect the device, navigate back to the “Waiting for device” screen using the same menu selections and try it again.

Something different will happen this time since you have followed the instructions regarding USB Debugging. You will normally be asked to if you want to "Allow USB Debugging" on the device screen when the UFED tries to communicate with the device after you hit continue. You will get this message if you attempt to communicate with the device via Command Prompt also. You may also be taken to a screen that asks you to select what USB mode you want. Select MTP or File transfer if that happens. When you do that and hit "Continue", the extraction will succeed – after the device reboots into EDL Mode as commanded by the UFED. Not seeing the "Allow USB Debugging" request during this type of EDL extraction might be a clue that you haven't enabled the proper settings in Developer Options.

So ironically, when there is an error with this type of extraction, we don't get an error message that instructs us to enable USB Debugging even though we are trying to extract a device using that very method via EDL. But we do get the "Enable USB Debugging" message when trying to perform an extraction with a lock-bypass method that does not require USB Debugging to be enabled – as in the first Error # 1, I created.

It can be confusing to users who haven't had experience with many EDL extractions using the UFED. I cannot explain exactly why this is, except that, remember I am conducting these tests under the "Generic" menus. In the generic menus, Cellebrite is allowing users to try devices and procedures that are untested and/or not officially supported using a variety of different combinations and techniques. So unlike an extraction under a specific device profile that has been tested by Cellebrite with specifically tailored instructions for that device, we are freestyling in the Generic menus and thus may not always get an error message that makes sense based on what we are attempting.

5.4.4 Error # 3 – Device in EDL Mode but not supported for extraction by the UFED (10-12-18)

As I explained in previous sections of this document, many devices running Qualcomm processors can be placed in EDL Mode. Only devices running "widely supported" procesors or specific devices running processors with "limited support" can be extracted by the UFED. Of course we don't always know for sure if the device we have is supported for extraction, so we place the device in EDL Mode and attempt the extraction. If the extraction fails, how do we know it failed because it is not supported or if the failure was the result of faulty procedures? I intentionally selected devices that I know are not supported for extraction by the UFED and attempted to extract them.

The Motorola MotoG6 Play XT1922-7 is running an **MSM8920** processor. This processor is not supported for extraction by the UFED. I also know this phone is not supported for an EDL extraction by the UFED. I have identified test points on the device that reliably place the device in EDL Mode. The device is encrypted by default but I attempted to extract the device using the Generic Qualcomm EDL (recommended) and Generic Decrypting Qualcomm EDL (Recommended). Both extractions failed in Step 1, which is normal for devices that are in EDL Mode but not supported for EDL extraction in the UFED. So the user will connect the device to the UFED while holding tweezers on the test points with the battery connected. This will result in an immediate audible handshake and the device will be in EDL Mode. The "Continue" button will become active. After selecting the "Continue" button you may see the "Connecting" message, followed by "Initializing step 1 out of 2." if using the non-decrypting method.

Figure 40 - No Additional Bootloaders

It will be “Initializing step 1 out of 6.” if using the decrypting method. The extraction will then fail with the message “**Process failed / There are no additional boot loader methods available**”. All of this means that you were sucessful in placing the device in EDL Mode and the UFED was able to see it and attempt an EDL extraction. After exhausting all of its generic options, the UFED did not have the ability to exploit EDL Mode on this particular phone and/or processor, so the extraction fails.

5.4.5 Error # 4 – Supported devices extracting via non-decrypting EDL method but fails with decrypting EDL method (10-12-18)

The Oppo R9S is running an MSM8953 processor and is supported in the UFED for a decrypting EDL extraction. But the instructions on this device are very specific. I used this device to demonstrate in the 2nd EDL webinar on 9-12-18. This device can be placed in EDL Mode with the Cable #523 and by pressing Volume Up and Volume Down. Pressing Volume Up and Volume Down while connecting the device to USB will create EDL Mode. This method matches the instructions on many other devices and this procedure will either create EDL Mode or DFU Mode on various devices. For decrypting extractions the device must fully boot in order for the decrypting bootloader to be loaded on the device. If the device fails to boot, boots to a charge-only mode, or the timing of the boot is off, the device will fail to extract. When this happens the device will generally fail in step 6.

With the Oppo R9S I intentionally did not follow instructions in the UFED which require *connecting the device to the UFED and waiting for the charging indicator to appear*. My method did create EDL Mode and the UFED was able to start the extraction process, but because I failed to follow instructions, the phone did not boot during the

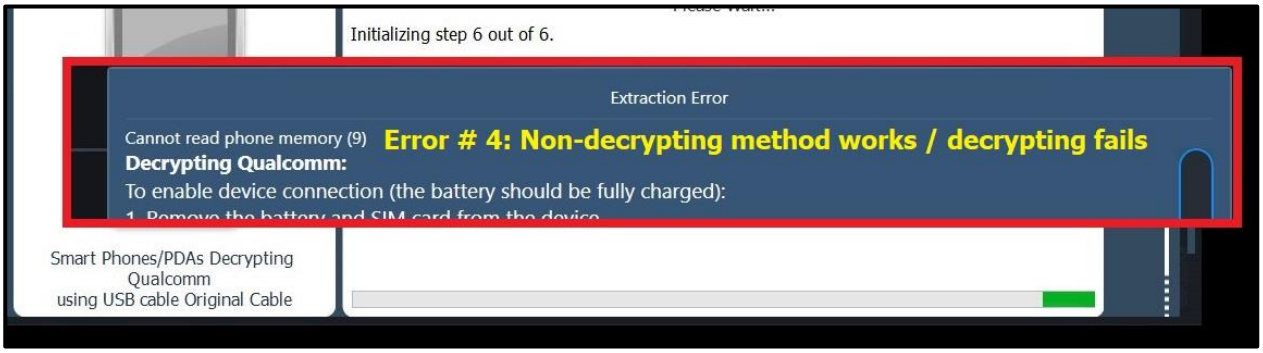


Figure 41 - Error # 4: Non-decrypting method works / decrypting fails

process. The phone restarted from EDL Mode but failed to boot when commanded by the UFED. Devices that fail after step four are supported for an EDL extraction by the UFED and they will likely extract via non-decrypting methods but that will only yield encrypted user data on devices like the Oppo R9S, which is encrypted. The R9S and some other devices require specific steps which must be followed in order for the decrypting bootloader to load properly and to avoid the issue of the charge-only mode. Connecting the device to the UFED, waiting for the charging indicator on the phone, and then creating EDL Mode without disconnecting will usually solve that issue.

5.5 Extractions of badly damaged, encrypted devices using the EDL exploit

I receive a significant number of devices that have been damaged or have been intentionally destroyed. One of the reasons I wanted to learn JTAG, ISP and Chip-Off was to gain access to damaged devices. In the past, those methods have allowed examiners to extract many devices which could never be started again. Those methods are still very useful and relevant today as I have tried to emphasize in this paper. With that said, we can’t deny that encryption has impeeded all aspects of digital foresics significantly, especially concerning devices that have been significantly damaged or destroyed.

At this moment, chipping a device or using ISP to extract data from an encrypted device is virtually useless. The physical process still works, but the user data is encrypted. There are companies working on a solution - that is the possibility of decrypting data after it has been extracted from the device that encrypted it. For now, I think we can safely use a broad generalization and say, it can't be done. I would like to have to eat those words in the future.

5.5.1 Using Cellebrite's EDL exploit on encrypted devices that don't function

One of the things I realized when I began experimenting with EDL was not just the decrypting aspect of Cellebrite's EDL exploit. That is exciting and impressive in itself but there are other types of extractions and lock bypasses that will yield decrypted data. One of the benefits of EDL extractions is that the device can be extracted via EDL without the need to interact with the device screen at all. Introducing eMMC faults and placing the device in EDL Mode can be done with no requirement to see or interact with a screen. From that point, the UFED can take control of the device and extract decrypted data.

With damaged devices, the USB port becomes one of the most important parts of the device. If the phone is broken in half and the USB port is beyond repair, can an encrypted device be extracted? We need the device to give us the data through the USB lines, but if the port is gone what can we do? Removing the storage is out due to encryption. The solution to the problem is really as simple as using skills learned from JTAG and ISP training. Not just the soldering, which is important. I am talking about pinning devices for USB data and power connections.

5.5.2 Damaged devices

Damaged and broken devices means you have to locate alternate locations on the device to tap into the USB data lines. When utilizing the decrypting EDL method in the UFED, the device also needs to boot. That means it may be necessary to duplicate power and communication normally done through USB and the battery. All of those things are possible and can be done with a multimeter, some soldering, pinning, and a lot of research and patience.

There are a lot of little details related to finding these locations and getting it all to work. I will include some links to photos, diagrams, and videos showing examples of sucessful EDL extractions of badly damaged devices and other types of extractions available in the UFED.

EDL extractions and other physical lock bypass exploits can allow us to extract physical images from badly damaged, encrypted devices. In that respect, EDL gives us options for encrypted devices which are similar to the use of ISP, JTAG, and Chip-Off with non-encrypted devices.

5.5.3 Warning about advanced procedures

When using techniques like this, I would highly suggest that test devices should be obtained first. The first time performing this procedure should not be on evidence. Make it work on a test device first – more than one if you can. Consult with Cellebrite's CAS or other professional services and make sure this is your only option. Then consider that this method may fail and you may destroy the device and your evidence.

5.5.4 Damaged Devices - Bypassing and Surgery Diagrams and Videos

WARNING! Consult with professional services or companies before attempting procedures that are experimental or require destruction or surgery on devices. These procedures should be viewed as a last resort when no other options are available.

Phone	Processor	Encrypted	USB Bypass	Decrypted EDL or other type of Extraction
ZTE Z799VL	MSM8909	Yes	Yes	Decrypted EDL
ZTE N9136	MSM8909	Yes	Yes	Decrypted EDL
Samsung S327VL	MSM8917	Yes	Yes	Physical Lock Bypass of Samsung devices running Qualcomm

				processors introduced in UFED 7.10 for MSM8917 / 8937 / 8953 / 8976
Samsung J3 SM-J327P USB Bypass Video	MSM8917	Yes	Yes & home button bypass	Physical Lock Bypass of Samsung devices running Qualcomm processors introduced in UFED 7.10 for MSM8917 / 8937 / 8953 / 8976
SM-J327P USB bypass diagram	MSM8917	Yes	Yes & home & Vol button bypass	Diagram showing bypass of home USB, home button, and volume buttons

6 Links to Documents and Information on EDL in this Paper

Item (Link)	Description
Practical Guide for Qualcomm EDL Physical Extractions	Shahar Tal's Introduction to EDL Extractions
Cellebrite EDL Webinar 2-21-18	PDF of Cellebrite's EDL Webinar on 2-21-18
Cellebrite EDL Webinar # 2 - 9-12-18	PowerPoint of Cellebrite's 2nd EDL Webinar 9-12-18 – videos included
EDL Non-decrypting Pulling Steps	Step-by-step guide for using the non-decrypting EDL method and testing any device for EDL – by Lorenz
EDL Decrypting Pulling Steps	Step-by-step guide for using the generic decrypting method for EDL extraction – by Lorenz
UFED Generic EDL Instructions	General Instructions given by the UFED on placing devices into EDL Mode on "waiting to extract" screen.
EDL Mode in Device Manager	What EDL looks like in Device Manager
Three Choices for Extraction	A description of each button's purpose on non-decrypting and decrypting EDL extractions – by Lorenz
Testing and Extracting with Cellebrite Cable # 523	Methods and steps in using Cellebrite's EDL cable #523 – by Lorenz
Non-Decrypting EDL Pulls Without Battery	A visual description of a phones condition when extracted via the non-decrypting EDL method – by Lorenz
EDL Mode Via ADB with UFED 4PC and Command Prompt	A step-by-step guide to using automated and manual methods of creating EDL via ADB – by Lorenz
See Moto Heat Shield Removal Video	Video showing how to remove tough heat shields without damaging the device – by Lorenz
Moto Heat Shield Removal Close Up	Video showing how to remove tough heat shields without damaging the device – microscope view by Lorenz
eMMC Chip Pinouts	Diagrams showing pinouts of 3 eMMC chips – by Lorenz
ZTE Z799VL - Which Points Create EDL	Diagram demonstrating all possible eMMC short locations – by Lorenz
Needle Short Method	eMMC Shorting a device into EDL via a needle and a wire with no soldering – by Lorenz
Clip Method	eMMC shorting devices into EDL via a wire and clip
Breadboard Method	eMMC shorting devices into EDL via breadboard method
eMMC VS UFS Chips	Diagram showing the different between eMMC and UFS chips
Galaxy S7 SM-G930A EDL Short Points	Diagram showing multiple EDL short points on the Galaxy S7
Soldering Short Points	Diagram showing clean soldering of eMMC points
XT-1650-2 Hidden Short Locations	Diagram showing hidden short locations for creating EDL
LG Test Points that Look Like a Cross	A sample LG pinout demonstrating common LG Test Points that look like a cross – by Lorenz
Kim Thomson – ISP Recognition	A paper discussing techniques to locate ISP pins without chipping a test phone – by Kim Thomson

MSM8909 PIN PATTERNS	ISP pin patterns based on devices running the MSM8909 processor – by Lorenz
MSM8916 PIN PATTERNS	ISP pin patterns based on devices running the MSM8916 processor – by Lorenz
Soldering to ISP Points using Clip Method	PowerPoint slides showing solder to ISP points to create EDL using the clip method – by Lorenz
EDL Procedure for H1611	A procedure for shorting the Huawei H1611 – by Lorenz
EDL Procedure for Z799VL / Z798BL / Z899VL	Diagram showing step-by-step instructions for extracting the Majesty Pro Series of phones using EDL
UFED LG EDL Removal tool – EDL "bricked"	Using the UFED LG EDL Removal tool to restore a device that is stuck in EDL Mode because of improper disconnect procedures (2:59)
ADB Console – Decrypting EDL	Step by step chart for decrypting EDL extractions using ADB Console
ADB Console – Non-Decrypting EDL	Step by step chart for non-decrypting EDL extractions using ADB Console
MSM8917 – BGA221 pinout and data flow	Diagram show all ISP pins for MSM8917 processor and BGA 221 chip
JTAG Processors	List of many of the processors supported for JTAG for RIFF and Octoplus
User Data Wipe – Encryption Test	Two ATT phones one encrypted and one not encrypted running Android 8

7 Links to Original How-to Videos – Narrated (These are longer videos – may need to download to avoid audio syncing issues)

DFU and FTM Extractions	Using button combos to place devices in EDL and FTM mode and extracting them (12:45)
EDL Cable Testing and Extraction	Testing for EDL cable support and extracting with the EDL cable (20:34)
eMMC Faults – Shorting	Shorting devices into EDL Mode using soldering and needlepoint method by creating eMMC Faults (35:02)
LG M150 Test Point Decrypting EDL	Using Test Points to create EDL on the LG M150 and extracting using decrypting bootloader (28:37)
Motorola XT1650-02 EDL short Points	(Not supported for EDL extraction as of 11-4-18) Locating the hidden points that create EDL Mode when shorted on the Motorola XT1650-02 (4:28)
Alcatel A574BL Raven AKA: 5044R idealXCITE/ 5044 Duo / 5044A / 5044O	Using needle method to reach the CLK pin under the heat shield to short these devices into EDL Mode (2:34)
UFED Touch - eMMC Faults (shorting) needle method and soldering Kyocera E6810	Demonstrating shorting using needle and soldering on Kyocera E6810 and EDL decrypting extraction with UFED Touch 2

8 2nd EDL Webinar Videos – 9-12-18

01_Z971 Cable #523 Extraction	2nd EDL Webinar – use of Cable #523 for EDL
02_Z839 DFU Mode Extraction	2nd EDL Webinar – DFU Mode procedure
03_N9136 FTM Extraction	2nd EDL Webinar – FTM mode procedures
04_MS330 Test Point Extraction	2nd EDL Webinar – LG Test points
05_Z719DL - Needle eMMC Fault Extraction	2nd EDL Webinar – needle used to short
06_Z852 eMMC Fault Extraction	2nd EDL Webinar - soldering

07_Coolpad 3636A Decrypting EDL - boot to charge-only	2nd EDL Webinar – Issue for applying decrypting bootloader to devices that boot to charge-only mode
08_Oppo R9S - Cable #523 Decrypting Extraction	2nd EDL Webinar - Issue for applying decrypting bootloader to devices that boot to charge-only mode

9 Android Debug Console Extractions

ADB Console – Decrypting EDL Chart	Step chart showing decrypting EDL steps using ADB Console
ADB Console – Non-decrypting EDL Chart	Step chart showing non-decrypting EDL steps using ADB Console
ADB Console – UFED Touch	UFED Touch extraction of N9136 using ADB Console (7:11)
ADB Console – EDL extraction via FTM	Using ADB Console for EDL extraction via FTM (3:34)
ADB Console – EDL extraction via cable 523	Using ADB Console for EDL extraction via cable # 523 (4:06)
ADB Console – Display device info on phone	Using ADB Console to read the device information on phone (2:07)

10 Example of devices sold with Android 6 or higher that were not encrypted by default

Devices Sold With Android 6 or Higher That Were Not Encrypted by Default						
Phone	**Processor**	**OS Version**	**RAM**	**Storage**	**Carrier**	**Specs**
Alcatel 5044R idealXCITE	MSM8909	7.0	2GB	16GB	ATT Prepaid	PS
Alcatel A577VL	MSM8909	7.1.1	2GB	Up to 16GB	Straight Talk	PS
ATT Axia QS5509A	MSM8909	8.1.0 (Go)	1GB	16GB	ATT Prepaid	Best Buy
Coolpad illumina 3310A	MSM8909	8.1.0 (Go)	1GB	8GB	Sprint	PS
Kyocera Cadence S2720PP	MSM8909	7.1.1	2GB	16GB	Verizon	PS
Motorola XT1609 Moto G4 Play	MSM8916	6.0.1	2GB	16GB	Verizon	PS
ZTE N9137 Tempo X	MSM8909	7.1.1	1GB	8GB	Virgin Mobile	PS
ZTE Z719DL Zmax One	MSM8909	7.1.1	2GB	16GB	Tracfone	Z719DL
ZTE Z835 Maven 3	MSM8909	7.1.1	1GB	8GB	ATT Prepaid	PS
ZTE Z839 Blade Vantage	MSM8909	7.1.1	2GB	16GB	Verizon	PS
ZTE Z852 Fanfare 3	MSM8909	7.1.1	1GB	8GB	Cricket	PS
ZTE Z899VL Majesty Pro Plus	MSM8909	7.1.1	2GB	16GB	Straight Talk	PS
ZTE Z963VL Max Duo	MSM8952	6.0	2GB	16GB	Tracfone	PS

11 Bibliography

Afonin, O., & Katalov, V. (2016). *Mobile Forensics - Advanced Investigative Strategies.* Birmingham: Packt Publishing.

Android. (2015, January 12). *Android Source.* Retrieved from Android: https://source.android.com/compatibility/5.0/android-5.0-cdd.html

Android. (2015, October 16). *Android Source.* Retrieved from Android: https://source.android.com/compatibility/6.0/android-6.0-cdd.pdf

Android. (n.d.). *Android Compatibility Document Definition.* Retrieved from Android: https://source.android.com/compatibility/cdd

Gragg, O. (2018, September 23). *Android turns 10: Remembering the first Android phone, the T-Mobile G1 / HTC Dream*. Retrieved from Android Authority: https://www.androidauthority.com/first-android-phone-t-mobile-g1-htc-dream-906362/

Kamdar, S. (2017, December 5th). *Introducing Android Oreo (Go edition) with the release of Android 8.1*. Retrieved from Android: The Keyword: https://www.blog.google/products/android/introducing-android-oreo-go-edition/

Figure 1 - Decrypting or Not - ZTE Z812 9
Figure 2 - User Data Encrypted Notification 12
Figure 3 - Secure Startup Screen 13
Figure 4 - Secure Startup choice 13
Figure 5 - ZTE Z835 UFED Support 15
Figure 6 - Samsung Convoy 4 JTAG Pinout 19
Figure 7 - JTAG Processors in RIFF and Octoplus 20
Figure 8 - APQ8064T JTAG & ISP Pinout 20
Figure 9 - BGA 221 with Qualcomm MSM8917 encryption 21
Figure 10 - Android Compatibility Definition Documents 23
Figure 11 - Device Encryption by date 24
Figure 12 - Dediprog Chip-Off Solution for UFS Storage 26
Figure 13 - AT&T Prepaid Phones Walmart 27
Figure 14 - Waiting for Device Screen 30
Figure 15 - EDL Mode in Device Manager 30
Figure 16 - DFU Mode in Device Manager 30
Figure 17 - Methods of creating EDL selection options 31
Figure 18 - UFED ADB Console - Device Unlocked & EDL Mode 33
Figure 19 - ADB Console for FTM & DFU Mode 34
Figure 20 - Using and Test the EDL Cable # 523 35
Figure 21 - Non-decrypting extraction with logic board only 35
Figure 22 - Anatomy of a USB Cable 36
Figure 23 - Non-decrypting Extractions without a battery 36
Figure 24 - FTM & DFU Mode in Device Manager 39
Figure 25 - Device in FTM 40
Figure 26 - UFED uses LG LAF to create EDL Mode 42
Figure 27 - UFED LG EDL recovery tool 44
Figure 28 - Xiaomi Mi 5X EDL Test Points 45
Figure 29 - Motorola motog6 Play XT1922-7 Test Points 45
Figure 30 - Motorola moto E5 Play XT1921-5 Test Points 45
Figure 31 - LGL51AL - K7 Test Points 46
Figure 32 - Motorola XT1650-02 Shorting 49
Figure 33 - Soldering to ISP locations to create EDL Mode 50

Figure 34 - eMMC Chip pinout 51
Figure 35 - ZTE Majesty Pro Series ISP Pinout 52
Figure 36 - Creating eMMC Faults via Soldering to ISP Points 53
Figure 37 - Galaxy S7 SM-G930A - EDL Short Locations 54
Figure 38 - eMMC vs. UFS - Pinouts 55
Figure 39 - Verify USB Debugging Error 58
Figure 40 - No Additional Bootloaders 60
Figure 41 - Error # 4: Non-decrypting method works / decrypting fails 61