



Encrypted updates

Contents

2	Threat model	3
3	Objectives	3
4	Properties	3
5	Threats	3
6	Mitigations	3
7	Risks and impacts	3
8	the private key for signing is leaked	3
9	the private key for signing is lost	4
10	the symmetric key for encryption/decryption is leaked	4
11	the symmetric key for encryption/decryption is lost	5
12	Key infrastructure	5
13	How keys can be stored on devices	5
14	How keys can be deployed to devices	5
15	When new keys should be generated	5
16	How the build pipeline can fetch the keys	6
17	How multiple keys can be used for key rotations	6
18	How to handle the leak of a key to the public and how that impacts	
19	future updates	6
20	Encryption Parameters	6
21	The encryption of the update file makes accessing its contents more difficult for	
22	bystanders, but doesn't necessarily protect from more resourceful attackers that	
23	can extract the decryption key from the user-owned device.	
24	The bundle encryption is done using the loop device with standard/proven kernel	
25	facilities for de/encryption (e.g. dm-crypt/LUKS). This allows the mechanism	
26	to be system agnostic (not tied to OSTree bundles), and can be used to ship up-	
27	dates to multiple components at once by including multiple files in the bundle.	
28	dm-crypt is the Linux kernel module which provides transparent encryption of	
29	block devices using the kernel crypto API, see dm-crypt ¹ .	
30	LUKS is the standard for Linux hard disk encryption. It provides secure man-	
31	agement of multiple user passwords, see LUKS wiki ² .	
32	The authenticity of the update is checked by verifying the OSTree signature as	
33	dm-crypt utilises symmetric cryptography which can't be used to ensure trust	
34	as the on-device key can be used to encrypt malicious files, not just decrypt	
35	them.	

¹<https://gitlab.com/cryptsetup/cryptsetup/-/wikis/DMCrypt>

²<https://gitlab.com/cryptsetup/cryptsetup/-/wikis/home>

36 Threat model

37 Objectives

- 38 1. end-users can download updates from the product website and apply them
- 39 offline to the device via a USB key or SD card
- 40 2. only official updates should be accepted by the device
- 41 3. the contents of the updates should not be easily extracted, increasing
- 42 the effort required for an attacker and providing some protection for the
- 43 business' intellectual property

44 Properties

- 45 1. **integrity**: the device should only accept updates which have not been
- 46 altered
- 47 2. **authenticity**: the device should only accept updates coming from the
- 48 producer
- 49 3. **confidentiality**: the contents of the updates should not be disclosed

50 Threats

- 51 1. Alice owns a device and wants to make it run her own software
- 52 2. Emily owns a device and Alice wants to have her own software on Emily's
- 53 device
- 54 3. Vincent develops a competing product and wants to gain insights into the
- 55 inner workings of the device

56 Mitigations

- 57 1. **integrity**: the update is checksummed, causing alteration to be detectable
- 58 2. **authenticity**: the update is signed with a private key by the vendor and
- 59 the device only accepts updates with a signature matching one of the
- 60 public keys in its trusted set
- 61 3. **confidentiality**: the update is encrypted with a symmetric key (due to
- 62 technology limitations public key decryption is not available)

63 Risks and impacts

64 the private key for signing is leaked

65 Impact

- 66 • the private key allows Alice to generate updates that can be accepted by
- 67 all devices

68 Mitigations

- 69 • the private key is only needed on the vendor infrastructure producing the
70 updates
- 71 • the chance of leaks is minimized by securing the infrastructure and ensur-
72 ing that access to the key is restricted as much as possible
- 73 • public keys for the leaked private keys should be revoked
- 74 • multiple public keys should be trusted on the device, so if one is revoked
75 updates can be rolled out using a different key
- 76 • keys should not be re-used across products to compartmentalize them
77 against leaks

78 **the private key for signing is lost**

79 **Impact**

- 80 • updates can't be generated if no private key matching the on-device public
81 ones is available

82 **Mitigations**

- 83 • if multiple public keys are trusted on the device, the private key used can
84 be rotated if another private key is still available
- 85 • backup private keys should be stored securely in different locations

86 **the symmetric key for encryption/decryption is leaked**

87 **Impact**

- 88 • Alice has access to all symmetric keys stored in bundles encrypted with
89 the leaked key
- 90 • the symmetric key allows Alice to generate updates that can be decrypted
91 by devices

92 **Mitigations**

- 93 • due to its symmetric nature, the secret key has to be available on both
94 the vendor infrastructure and on each device
- 95 • secure enclave technologies can help use the symmetric key for decryption
96 without exposing the key in any way
- 97 • if secure enclave is not available the key has to be stored on the device
98 and can be extracted via physical access
- 99 • if the key can't be provisioned in the factory the key has to be provisioned
100 via unencrypted updates, from which an attacker can extract the keys
101 without physical access to the device
- 102 • multiple decryption keys must be provisioned, to be able to rotate them
103 in case of leaks

104 **the symmetric key for encryption/decryption is lost**

105 **Impact**

- 106 • encrypted updates can't be generated for devices only using this symmetric
107 key

108 **Mitigations**

- 109 • given that the key has to be available on each device, the chance of losing
110 the encryption/decryption key is small
- 111 • if multiple decryption keys are provisioned on the device, the encryption
112 key can be rotated
- 113 • if all keys are lost or corrupted on the device, it will not be possible to
114 decrypt bundles on USB/SDCard and so to update the device using this
115 method.

116 **Key infrastructure**

117 LUKS is able to manage up to 8 key slots, any of the 8 different keys can be
118 used to decrypt the update bundle. This can allow a bundle to be read using a
119 main key or fallback key(s), and/or by different devices with a different subsets
120 of the used keys.

121 On the device itself, Apertis Update Manager is in charge of decrypting the
122 bundle and it will try as many keys as needed to unlock the bundle, there's no
123 limitation on the number of keys which can be stored.

124 Random keys for bundle encryption can be generated using:

```
125 head -c128 /dev/random | base64 --wrap=0
```

126 **How keys can be stored on devices**

- 127 • Keys can be stored in separated files, located in read-only part of the
128 filesystem: `/usr/share/apertis-update-manager/`
- 129 • In future versions, keys may be stored using the secure-boot-verified key
130 storage system

131 **How keys can be deployed to devices**

- 132 • Keys stored in the filesystem can be deployed by the normal update mech-
133 anism

134 **When new keys should be generated**

135 New keys should be generated:

- 136 • for new products

- when a key has been compromised

How the build pipeline can fetch the keys

- As for the signing key, the key(s) used to encrypt the static delta bundle should be passed to the encryption script GitLab CI/CD variable(s)

How multiple keys can be used for key rotations

- When the keys are stored on the filesystem, key rotation will not provide any benefit as the leak of one key implies the leak of the others
- When the keys will be stored using the secure-boot-verified key storage system, the encrypted updates will be generated with non-leaked keys and will remove the leaked keys while adding the new keys to the secure-boot-verified key storage system, so the number of available keys remain the same

How to handle the leak of a key to the public and how that impacts future updates

- If the keys are stored on the filesystem, the leak of one key implies the leak of the others
- If the keys are stored using the secure-boot-verified key storage system, the next update should be signed with a key that hasn't been leaked and the update should revoke the leaked key

Encryption Parameters

In a classical usage, the encryption is setup through a benchmark on the computer/board which will use it, allowing a good balance between password strength and unlocking time. This could end-up by encrypted file not usable due to out of memory error or slow unlocking time.

LUKS key strength is managed through 3 `cryptsetup` parameters: `--pbkdf-memory`, `--pbkdf-force-iterations` and `--pbkdf-parallel`.

`--pbkdf-parallel` configures the maximum number of threads used to unlock the encrypted file. This is automatically decreased on hardware devices that have only one of just a few cores.

As encrypted update file is created during image build on computer with more CPU power and memory, and that it is important to find a balance between password strength and usability, the `--pbkdf-memory` and `--pbkdf-force-iterations` should be forced to appropriate values for the target board.