

# What users should know about Full Disk Encryption based on LUKS

Andrea VISCONTI

Department of Computer Science  
Università degli Studi di Milano



- 1 Introduction
- 2 The key management process
- 3 Linux Unified Key Setup (LUKS)
- 4 Analysis of a LUKS implementation [1]
- 5 Conclusions

# Problem Description

Mobile devices, laptops, USB memory usually **store large amounts of sensitive information frequently unprotected.**

If such devices are lost or stolen, the risk of **unauthorized disclosure of confidential, sensitive, or classified information** is very high.

Also **operating systems** store temporary files and/or swap partitions on hard drive and several problems arise when these files contain sensitive data.

# A possible solution

A possible solution is to encrypt the whole hard disk: **Full Disk Encryption (FDE)**.

FDE solutions aim to provide data security, even in the event that an encrypted device is lost or stolen.

All information is **encrypted/decrypted on the fly**, automatically and transparently.

Without the encryption key, the data stored on the disk remains inaccessible to any users (regular or malicious).

# Main issue and possible solution

One of the main issues facing Full Disk Encryption solutions is the **password management**. Indeed, **the master key** used to encrypt the whole disk **is stored on it**.

A well-known solution to this problem, is **to adopt a two level key hierarchy** but sometimes it is not enough (e.g. two level key hierarchy adopted by Android 3-4.3).

We analyze

- the **key management process** used to derive the encryption key;
- how the **choice of specific hash functions** and **aggressive power management options** may affect the security of a FDE solution.

# The key management process

LUKS adopts a **two level key hierarchy**:

- A **strong master key generated by the system** is used to **encrypt/decrypt whole hard disk**;
- The **master key is unique**;
- Such a key is **split, encrypted with a secret user key and stored on the device** itself;
- Each user has their own secret key;
- **Several encrypted master keys are stored** on disk, one for each user.

A possible solution is to adopt a KDF [2, 3]. The KDF implemented is PBKDF2 [4].

# The key management process (2)

$$DK = \text{PBKDF2}(p, s, c, dkLen)$$

More precisely, the derived key is computed as follows:

$$DK = T_1 || T_2 || \dots || T_{\lceil dkLen/hLen \rceil}$$

Each single block  $T_i$  is computed as

$$T_i = U_1 \oplus U_2 \oplus \dots \oplus U_c$$

where

$$U_1 = \text{PRF}(p, s || i)$$

$$U_2 = \text{PRF}(p, U_1)$$

$$\vdots$$

$$U_c = \text{PRF}(p, U_{c-1})$$

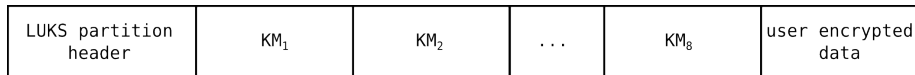
# The key management process (3)

In order to prevent building universal dictionaries, PBKDF2 uses a **salt**.

The **KDF** allows legitimate users to spend a moderate amount of time on key derivation, while **inserts CPU-intensive operations on the attacker side**.



# A LUKS partition



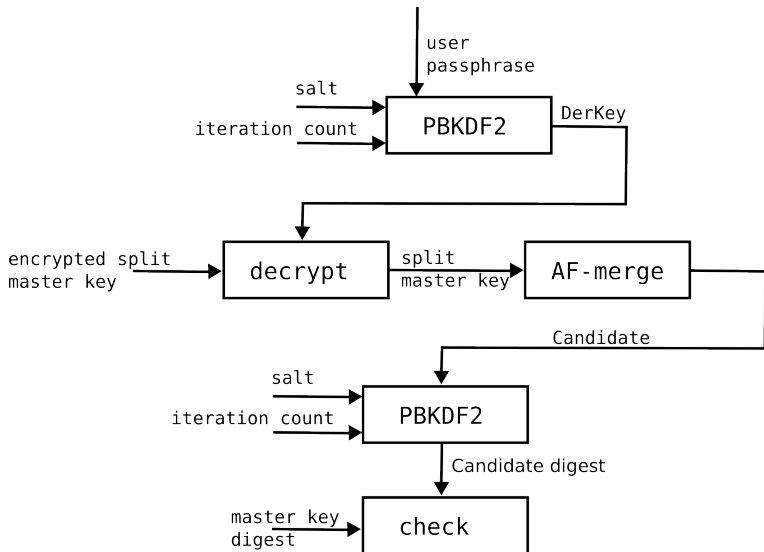
A **LUKS partition** contains information about

- start sector of key material;
- salt;
- iteration counts (e.g.  $c = 588,761$ );
- used cipher (e.g. AES);
- cipher mode (e.g. XTS);
- key length (e.g. 256 bits);
- hash function (e.g. SHA-1);
- master key checksum;
- ...

# A LUKS master key

When a user key is provided, it **unlocks one of the eight key slots** and the following algorithm is executed:

## LUKS: the master key recovery process



# Our testing activities...

Because **salt parameter is known** and user **password may be guessed**, we **focus on iteration counts** and their ability to slow down a brute force attack as much as possible.

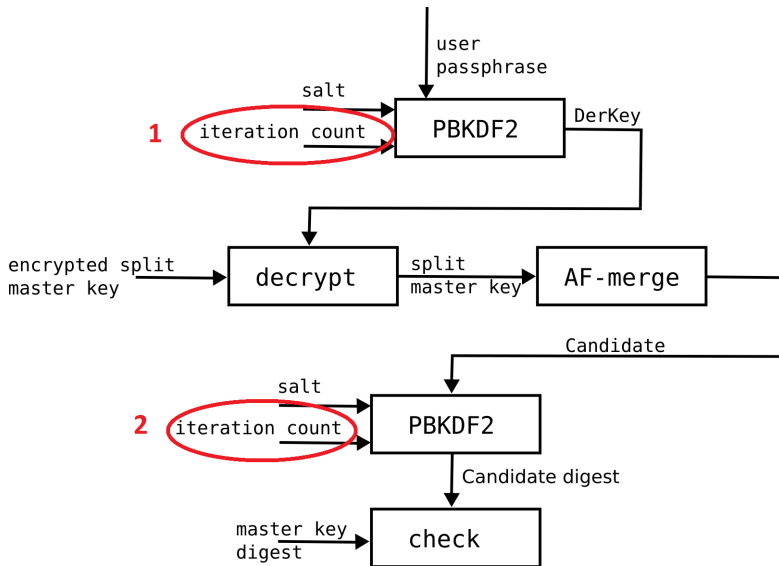
We try to understand

- 1 where and how the iteration counts are used;
- 2 how the choice of specific hash functions may affect the iteration count computation;
- 3 how unwitting users might significantly reduce the security of a LUKS implementation by setting aggressive power management options.

# Our testing activities...

1) Where and how the iteration counts are used...

## FIRST: Iteration counts (where and how)



## FIRST: Iteration counts (where and how) (2)

This table shows the average iteration count values involved in the **key derivation process**.

CPU	OS	sha1	sha512	sha256	ripemd160
Intel Atom z520	Debian 7.7 x86	31,035	7,019	18,567	29,491
Intel Core 2 Duo T6670	Kali 1.0 x86	151,772	22,821	67,634	111,791
Intel Pentium 3556U	Xubuntu 14.04 x64	126,617	50,082	77,379	103,287
Intel Core i3 2310M	Fedora 20 x64	136,375	50,107	77,682	111,536
Intel Pentium T4500	Ubuntu 12.04 x64	147,904	56,380	85,167	119,366
Intel Core i5 3320M	Debian 7.7 x64	232,203	88,843	139,985	196,209
Intel Core i7 2860QM	Kubuntu 14.04 x64	248,671	90,225	123,904	179,947
Intel Core i7 4710MQ	ArchLinux x64	588,761	302,148	392,916	350,378

## FIRST: Iteration counts (where and how) (3)

This table shows the average iteration count values involved in the **master key checksum process**.

CPU	OS	sha1	sha512	sha256	ripemd160
Intel Atom z520	Debian 7.7 x86	7,826	1,702	4,668	7,327
Intel Core 2 Duo T6670	Kali 1.0 x86	37,761	5,752	27,498	16,764
Intel Pentium 3556U	Xubuntu 14.04 x64	31,419	12,406	19,318	25,659
Intel Core i3 2310M	Fedora 20 x64	33,903	12,657	19,307	27,718
Intel Pentium T4500	Ubuntu 12.04 x64	36,913	14,009	21,495	29,951
Intel Core i5 3320M	Debian 7.7 x64	58,218	22,026	34,802	49,138
Intel Core i7 2860QM	Ubuntu 14.04 x64	54,371	19,353	30,926	44,927
Intel Core i7 4710MQ	ArchLinux x64	147,727	75,570	98,929	87,572

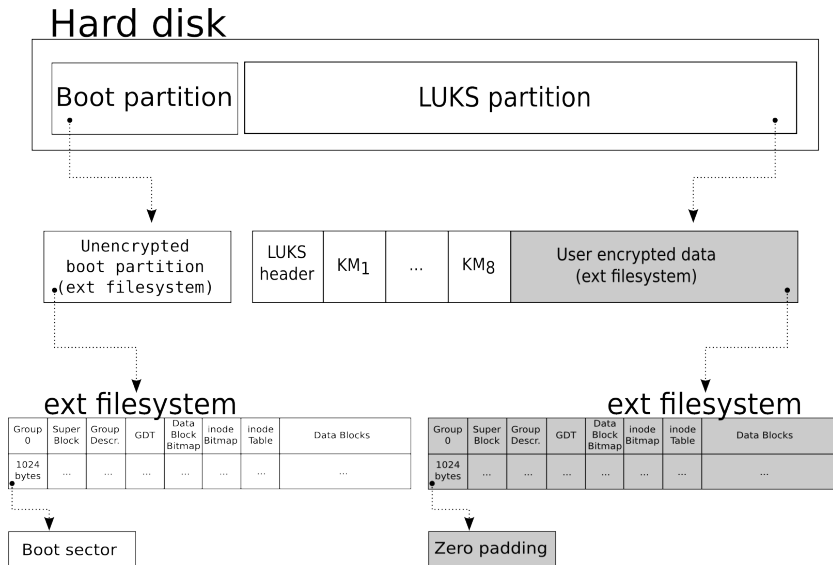


## FIRST: Iteration counts (where and how) (4)

**A closer look:** we experimentally observed that **about 75-80%** of the computational effort required to compute a derived key is generated by first iteration count, while the remaining **20-25%** by second one.

CPU	OS	sha1	sha512	sha256	ripemd160
Intel Atom z520	Debian 7.7 x86	31.035	7.019	18.567	29.491
Intel Atom z520	Debian 7.7 x86	7,826	1,702	4,668	7,327
Intel Core i3 2310M	Fedora 20 x64	136,375	50,107	77,682	111,536
Intel Core i3 2310M	Fedora 20 x64	33,903	12,657	19,307	27,718
Intel Core i7 4710MQ	ArchLinux x64	588,761	302,148	392,916	350,378
Intel Core i7 4710MQ	ArchLinux x64	147,727	75,570	98,929	87,572

## Avoiding the master key checksum process



# Our testing activities...

2) How the choice of specific hash functions may affect the iteration count computation...

## SECOND: Iteration counts and hash functions

The **iteration counts are automatically computed** by making some run-time tests when the encrypted partition is generated.

We **collected several partition headers** to be sure that such values are not conditioned by external factors, e.g. running programs.

More precisely, for each processor (eight) and each hash function (four), we **execute 100 runs for a total of  $8 \times 4 \times 100 = 3200$  partition headers collected**.

The variation across runs is observed to be less than 0.4%.

## SECOND: Iteration counts and hash functions (2)

As expected, devices with a **different hardware configuration** generate **different iteration count values**.

CPU	OS	sha1	sha512	sha256	ripemd160
Intel Atom z520	Debian 7.7 x86	31,035	7,019	18,567	29,491
Intel Core 2 Duo T6670	Kali 1.0 x86	151,772	22,821	67,634	111,791
Intel Pentium 3556U	Xubuntu 14.04 x64	126,617	50,082	77,379	103,287
Intel Core i3 2310M	Fedora 20 x64	136,375	50,107	77,682	111,536
Intel Pentium T4500	Ubuntu 12.04 x64	147,904	56,380	85,167	119,366
Intel Core i5 3320M	Debian 7.7 x64	232,203	88,843	139,985	196,209
Intel Core i7 2860QM	Kubuntu 14.04 x64	248,671	90,225	123,904	179,947
Intel Core i7 4710MQ	ArchLinux x64	588,761	302,148	392,916	350,378

Surprisingly, the **choice of a different hash function** may considerably **decrease the iteration count values**.

## SECOND: Iteration counts and hash functions (3)

The iteration counts related to SHA-256/512 are **considerably smaller** than those of SHA-1.

It is **easier to attack** a FDE solution which makes use of a **new hash function** (e.g., SHA256 or SHA512) rather than one which uses an **old hash function** (e.g., SHA-1).

**This is a counter-intuitive idea!**

From an user's point of view, SHA-256 and SHA-512 are still considered more secure than SHA-1, therefore a FDE solution based on SHA-2 is expected to be stronger.

## SECOND: Iteration counts and hash functions (4)

The **computational time** spent to compute a list of master key candidates does **not only depend on the iteration count values**.

Even the **number of fingerprints required** to compute a single iteration affects the total execution time!

## PBKDF2

$$DK = \text{PBKDF2}(p, s, c, dkLen)$$

More precisely, the derived key is computed as follows:

$$DK = T_1 || T_2 || \dots || T_{\lceil dkLen/hLen \rceil}$$

Each single block  $T_i$  is computed as

$$T_i = U_1 \oplus U_2 \oplus \dots \oplus U_c$$

where

$$U_1 = \text{PRF}(p, s || i)$$

$$U_2 = \text{PRF}(p, U_1)$$

$$\vdots$$

$$U_c = \text{PRF}(p, U_{c-1})$$



## SECOND: Iteration counts and hash functions

... but a SHA-1 fingerprint is only 160 bits in length and cannot be used as derived key, hence **a second fingerprint is necessary**:

$$DerKey = T_1 || T_2$$

On the other hand, **SHA-256 and SHA-512 generate enough bits** to compute a derived key:

$$DerKey = T_1$$

At equal iteration count values, we need to run the HMAC-SHA1 two times. Hence, a FDE solution based on HMAC-SHA1 slow down a brute force attack better than one based on HMAC-SHA256/SHA512.

# THIRD: Iteration counts and power management

3) How unwitting users might significantly reduce the security of a LUKS implementation by setting aggressive power management options...

## THIRD: Iteration counts and power management

Another **important feature** that users have to take into account during encryption operations are the **power management options**.

A common way to increase the battery life of devices is to enable **aggressive power saving policies**. Such policies save power, but they also **impact performance** by lowering CPU clock speed.

... and the **iteration count values fall down** even further!

CPU	OS	Max Freq (Plugged)	Min Freq (Unplugged)
Intel Atom z520	Debian 7.7 x86	1.33 GHz	0.80 GHz
Intel Pentium 3556U	Xubuntu 14.04 x64	1.70 GHz	0.80 GHz
Intel Core i7 4710MQ	ArchLinux x64	3.50 GHz	1.20 GHz

## THIRD: Iteration counts and power management

	SHA1		SHA512	
CPU	Plugged	Unplugged	Plugged	Unplugged
Intel Atom z520	31,035	18,693	7,019	4,288
Intel Pentium 3556U	126,617	62,969	50,082	25,161
Intel Core i7 4710MQ	588,761	202,143	302,148	104,216
	SHA256		RIPEMD	
CPU	Plugged	Unplugged	Plugged	Unplugged
Intel Atom z520	18,567	11,094	29,491	17,813
Intel Pentium 3556U	77,379	38,714	103,287	51,603
Intel Core i7 4710MQ	392,916	135,207	350,378	121,483

This means that power saving policies **might have an important impact** on the iteration count values, hence, on the strength of the FDE solution adopted.

# Testing: a toy example

We implemented a **brute-force attack** based on a password-list of 250,000 master keys (i.e. a dictionary attack).

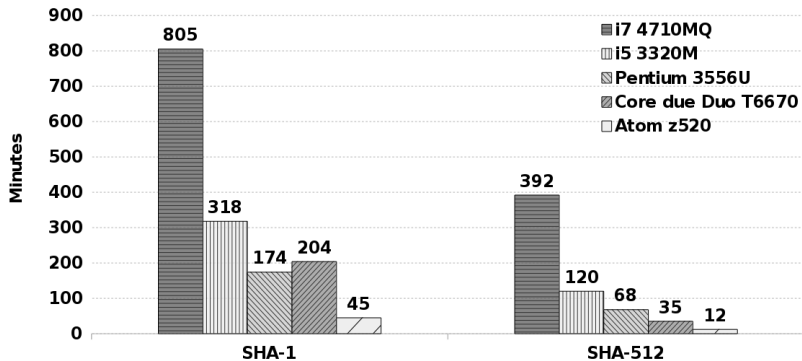
We run our code on a laptop equipped with an i7 4710MQ processor. No GPUs have been used.

We target two LUKS partitions collected using the following configuration options:

- default iteration count values, AES-256 XTS mode, **HMAC-SHA1**, laptop **plugged in**;
- default iteration count values, AES-256 XTS mode, **HMAC-SHA512**, laptop **unplugged**;

# Testing: a toy example (2)

The **second approach** (i.e., SHA-512 unplugged) **abruptly reduce the timeframe for brute forcing**, showing how the simple choice of configuration parameters may affect a FDE solution based on LUKS.



# Conclusions

We identify a number of issues that should be assessed and faced when a full disk encryption is implemented:

- The **iteration count values** are used to slow down a brute force attack, therefore, they **should not be too small**. Experimental results show that sometimes they are.
- The problem of EXT family file system allows attackers to substitute the master key checksum process by a simple decryption operation. The **CPU-intensive operations** used to compute a derived key **should not be avoided** by executing a set of functionally-equivalent instructions.
- From an user's point of view a FDE solution based on **HMAC-SHA256/SHA512**, is expected to be **much stronger than one based on SHA-1**, and be far more resistant to brute-force attacks. Our testing disprove this.

## Conclusions (2)

- Power management options should not affect the strength of a FDE solution. Testing results show that **aggressive power-saving approaches** may **have a relevant impact on** the iteration count values, hence, on **the strength of the solution adopted**.
- Master keys stored on disk are protected with **user keys which should have a minimum length requirement** in order to prevent a brute force attack. We experimentally observed that a number of distributions such as Debian, Ubuntu, and ArchLinux have no minimum length requirement, while Fedora has (but only eight characters).

...the cryptsetup 1.7 release changes defaults for LUKS...







Thanks for your attention!

`andrea.visconti@unimi.it`

`www.di.unimi.it/visconti`

`www.club.di.unimi.it`

# Bibliography

-  S. Bossi and A. Visconti, “What users should know about Full Disk Encryption based on LUKS,” in *Proceedings of the 14th International Conference on Cryptology and Network Security, CANS15*, 2015.
-  RSA Laboratories, “Pkcs #5 v2.1: Password based cryptography standard,” 2012.
-  NIST, “SP 800-132: Recommendation for password-based key derivation,” 2010.
-  A. Visconti, S. Bossi, H. Ragab, and A. Caló, “On the weaknesses of PBKDF2,” in *Proceedings of the 14th International Conference on Cryptology and Network Security, CANS15*, 2015.