Examining PBKDF2 security margin — case study of LUKS *

Andrea Visconti¹, Ondrej Mosnáček², Milan Brož², and Vashek Matyáš²

¹ Department of Computer Science, Università degli Studi di Milano
andrea.visconti@unimi.it,
WWW home page: http://www.di.unimi.it/visconti
² Faculty of Informatics, Masaryk University,
xmosnac@fi.muni.cz, xbroz@fi.muni.cz, matyas@fi.muni.cz

Abstract. Passwords are widely used to protect our sensitive information or to gain access to specific resources. They should be changed frequently and be strong enough to prevent well-known attacks. Unfortunately, user-chosen passwords are usually short and lack sufficient entropy. A possible solution to these problems is to adopt a Key Derivation Function (KDF) that allows legitimate users to spend a moderate amount of time on key derivation, while imposing CPU/memory-intensive operations on the attacker side. In this paper, we focus on long-term passwords secured by the Password-Based Key Derivation Function 2 (PBKDF2) and present the case study of Linux Unified Key Setup (LUKS), a disk-encryption specification commonly implemented in Linux based operating systems. In particular, we describe how LUKS protects long-term keys by means of iteration counts defined at runtime, and analyze how external factors may affect the iteration counts computation. In doing so, we provide means of evaluating the iteration count values defined at run-time and experimentally show to what level PBKDF2 is still capable of providing sufficient security margin for a LUKS implementation.

Keywords: Password-Based Key Derivation Function 2 (PBKDF2), Iteration counts, GPU, Linux Unified Key Setup (LUKS)

1 Introduction

User-chosen passwords are widely used to protect personal data or to gain access to specific resources. Therefore, passwords should be strong enough to prevent

* This paper extends and improves the previous work of the first author “What users should know about Full Disk Encryption based on LUKS” presented at the 14th International Conference on Cryptology and Network Security (CANS 2015). New experiments were conducted for this article (on 32-bit and 64-bit architectures), using different library versions and different cryptographic backends. In addition, a costing model is defined and used to estimate the cost of attacks on LUKS partition passwords created using Cryptsetup.
well-known attacks such as dictionary and brute-force attacks. User-chosen passwords are usually short and lack enough entropy \cite{43, 23} and cannot be directly used as a key for secure cryptographic systems. A solution to this problem is described in \cite{42}. By applying a Key Derivation Function (KDF) to a user-chosen password, we allow legitimate users to spend a moderate amount of time on key derivation, while we impose CPU-intensive operations on the attacker side.

The developments of the Graphic Processing Unit (GPU) architecture have changed this perspective. As opposed to CPUs, the GPU architecture consists of a large number of small processors (so-called streaming multiprocessors), which are capable of running a group of threads that execute the same program (and always the same instruction) over different data \cite{41}. In principle, this architecture is convenient for password cracking, because it involves computing the same function (a password-hashing function or a PBKDF) over a large number of passwords. However, since each GPU thread has only a small amount of fast memory available, the potential efficiency of the GPU implementation strongly depends on the design of the password processing function.

The first standardized PBKDF is PBKDF2 \cite{30, 45}, a CPU-intensive key strengthening algorithms. The design of PBKDF2 includes a time-based security parameter (the iteration count), but it has no parameter to increase the necessary memory usage of the function. PBKDF2 is defined generically in the sense that it does not strictly specify the pseudorandom function (PRF) to be used as its core, but in practice HMAC \cite{31} is almost solely used (usually with SHA-1, often also with SHA-256 or SHA-512). This means that PBKDF2 can be often implemented very efficiently on GPUs, thereby providing an attacker a huge potential speedup compared to the defender (who almost always runs PBKDF2 on a CPU).

As a reaction to the weakness of PBKDF2 against GPU-based attacks (and some other flaws), several memory-intensive key strengthening algorithms have been developed.

Bcrypt, for example, was introduced in 1999 \cite{40} and is used for password hashing in the OpenBSD operating system and PHP \cite{15}. It employs some protections against GPU/ASIC/FPGA attacks, but the memory usage is still fixed, so it is not completely safe for the future \cite{38}.

Scrypt was introduced in 2009 \cite{35} and allows tuning both time-based and memory-based security parameters. However, it has been criticised for allowing a so-called time-memory tradeoff, which makes it possible to achieve constant memory usage by performing more computation \cite{36}. It has also been recently (August 2016) standardized as RFC 7914 \cite{34}.

In 2013, an open competition called Password Hashing Competition was announced. The aim of the competition was to “identify new password hashing schemes in order to improve on the state-of-the-art” \cite{9}. In July 2015, the competition selected Argon2 as the winning algorithm \cite{20} and gives special recognition to Catena \cite{25}, Lyra2 \cite{44}, yescrypt \cite{37}, and Makwa \cite{39}.
1.1 Motivations and contributions

Although Argon2 is expected to become a new de-facto standard for password-based key derivation and password hashing, PBKDF2 is still widely implemented to derive keys in many security-related systems such as WPA/WPA2 encryption process [29], Linux Unified Key Setup (LUKS) [21,27], VeraCrypt [16], EncFS [2], FileVault Mac OS X [17,24], GRUB2 [3], Winrar [4], and many others.

In PKCS#5 [42], a number of recommendations for the implementation of PBKDF2 have been described. More precisely, PBKDF2 uses salt and iteration count to slow down the attackers as much as possible. The salt, randomly selected, is used to generate a large set of keys corresponding to a given password, while the iteration count specifies the number of times the underlying pseudorandom function is called to generate a block of keying material. Both salt and iteration count do not need to be kept secret. NIST suggests to select the iteration count as large as possible, as long as the time required to generate the key is acceptable for the user [45]. More precisely, SP 800-132 specifies that 10,000,000 may be an appropriate iteration count value for very critical keys on very powerful system, and 1,000 is a minimum recommended value.

In real-world applications, mainly two different approaches have been adopted in providing the iteration count:

1. define a value a priori — e.g., WPA/WPA2 encryption process [29], VeraCrypt [10];
2. define a value at runtime — e.g., LUKS disk encryption specification [21,27].

The first approach is widely used, yet it does not take into account hardware specifications of the devices on which PBKDF2 should run. This means that old devices and more powerful ones use the same iteration count value that is (usually) provided adopting a conservative approach — i.e., it favors performance at the expense of security. On the contrary, when applications care about security, the iteration count value is increased considerably — e.g., VeraCrypt [10] — impacting system performance and usability.

The second approach tries to resolve the cons previously described. It executes a runtime test on a device gathering information about the environment and, on the bases of hardware and software characteristics collected, provides an appropriate iteration count value — the details of the runtime test can be found in Section 4.1 or in LUKS disk encryption specification [26]. Unfortunately, this approach does not solve all problems. Indeed, the runtime testing may be negatively affected by several external factors such as the performance of different cryptographic backends, e.g., OpenSSL, Libgcrypt, the version of such cryptographic libraries, the particular architecture on which the code is running on, e.g., 32-bit/64-bit architecture, and so on.

In addition, we do not forget that (a) legitimate users and attackers run their algorithms on different hardware — i.e., regular users run their code on CPUs, while attackers may also run it on specialized hardware (ASIC/FPGA) or GPUs — and (b) some applications require long-term keys and therefore attackers can easily run their code off-line for a long time.
In such a scenario, we focus on the iteration count defined at runtime, presenting the case study of the Linux Unified Key Setup (LUKS), a disk-encryption specification commonly implemented in Linux based operating systems. In particular, we describe how LUKS protects long-term keys by means of an iteration count defined at runtime, and analyze how external factors may affect the iteration counts computation. In doing so, we provide means of evaluating the iteration count values defined at run-time and experimentally show to what level PBKDF2 is still capable of providing sufficient security margin for a LUKS implementation.

1.2 Organization of the paper

The remainder of the paper is organized as follows. In Section 2, we briefly introduce the Password-Based Key Derivation Function version 2. In Section 3, we introduce a method of estimating costs and duration of cracking a password protected by PBKDF2. In Section 4, we describe the Linux Unified Key Setup, a disk encryption specification based on PBKDF2, which computes the iteration count values by executing a runtime test. We experimentally show how external factors may affect the iteration count computation. In Section 5, we provide means of evaluating the iteration count values defined at run-time and experimentally show to what level PBKDF2 is still capable of providing sufficient security margin for a LUKS implementation. Finally, discussion and conclusions are drawn in Section 6.

2 Password-Based Key Derivation Function 2

PBKDF2 is a Password-Based Key Derivation Function described in PKCS #5 [42], [45]. For providing better resistance against brute force attacks, PBKDF2 introduce CPU-intensive operations. These operations are based on an iterated pseudorandom function (PRF) which maps input values to a derived key. The most important properties to assure is that the iterated pseudorandom function is cycle free. If this is not so, a malicious user can avoid the CPU-intensive operations and, as described in [46,47], get the derived key by executing a set of functionally-equivalent instructions.

PBKDF2 inputs a pseudorandom function PRF, the user password $p$, a random salt $s$, an iteration count $c$, and the desired length $len$ of the derived key. It outputs a derived key $DerKey$.

$$DerKey = PBKDF2(PR, p, s, c, len)$$

More precisely, the derived key is computed as follows:

$$DerKey = T_1||T_2||\ldots||T_{len},$$

where

$$T_1 = Function(p, s, c, 1),$$
Each single block $T_i$ — i.e., $T_i = \text{Function}(p, s, c, i)$ — is computed as

$$T_i = U_1 \oplus U_2 \oplus \ldots \oplus U_c,$$

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 PRF adopted can be a hash function [33], cipher, or HMAC [18], [19], and [31]. In this paper, we will refer to HMAC-SHA-1, HMAC-SHA-256, HMAC-SHA-512, and HMAC-RIPEMD-160.

### 3 Estimating the cost of attacking PBKDF2

Usually, the computational cost of cracking a password protected by PBKDF2 is estimated on the number of passwords per second that can be processed. In this section, we introduce a different approach of estimating such a cost. We will focus not only on the time required to recover a password, but also on the amount of computing resources needed to do so. In particular, the method suggested is based on two main sources of the cost: the cost of the power consumed by the hardware doing the cracking and the cost of the hardware itself. The cost is influenced by several factors — the PBKDF2 iteration count, the PRF used for PBKDF2, the size of the derived key/hash, the strength of the password and the performance characteristics of the hardware used for cracking. Since there might be many other contributing factors adding to the cost of an actual attack, our proposed estimations can only be seen as a lower bound for the actual cost.

#### 3.1 Attack scenario

The method we introduce assumes either brute-force or dictionary offline attack — i.e., that the attacker has access to the PBKDF2 hash of the searched password (along with the associated salt and iteration count), or some other information that allows her to verify that the output of PBKDF2 matches the expected one (e.g., she knows a plaintext-ciphertext pair encrypted with the output used as the key). The method further assumes that the time and computation power needed for verifying the PBKDF2 output of a given password candidate is negligible.

---

3 The sex of the attacker was set by a random coin toss.
3.2 Method of calculation

The estimation uses the following input variables:

- \( I \) \([\text{iter} \cdot \text{pw}^{-1}]\) – the number of PBKDF2 iterations,
- \( B \) \(\left\lceil \frac{\text{derived key size}}{\text{PRF output size}} \right\rceil\) – the number of PBKDF2 output blocks,
- \( S \) \([\text{pw}]\) – the password search space,
- \( T \) \([\text{s}]\) – the maximum acceptable (average) duration of the attack,
- \( E \) \(\left[\text{\$} \cdot \text{kWh}^{-1}\right]\) – the price of electricity available to the attacker\(^4\),
- \( V \) \([\text{iter} \cdot \text{s}^{-1} \cdot \text{dev}^{-1}]\) – the attacker’s device’s PBKDF2 computation speed (in terms of PBKDF2 iterations per second),
- \( P \) \([\text{W}]\) – the attacker’s device’s power draw while in full operation,
- \( D \) \([\text{\$}]\) – the purchase price of a single device,
- \( L \) \([\text{s}]\) – the average lifetime of a single device.

The estimated number of devices needed for the attack and its cost are calculated using the following formulas:

\[
\text{number of devices needed [dev]} = \frac{1}{2} IS B^2 V T \tag{4}
\]

\[
\text{average attack cost [\$]} = \frac{1}{2} IS B^2 \left( PE + \frac{D}{L} \right) \tag{5}
\]

3.3 Parameters for calculation

**Password search space:** This parameter \( S \) should express the assumed size of the password search space. This number depends on the way the password was selected and may vary for different scenarios. For example if the password is a random string of 8 characters, e.g., A-Z, a-z, and 0-9, the search space is \((26 + 26 + 10)^8 \approx 2^{48}\).

**Maximum acceptable attack duration:** When cracking a password, it is usually required that the attack succeeds within a reasonable amount of time (e.g., 1 month/1 year/5 years/...). Since a brute-force or dictionary attack can be trivially parallelized, it is sufficient to increase the number of devices that perform the cracking. Such an optimization does not (in theory) increase the total cost of computation, but there is certain practical limit on how many devices the attacker can use in parallel.

The value of this parameter \( T \) should reflect the upper bound on the attack duration and allows us to calculate the number of devices that would be needed to achieve such duration.

**Price of electricity:** Since we assume that the attack cost is mainly defined by the cost of consumed electricity, it is necessary to specify minimum expected electricity price for the potential attacker. The price of electricity varies a lot depending on the type of source and geographic location\(^4\). We suggest a conservative value of \( E = \$0.05 \text{kWh}^{-1} \) to be used for the calculations.

\(^4\) \$ = US dollar.
Device-specific constants: When estimating the general cost of an attack one needs to consider what kind of hardware the attacker will use for the attack. In general, it is best to assume the worst case – that the attacker will use the most cost-efficient solution available. In the case of PBKDF2, this is especially important, since computing PBKDF2 is significantly faster and cheaper on highly parallel hardware (such as GPUs or FPGAs) than on a regular CPU (see section 1). Therefore, it is important to base the cost estimation on the most efficient hardware, for which we have the PBKDF2 efficiency characteristics available.

PBKDF2 computation speed: This constant \( V \) is the experimentally measured speed of PBKDF2 computation on the device. It is expressed in terms of iterations per second per PRF output block.

Device power draw: This constant \( P \) represents the power draw of the device during the computation. The value can be measured or obtained from the device’s data sheet.

4 How to protect long-term keys: a case study of LUKS

4.1 A brief introduction to LUKS

The Linux Unified Key Setup (LUKS) is a disk-encryption specification commonly implemented in Linux based operating systems. It is a platform-independent standard on-disk format developed by Clemens Fruhwirth in 2004 [26, 27]. A LUKS partition (see Figure 1) includes:

1. the partition header,
2. the key material (i.e., a number of key slots), and
3. the user encrypted data.

![Fig. 1. A LUKS partition.](image)

Firstly, the partition header contains information about cipher, cipher mode, key length, hash function, master key checksum, salt, iteration counts, etc. [26]. Secondly, a number of key slots (maximum eight) are used to store the encrypted master key. More precisely, LUKS is based on a two-level key hierarchy
A strong master key generated by the system is used to encrypt/decrypt the whole hard disk. This key is encrypted with a secret user key. The master key is unique, but a number of encrypted keys are stored, one for each user who has access to the device. LUKS protects the keys stored on the hard disk using (1) a key derivation function (i.e., PBKDF2), and (2) an anti-forensic splitter to solve the remanence issues in magnetic storage devices. In particular, the anti-forensic splitter inflates and splits the master key, then a hash function is used as diffusion element. In order to recover the master key, we need a valid LUKS partition header and a secret user key. The user key unlocks a specific user key slot. Then, PBKDF2, the anti-forensic splitter, and a cipher are used to compute the master key as shown in Figure 2.

Finally, user encrypted data are stored on the device.

Among the various parameters stored in a LUKS partition header, the two most important are salt and iteration counts (see Figure 2), because they are used by PBKDF2 to slow down brute force attacks. The salt is fetched from a random source, while the iteration counts are computed by making a run-time test when the encrypted partition is generated. More precisely, the run-time test computes the iteration count value by running and evaluating a PBKDF2 implementation over one- or two-second window size — respectively before and after release 1.7.0 — with a predefined password, a specific salt and
a fixed key size. However, users can easily adjust this value as desired, defining
a time in seconds to benchmark the iteration counts. In accordance with NIST
specifications \cite{nist}, the iteration count computed cannot be smaller than 1,000.
Our analysis will focus on iteration count values computed in this way. In
particular, we try to understand how this parameter computed at runtime may
be affected by external factors.

4.2 LUKS: Iteration count values computed at runtime
In \cite{visconti2016,visconti2019} and \cite{fioravanti2019}, Visconti et al. showed how some weaknesses of PBKDF2 and
Cryptsetup affected the runtime testing used by LUKS to compute the iteration
count values. These weaknesses have been patched in Libgcrypt 1.7.0 \cite{libgcrypt} and
Cryptsetup 1.7.0 \cite{cryptsetup}, respectively. However, the runtime testing may be also
affected by regular processes that run on Linux distributions, the execution speed
of cryptographic algorithms implemented in different cryptographic backends
and also the choice of a specific architecture — i.e., 32-bit or 64-bit OS. The
influence of these external factors cannot be understood as a “weakness” and
cannot be patched. Therefore, we tried to identify which factors negatively affect
the iteration count computation.

In doing so, we experimentally collected several partition headers using two
laptops equipped with different hardware configurations: (1) a machine with an
Intel Core i5 5300U processor (with maximum frequency of 2.9 GHz, 2 cores
and 4 threads) and 16 GB of RAM; (2) a machine with an Intel Core i7 4500U
processor (with maximum frequency of 3.0 GHz, 2 cores and 4 threads) and 16
GB of RAM. In our testing activities, we used the following Linux distributions:

- Arch Nov 26, 2015,
- Debian (XFCE) 8.2.0,
- Fedora (GNOME) 23,
- Kali (GNOME) 2.0,
- Lubuntu (LXDE) 15.10,
- Mint (Cinnamon) 17.2,
- Ubuntu (Unity) 15.10.

All tests have been repeated for 32-bit and 64-bit architecture, using “Live CDs”.
To implement a uniform method of data collection, a common configuration
for all distributions has been adopted. In particular, we installed the following
libraries\footnote{Note that Libgcrypt 1.6.5 and OpenSSL 1.0.2g were the latest stable version available
at the time of testing.}: Libgcrypt 1.6.5, OpenSSL 1.0.2g, Cryptsetup 1.6.8 and Cryptsetup
1.7.0.

For each distribution (seven), each architecture (two) and each library (four)
listed above, we executed 100 runs for a total of $7 \times 2 \times 4 \times 100 = 5600$ iteration
counts collected. Average values are reported in Tables \ref{table1} and \ref{table3}. They refer
to Intel Core i5 5300U and Intel Core i7 4500U processor, respectively.
**Table 1.** Average iteration counts involved in the key derivation process (Libgcrypt 1.6.5, Live CD, Intel Core i5 5300U processor).

<table>
<thead>
<tr>
<th>Cryptsetup version 1.6.8</th>
<th>Cryptsetup version 1.7.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>32-bit OS</td>
<td>64-bit OS</td>
</tr>
<tr>
<td>Arch</td>
<td></td>
</tr>
<tr>
<td>SHA-1</td>
<td>381,013</td>
</tr>
<tr>
<td>SHA-256</td>
<td>183,376</td>
</tr>
<tr>
<td>SHA-512</td>
<td>64,834</td>
</tr>
<tr>
<td>RIPEMD-160</td>
<td>272,278</td>
</tr>
<tr>
<td>Debian</td>
<td></td>
</tr>
<tr>
<td>SHA-1</td>
<td>367,932</td>
</tr>
<tr>
<td>SHA-256</td>
<td>191,432</td>
</tr>
<tr>
<td>SHA-512</td>
<td>61,667</td>
</tr>
<tr>
<td>RIPEMD-160</td>
<td>254,239</td>
</tr>
<tr>
<td>Fedora</td>
<td></td>
</tr>
<tr>
<td>SHA-1</td>
<td>387,923</td>
</tr>
<tr>
<td>SHA-256</td>
<td>205,202</td>
</tr>
<tr>
<td>SHA-512</td>
<td>63,331</td>
</tr>
<tr>
<td>RIPEMD-160</td>
<td>274,983</td>
</tr>
<tr>
<td>Kali</td>
<td></td>
</tr>
<tr>
<td>SHA-1</td>
<td>387,236</td>
</tr>
<tr>
<td>SHA-256</td>
<td>203,523</td>
</tr>
<tr>
<td>SHA-512</td>
<td>56,562</td>
</tr>
<tr>
<td>RIPEMD-160</td>
<td>252,314</td>
</tr>
<tr>
<td>Lubuntu</td>
<td></td>
</tr>
<tr>
<td>SHA-1</td>
<td>385,344</td>
</tr>
<tr>
<td>SHA-256</td>
<td>205,652</td>
</tr>
<tr>
<td>SHA-512</td>
<td>65,259</td>
</tr>
<tr>
<td>RIPEMD-160</td>
<td>267,354</td>
</tr>
<tr>
<td>Mint</td>
<td></td>
</tr>
<tr>
<td>SHA-1</td>
<td>279,701</td>
</tr>
<tr>
<td>SHA-256</td>
<td>196,339</td>
</tr>
<tr>
<td>SHA-512</td>
<td>59,826</td>
</tr>
<tr>
<td>RIPEMD-160</td>
<td>258,148</td>
</tr>
<tr>
<td>Ubuntu</td>
<td></td>
</tr>
<tr>
<td>SHA-1</td>
<td>385,432</td>
</tr>
<tr>
<td>SHA-256</td>
<td>202,344</td>
</tr>
<tr>
<td>SHA-512</td>
<td>65,672</td>
</tr>
<tr>
<td>RIPEMD-160</td>
<td>260,230</td>
</tr>
</tbody>
</table>
Table 2. Average iteration counts involved in the key derivation process (OpenSSL 1.0.2g, Live CD, Intel Core i5 5300U processor).

<table>
<thead>
<tr>
<th>OS</th>
<th>Arch</th>
<th>SHA-1</th>
<th>SHA-256</th>
<th>SHA-512</th>
<th>RIPEMD-160</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>32-bit</td>
<td>64-bit</td>
<td>32-bit</td>
<td>64-bit</td>
<td></td>
</tr>
<tr>
<td></td>
<td>version 1.6.8</td>
<td>version 1.7.0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>OS</td>
<td>OS</td>
<td>OS</td>
<td>OS</td>
<td></td>
</tr>
<tr>
<td>32-bit</td>
<td>629,231</td>
<td>825,123</td>
<td>1,312,223</td>
<td>1,635,781</td>
<td></td>
</tr>
<tr>
<td>64-bit</td>
<td>468,335</td>
<td>616,545</td>
<td>2,007,356</td>
<td>2,455,634</td>
<td></td>
</tr>
<tr>
<td>32-bit</td>
<td>294,455</td>
<td>443,723</td>
<td>1,177,231</td>
<td>1,787,083</td>
<td></td>
</tr>
<tr>
<td>64-bit</td>
<td>389,345</td>
<td>504,433</td>
<td>792,845</td>
<td>941,175</td>
<td></td>
</tr>
<tr>
<td>Debian</td>
<td>SHA-1</td>
<td>570,418</td>
<td>782,920</td>
<td>1,131,172</td>
<td>1,561,416</td>
</tr>
<tr>
<td></td>
<td>SHA-256</td>
<td>364,159</td>
<td>481,299</td>
<td>1,444,241</td>
<td>1,891,303</td>
</tr>
<tr>
<td></td>
<td>SHA-512</td>
<td>233,972</td>
<td>377,063</td>
<td>929,403</td>
<td>1,479,630</td>
</tr>
<tr>
<td></td>
<td>RIPEMD-160</td>
<td>361,662</td>
<td>491,514</td>
<td>720,363</td>
<td>983,532</td>
</tr>
<tr>
<td>Fedora</td>
<td>SHA-1</td>
<td>537,051</td>
<td>803,023</td>
<td>1,064,544</td>
<td>1,577,923</td>
</tr>
<tr>
<td></td>
<td>SHA-256</td>
<td>423,119</td>
<td>593,211</td>
<td>1,667,642</td>
<td>2,362,423</td>
</tr>
<tr>
<td></td>
<td>SHA-512</td>
<td>261,398</td>
<td>439,068</td>
<td>1,049,324</td>
<td>1,753,742</td>
</tr>
<tr>
<td></td>
<td>RIPEMD-160</td>
<td>345,237</td>
<td>496,023</td>
<td>686,424</td>
<td>975,962</td>
</tr>
<tr>
<td>Kali</td>
<td>SHA-1</td>
<td>577,389</td>
<td>775,991</td>
<td>1,099,834</td>
<td>1,559,711</td>
</tr>
<tr>
<td></td>
<td>SHA-256</td>
<td>368,283</td>
<td>459,746</td>
<td>1,459,604</td>
<td>1,881,460</td>
</tr>
<tr>
<td></td>
<td>SHA-512</td>
<td>232,229</td>
<td>375,621</td>
<td>919,793</td>
<td>1,479,147</td>
</tr>
<tr>
<td></td>
<td>RIPEMD-160</td>
<td>365,082</td>
<td>491,319</td>
<td>725,479</td>
<td>786,642</td>
</tr>
<tr>
<td>Lubuntu</td>
<td>SHA-1</td>
<td>644,752</td>
<td>820,537</td>
<td>1,255,597</td>
<td>1,621,947</td>
</tr>
<tr>
<td></td>
<td>SHA-256</td>
<td>493,033</td>
<td>619,805</td>
<td>1,923,145</td>
<td>2,427,355</td>
</tr>
<tr>
<td></td>
<td>SHA-512</td>
<td>289,278</td>
<td>435,479</td>
<td>1,146,912</td>
<td>1,718,007</td>
</tr>
<tr>
<td></td>
<td>RIPEMD-160</td>
<td>385,815</td>
<td>487,371</td>
<td>771,541</td>
<td>1,007,931</td>
</tr>
<tr>
<td>Mint</td>
<td>SHA-1</td>
<td>627,625</td>
<td>788,036</td>
<td>1,224,068</td>
<td>1,564,635</td>
</tr>
<tr>
<td></td>
<td>SHA-256</td>
<td>391,481</td>
<td>482,962</td>
<td>1,532,511</td>
<td>1,898,248</td>
</tr>
<tr>
<td></td>
<td>SHA-512</td>
<td>243,739</td>
<td>385,121</td>
<td>961,830</td>
<td>1,540,997</td>
</tr>
<tr>
<td></td>
<td>RIPEMD-160</td>
<td>384,194</td>
<td>499,491</td>
<td>758,629</td>
<td>995,816</td>
</tr>
<tr>
<td>Ubuntu</td>
<td>SHA-1</td>
<td>645,242</td>
<td>824,821</td>
<td>1,293,239</td>
<td>1,610,281</td>
</tr>
<tr>
<td></td>
<td>SHA-256</td>
<td>496,901</td>
<td>604,167</td>
<td>1,939,750</td>
<td>2,427,615</td>
</tr>
<tr>
<td></td>
<td>SHA-512</td>
<td>286,276</td>
<td>439,493</td>
<td>1,154,148</td>
<td>1,740,286</td>
</tr>
<tr>
<td></td>
<td>RIPEMD-160</td>
<td>391,360</td>
<td>504,617</td>
<td>781,673</td>
<td>1,008,184</td>
</tr>
<tr>
<td></td>
<td>Cryptsetup version 1.6.8</td>
<td>Cryptsetup version 1.7.0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>----------</td>
<td>-------------------------</td>
<td>-------------------------</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>32-bit OS</td>
<td>64-bit OS</td>
<td>32-bit OS</td>
<td>64-bit OS</td>
<td></td>
</tr>
<tr>
<td>Arch</td>
<td>406,348</td>
<td>495,164</td>
<td>764,171</td>
<td>990,328</td>
<td></td>
</tr>
<tr>
<td>sha256</td>
<td>193,938</td>
<td>329,896</td>
<td>870,472</td>
<td>1,051,325</td>
<td></td>
</tr>
<tr>
<td>sha512</td>
<td>63,240</td>
<td>259,371</td>
<td>247,582</td>
<td>1,048,105</td>
<td></td>
</tr>
<tr>
<td>ripemd</td>
<td>241,508</td>
<td>304,761</td>
<td>573,347</td>
<td>609,523</td>
<td></td>
</tr>
<tr>
<td>Debian</td>
<td>359,549</td>
<td>516,128</td>
<td>715,082</td>
<td>1,036,435</td>
<td></td>
</tr>
<tr>
<td>sha256</td>
<td>192,770</td>
<td>336,841</td>
<td>766,468</td>
<td>1,354,496</td>
<td></td>
</tr>
<tr>
<td>sha512</td>
<td>58,823</td>
<td>265,164</td>
<td>233,576</td>
<td>1,053,436</td>
<td></td>
</tr>
<tr>
<td>ripemd</td>
<td>248,061</td>
<td>309,178</td>
<td>496,123</td>
<td>615,382</td>
<td></td>
</tr>
<tr>
<td>Fedora</td>
<td>337,285</td>
<td>518,217</td>
<td>677,248</td>
<td>1,027,080</td>
<td></td>
</tr>
<tr>
<td>sha256</td>
<td>203,173</td>
<td>345,012</td>
<td>813,990</td>
<td>1,381,916</td>
<td></td>
</tr>
<tr>
<td>sha512</td>
<td>62,135</td>
<td>260,958</td>
<td>233,150</td>
<td>1,057,849</td>
<td></td>
</tr>
<tr>
<td>ripemd</td>
<td>241,965</td>
<td>308,062</td>
<td>484,487</td>
<td>613,908</td>
<td></td>
</tr>
<tr>
<td>Kali</td>
<td>423,340</td>
<td>581,718</td>
<td>652,121</td>
<td>1,152,271</td>
<td></td>
</tr>
<tr>
<td>sha256</td>
<td>230,216</td>
<td>304,262</td>
<td>916,570</td>
<td>1,220,263</td>
<td></td>
</tr>
<tr>
<td>sha512</td>
<td>62,015</td>
<td>251,418</td>
<td>248,306</td>
<td>1,067,746</td>
<td></td>
</tr>
<tr>
<td>ripemd</td>
<td>215,060</td>
<td>421,052</td>
<td>646,927</td>
<td>836,201</td>
<td></td>
</tr>
<tr>
<td>Lubuntu</td>
<td>381,094</td>
<td>501,693</td>
<td>732,653</td>
<td>999,307</td>
<td></td>
</tr>
<tr>
<td>sha256</td>
<td>211,327</td>
<td>321,316</td>
<td>829,288</td>
<td>1,309,906</td>
<td></td>
</tr>
<tr>
<td>sha512</td>
<td>64,831</td>
<td>260,017</td>
<td>255,528</td>
<td>1,061,365</td>
<td></td>
</tr>
<tr>
<td>ripemd</td>
<td>279,481</td>
<td>306,241</td>
<td>551,369</td>
<td>608,947</td>
<td></td>
</tr>
<tr>
<td>Mint</td>
<td>362,468</td>
<td>488,548</td>
<td>735,630</td>
<td>977,097</td>
<td></td>
</tr>
<tr>
<td>sha256</td>
<td>203,821</td>
<td>329,896</td>
<td>805,031</td>
<td>1,319,585</td>
<td></td>
</tr>
<tr>
<td>sha512</td>
<td>57,552</td>
<td>264,461</td>
<td>254,978</td>
<td>1,049,179</td>
<td></td>
</tr>
<tr>
<td>ripemd</td>
<td>283,185</td>
<td>300,468</td>
<td>568,876</td>
<td>595,347</td>
<td></td>
</tr>
<tr>
<td>Ubuntu</td>
<td>367,815</td>
<td>484,847</td>
<td>670,156</td>
<td>977,097</td>
<td></td>
</tr>
<tr>
<td>sha256</td>
<td>193,938</td>
<td>326,530</td>
<td>815,285</td>
<td>1,312,620</td>
<td></td>
</tr>
<tr>
<td>sha512</td>
<td>63,744</td>
<td>262,294</td>
<td>254,978</td>
<td>1,057,849</td>
<td></td>
</tr>
<tr>
<td>ripemd</td>
<td>228,462</td>
<td>300,468</td>
<td>474,561</td>
<td>598,128</td>
<td></td>
</tr>
</tbody>
</table>
Table 4. Average iteration counts involved in the key derivation process (OpenSSL 1.0.2g, CD, Intel Core i7 4500U processor).

<table>
<thead>
<tr>
<th>Cryptsetup version 1.6.8</th>
<th>Cryptsetup version 1.7.0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>32-bit OS</td>
</tr>
<tr>
<td></td>
<td>32 bits</td>
</tr>
<tr>
<td>Arch Linux</td>
<td></td>
</tr>
<tr>
<td>sha1</td>
<td>619,854</td>
</tr>
<tr>
<td>sha256</td>
<td>483,018</td>
</tr>
<tr>
<td>sha512</td>
<td>281,318</td>
</tr>
<tr>
<td>ripemd</td>
<td>389,649</td>
</tr>
<tr>
<td>Debian</td>
<td></td>
</tr>
<tr>
<td>sha1</td>
<td>528,924</td>
</tr>
<tr>
<td>sha256</td>
<td>342,245</td>
</tr>
<tr>
<td>sha512</td>
<td>217,686</td>
</tr>
<tr>
<td>ripemd</td>
<td>338,624</td>
</tr>
<tr>
<td>Fedora</td>
<td></td>
</tr>
<tr>
<td>sha1</td>
<td>493,255</td>
</tr>
<tr>
<td>sha256</td>
<td>414,239</td>
</tr>
<tr>
<td>sha512</td>
<td>257,544</td>
</tr>
<tr>
<td>ripemd</td>
<td>345,478</td>
</tr>
<tr>
<td>Kali</td>
<td></td>
</tr>
<tr>
<td>sha1</td>
<td>656,410</td>
</tr>
<tr>
<td>sha256</td>
<td>492,307</td>
</tr>
<tr>
<td>sha512</td>
<td>283,185</td>
</tr>
<tr>
<td>ripemd</td>
<td>395,061</td>
</tr>
<tr>
<td>Lubuntu</td>
<td></td>
</tr>
<tr>
<td>sha1</td>
<td>671,207</td>
</tr>
<tr>
<td>sha256</td>
<td>511,963</td>
</tr>
<tr>
<td>sha512</td>
<td>299,395</td>
</tr>
<tr>
<td>ripemd</td>
<td>401,935</td>
</tr>
<tr>
<td>Mint</td>
<td></td>
</tr>
<tr>
<td>sha1</td>
<td>589,152</td>
</tr>
<tr>
<td>sha256</td>
<td>369,529</td>
</tr>
<tr>
<td>sha512</td>
<td>217,637</td>
</tr>
<tr>
<td>ripemd</td>
<td>313,863</td>
</tr>
<tr>
<td>Ubuntu</td>
<td></td>
</tr>
<tr>
<td>sha1</td>
<td>670,156</td>
</tr>
<tr>
<td>sha256</td>
<td>507,935</td>
</tr>
<tr>
<td>sha512</td>
<td>296,295</td>
</tr>
<tr>
<td>ripemd</td>
<td>390,243</td>
</tr>
</tbody>
</table>

To understand if our testing configuration — i.e., “Live CDs” — may have affected the runtime testing, we also installed some of the Linux distributions on the first laptop (Intel Core i5 5300U processor). Then we collected 1200 iteration count values with the distributions installed (see Tables 5 and 6) and compared these values with those shown in Tables 1 and 2. Note that the difference between the configurations called “OSs installed” and “Live CDs” is negligible.

*For each distribution (three), each architecture (one) and each library (four), we executed 100 runs for a total of $3 \times 1 \times 4 \times 100 = 1200$ iteration counts collected. Average values are reported in Tables 5 and 6.*
Table 5. Average iteration counts involved in the key derivation process (Libgcrypt 1.6.5, OSs installed).

<table>
<thead>
<tr>
<th>Arch</th>
<th>SHA-1</th>
<th>480,019</th>
<th>959,102</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SHA-256</td>
<td>324,114</td>
<td>1,290,239</td>
</tr>
<tr>
<td></td>
<td>SHA-512</td>
<td>266,921</td>
<td>1,069,921</td>
</tr>
<tr>
<td></td>
<td>RIPEMD-160</td>
<td>294,801</td>
<td>585,291</td>
</tr>
<tr>
<td>Debian</td>
<td>SHA-1</td>
<td>518,882</td>
<td>1,032,923</td>
</tr>
<tr>
<td></td>
<td>SHA-256</td>
<td>335,012</td>
<td>1,301,522</td>
</tr>
<tr>
<td></td>
<td>SHA-512</td>
<td>272,611</td>
<td>1,079,185</td>
</tr>
<tr>
<td></td>
<td>RIPEMD-160</td>
<td>303,092</td>
<td>603,801</td>
</tr>
<tr>
<td>Fedora</td>
<td>SHA-1</td>
<td>513,721</td>
<td>1,020,105</td>
</tr>
<tr>
<td></td>
<td>SHA-256</td>
<td>337,001</td>
<td>1,320,007</td>
</tr>
<tr>
<td></td>
<td>SHA-512</td>
<td>268,092</td>
<td>1,055,801</td>
</tr>
<tr>
<td></td>
<td>RIPEMD-160</td>
<td>302,102</td>
<td>599,987</td>
</tr>
</tbody>
</table>

Table 6. Average iteration counts involved in the key derivation process (OpenSSL 1.0.2g, OSs installed).

<table>
<thead>
<tr>
<th>Arch</th>
<th>SHA-1</th>
<th>820,723</th>
<th>1,537,322</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SHA-256</td>
<td>613,023</td>
<td>2,467,901</td>
</tr>
<tr>
<td></td>
<td>SHA-512</td>
<td>438,491</td>
<td>1,765,118</td>
</tr>
<tr>
<td></td>
<td>RIPEMD-160</td>
<td>507,002</td>
<td>976,992</td>
</tr>
<tr>
<td>Debian</td>
<td>SHA-1</td>
<td>768,173</td>
<td>1,535,254</td>
</tr>
<tr>
<td></td>
<td>SHA-256</td>
<td>487,468</td>
<td>1,877,618</td>
</tr>
<tr>
<td></td>
<td>SHA-512</td>
<td>370,287</td>
<td>1,459,036</td>
</tr>
<tr>
<td></td>
<td>RIPEMD-160</td>
<td>491,227</td>
<td>976,604</td>
</tr>
<tr>
<td>Fedora</td>
<td>SHA-1</td>
<td>701,334</td>
<td>1,568,833</td>
</tr>
<tr>
<td></td>
<td>SHA-256</td>
<td>605,046</td>
<td>2,340,005</td>
</tr>
<tr>
<td></td>
<td>SHA-512</td>
<td>434,133</td>
<td>1,726,227</td>
</tr>
<tr>
<td></td>
<td>RIPEMD-160</td>
<td>484,943</td>
<td>984,109</td>
</tr>
</tbody>
</table>

4.3 How external factors may affect the iteration count values

Tables 1 and 2 can help us to identify which factors negatively affect the iteration count computation. In particular, we look closely at library versions, architectures (32/64-bit OS), backends and distributions.

Library versions: The first external factor that we analyzed is the version of the libraries installed. Such a factor may affect the iteration count computation and consequently the attack cost. As an example, we show the differences from
Cryptsetup version 1.7.0 to 1.6.8 (see Figure 3). The value indicated in the graph has been obtained by dividing the iteration count for version 1.7.0 by the corresponding iteration count for version 1.6.8. These ratios are compared across different configurations. Note that the ratio between two costs is equivalent to the ratio between the corresponding iteration counts, as long as the hash functions and GPU devices used in Equation 5 are the same.

Fig. 3. Difference in attack cost in Cryptsetup 1.7.0 vs 1.6.8.

Disregarding some minor measurement variations, the attack cost is increased by a factor of 2 for SHA-1 and RIPEMD-160 and by a factor of 4 for SHA-256 and SHA-512. This increase is due to two independent bugs in the iteration count calculation that have been fixed in Cryptsetup version 1.7.0.

Interestingly, in Live CD distributions we found old library versions (see Table 7) such as Cryptsetup 1.6.1 (March 2013), Libgcrypt 1.6.1 (January 2014) and OpenSSL 1.0.1f (January 2014), while the latest version available at the time of writing were Cryptsetup 1.7.2 (June 2016), Libgcrypt 1.7.0 (April 2016) and OpenSSL 1.0.2h (May 2016).

Table 7. Library versions available in Live CDs (December 2015).

<table>
<thead>
<tr>
<th>Cryptsetup</th>
<th>Libgcrypt</th>
<th>OpenSSL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arch</td>
<td>1.7.0</td>
<td>1.6.4</td>
</tr>
<tr>
<td>Debian</td>
<td>1.6.6</td>
<td>1.6.3</td>
</tr>
<tr>
<td>Fedora</td>
<td>1.6.8</td>
<td>1.6.4</td>
</tr>
<tr>
<td>Kali</td>
<td>1.6.6</td>
<td>1.6.3</td>
</tr>
<tr>
<td>Lubuntu</td>
<td>1.6.6</td>
<td>1.6.3</td>
</tr>
<tr>
<td>Mint</td>
<td>1.6.1</td>
<td>1.6.1</td>
</tr>
<tr>
<td>Ubuntu</td>
<td>1.6.6</td>
<td>1.6.3</td>
</tr>
</tbody>
</table>

Unfortunately, also updated library versions available in official repositories are far from being up-to-date (see Table 8). Most of them are affected by the weaknesses described in [46, 21]. Only Arch supports libraries that are not.
Table 8. Updated library versions available in official repositories (May 2016).

<table>
<thead>
<tr>
<th>Arch</th>
<th>Cryptsetup</th>
<th>Libgcrypt</th>
<th>OpenSSL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Debian</td>
<td>1.6.6</td>
<td>1.6.3</td>
<td>1.0.1k</td>
</tr>
<tr>
<td>Fedora</td>
<td>1.6.8</td>
<td>1.6.4</td>
<td>1.0.2h</td>
</tr>
<tr>
<td>Kali</td>
<td>1.6.6</td>
<td>1.6.3</td>
<td>1.0.1k</td>
</tr>
<tr>
<td>Lubuntu</td>
<td>1.6.6</td>
<td>1.6.3</td>
<td>1.0.2d</td>
</tr>
<tr>
<td>Mint</td>
<td>1.6.1</td>
<td>1.6.1</td>
<td>1.0.1f</td>
</tr>
<tr>
<td>Ubuntu</td>
<td>1.6.6</td>
<td>1.6.3</td>
<td>1.0.2d</td>
</tr>
</tbody>
</table>

Architectures (32/64-bit OS): A second external factor that may affect the iteration count computation is the architecture on which our code is running. Experimental results reported in Tables 1 and 2 show an important gap between the iteration count values computed using several configurations. As an example, we list the following:

- “Table 1, Ubuntu, SHA-512, 32-bit OS, Cryptsetup 1.7.0” versus “Table 1, Ubuntu, SHA-512, 64-bit OS, Cryptsetup 1.7.0”. The iteration counts computed are 275, 296 (32-bit OS) and 1,197, 264 (64-bit OS). They quadrupled.
- “Table 1, Kali, SHA-512, Cryptsetup 1.6.8” (56, 562 vs 273, 236, 32-bit and 64-bit OS respectively). Again, they quadrupled;
- “Table 1, Fedora, SHA-512, Cryptsetup 1.7.0” (253, 724 vs 1,089, 109, 32-bit and 64-bit OS respectively). Again;
- “Table 2, Fedora, SHA-512, Cryptsetup 1.6.8” (261, 398 vs 439, 068, 32-bit and 64-bit OS);
- “Table 2, Mint, SHA-512, Cryptsetup 1.7.0” (961, 830 vs 1,540, 997, 32-bit and 64-bit OS);
- ...

This means that, from security perspective, it is better for the user to run the code on a 64-bit OS rather than a 32-bit OS, since we can get higher iteration counts (and thus higher costs for the attacker) for the same level of user inconvenience (i.e., time for unlocking the disk).

Backends: The third factor that we analyzed is the backend installed on the system, i.e., Libgcrypt vs. OpenSSL. Interesting results can be observed when analyzing the following configurations:

- “Table 1, Kali, SHA-512, 32-bit OS, Cryptsetup 1.6.8” versus “Table 2, Kali, SHA-512, 32-bit OS, Cryptsetup 1.6.8”. The iteration counts collected are 56, 562 (Libgcrypt) and 232, 229 (OpenSSL). They quadrupled.
- “Lubuntu, SHA-512, 32-bit OS, Cryptsetup 1.7.0” (261, 396 vs 1,146, 912, Table 1 and Table 2 respectively). Again, they quadrupled;
- “Fedora, SHA-512, 32-bit OS, Cryptsetup 1.6.8” (63, 331 vs 261, 398, Table 1 and Table 2 respectively). Again;
More in general, on 64-bit systems, Cryptsetup with OpenSSL used as the crypto backend produced between 1.5 and 2 times higher iteration counts as opposed to Cryptsetup with Libgcrypt. On 32-bit systems the following differences have been observed:

- for RIPEMD-160: 1.2-1.5 higher with OpenSSL vs. Libgcrypt,
- for SHA-1: 1.4-1.7 higher with OpenSSL vs. Libgcrypt (and 2.25 higher on Linux Mint),
- for SHA-256: 1.8-2.5 higher with OpenSSL vs. Libgcrypt,
- for SHA-512: 3.8-4.5 higher with OpenSSL vs. Libgcrypt.

Note that part of this increase is due to a performance bug \[46\] in PBKDF2 provided by Libgcrypt and fixed in Libgcrypt version 1.7.0 \[48\]. Figure 4 helps us to visualize these differences. The ratios in this graph have been obtained analogically to that of Figure 3 — i.e., by dividing the iteration count computed with OpenSSL by the corresponding Libgcrypt. Then, such ratios are compared across a number of configurations.

The graph suggests that the choice of cryptographic library to be used as the backend for Cryptsetup has a significant impact on the resulting iteration counts and, consequently, the attack cost.

**Linux Distributions:** The last factor which may affect the iteration count computation is the distribution installed. In particular, we analyzed data gathered with a number of Linux desktop distributions and we found the following result:

- In “Table 1, SHA-1, 32-bit OS, Cryptsetup 1.6.8”, the iteration counts gathered are 279, 701 (Mint) and 387, 923 (Fedora);
In cases listed above, the iteration count values are increased about 36-40 percent. Interestingly, the iteration count on the 32-bit Linux Mint with the Libgcrypt crypto backend is unusually low compared to other distributions. We were not able to identify the reason behind this deviation.

5 LUKS: Examining PBKDF2 security margin

In this Section we evaluate the estimated cost of attack on the password of a LUKS partition created using Cryptsetup. We assumed an optimized version of the attack where only key derivation PBKDF2 computation is needed. In general case the attack computation would have to include also keyslot decryption and master key checksum computation, which would increase the computation time by about 20-25% (see [21, section 4.1]).

To model an attacker’s hardware, we used PBKDF2 benchmarks produced by an open-source tool [32] using the experimentally collected iteration counts listed in Section 4.2. To estimate the cost of cracking a password protected by PBKDF2, we chose specific system parameters — e.g., the derived key length, the electricity price, the purchase price of a single device, and so on — and imposed limitations to some of them — e.g., a maximum acceptable attack duration. Finally, we ran our code using different GPUs such as NVIDIA GeForce 465, NVIDIA Tesla M2090, NVIDIA Tesla K20, and NVIDIA Tesla K20X. More details about system parameters and GPUs can be found in Tables 9, 10, and 11.

Table 9. Attack estimation parameters.

<table>
<thead>
<tr>
<th>Parameter name</th>
<th>Value</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Derived key size</td>
<td>256 bits</td>
<td></td>
</tr>
<tr>
<td>Password search space ( (S) )</td>
<td>( 2^{48} )</td>
<td>(8-character alphanumeric)</td>
</tr>
<tr>
<td>Max. attack duration ( (T) )</td>
<td>5 years</td>
<td></td>
</tr>
<tr>
<td>Price of electricity ( (E) )</td>
<td>$0.05 kWh(^{-1})</td>
<td>[1]</td>
</tr>
</tbody>
</table>

5.1 Attack feasibility evaluation

We select two configurations of the user’s system (see Table 12) between those listed in Tables 1 and 2. More precisely, we chose the configurations that produce the worst and best iteration count value.

Using Eq. (4) and multiplying this result by the cost of the respective device, we get an idea of the cost of the whole equipment needed to perform an attack.
## Table 10. Attack estimation parameters for GPU devices.

<table>
<thead>
<tr>
<th>Device</th>
<th>Parameter name</th>
<th>Value</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>NVIDIA</td>
<td>Power draw ($P$)</td>
<td>200 W</td>
<td>[11]</td>
</tr>
<tr>
<td>GTX 465</td>
<td>Lifetime ($L$)</td>
<td>5 years</td>
<td>(conservative estimate)</td>
</tr>
<tr>
<td>NVIDIA</td>
<td>Power draw ($P$)</td>
<td>250 W</td>
<td>[14]</td>
</tr>
<tr>
<td>Tesla</td>
<td>Purchase price ($D$)</td>
<td>$2500</td>
<td>[14] (MSRP)</td>
</tr>
<tr>
<td>M2090</td>
<td>Lifetime ($L$)</td>
<td>10 years</td>
<td>(conservative est., based on [6])</td>
</tr>
<tr>
<td>NVIDIA</td>
<td>Power draw ($P$)</td>
<td>225 W</td>
<td>[12]</td>
</tr>
<tr>
<td>Tesla</td>
<td>Purchase price ($D$)</td>
<td>$3150</td>
<td>[12] (MSRP)</td>
</tr>
<tr>
<td>K20</td>
<td>Lifetime ($L$)</td>
<td>15 years</td>
<td>[6]</td>
</tr>
<tr>
<td>NVIDIA</td>
<td>Power draw ($P$)</td>
<td>235 W</td>
<td>[13]</td>
</tr>
<tr>
<td>Tesla</td>
<td>Purchase price ($D$)</td>
<td>$7700</td>
<td>[13] (MSRP)</td>
</tr>
<tr>
<td>K20X</td>
<td>Lifetime ($L$)</td>
<td>15 years</td>
<td>[7]</td>
</tr>
</tbody>
</table>

## Table 11. Measured PBKDF2 computation speeds for GPU devices

<table>
<thead>
<tr>
<th>Device</th>
<th>Hash function</th>
<th>PBKDF2 computation speed ($V$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>NVIDIA</td>
<td>SHA-1</td>
<td>178437745.7</td>
</tr>
<tr>
<td>GeForce</td>
<td>RIPEMD-160</td>
<td>138319407.9</td>
</tr>
<tr>
<td>GTX 465</td>
<td>SHA-256</td>
<td>75482616.7</td>
</tr>
<tr>
<td></td>
<td>SHA-512</td>
<td>24101794.6</td>
</tr>
<tr>
<td>NVIDIA</td>
<td>SHA-1</td>
<td>291032649.1</td>
</tr>
<tr>
<td>Tesla</td>
<td>RIPEMD-160</td>
<td>219635621.6</td>
</tr>
<tr>
<td>M2090</td>
<td>SHA-256</td>
<td>120639441.8</td>
</tr>
<tr>
<td></td>
<td>SHA-512</td>
<td>37930735.9</td>
</tr>
<tr>
<td>NVIDIA</td>
<td>SHA-1</td>
<td>676654200.7</td>
</tr>
<tr>
<td>Tesla</td>
<td>RIPEMD-160</td>
<td>524113035.1</td>
</tr>
<tr>
<td>K20</td>
<td>SHA-256</td>
<td>178302193.3</td>
</tr>
<tr>
<td></td>
<td>SHA-512</td>
<td>33038175.2</td>
</tr>
<tr>
<td>NVIDIA</td>
<td>SHA-1</td>
<td>721176830.2</td>
</tr>
<tr>
<td>Tesla</td>
<td>RIPEMD-160</td>
<td>540764271.2</td>
</tr>
<tr>
<td>K20X</td>
<td>SHA-256</td>
<td>191052691.1</td>
</tr>
<tr>
<td></td>
<td>SHA-512</td>
<td>36163354.4</td>
</tr>
</tbody>
</table>

## Table 12. Attack estimation user configurations.

<table>
<thead>
<tr>
<th>Configuration</th>
<th>Distribution</th>
<th>Architecture</th>
<th>Backend</th>
<th>Cryptsetup version</th>
</tr>
</thead>
<tbody>
<tr>
<td>“worst-case”</td>
<td>Mint</td>
<td>32-bit</td>
<td>Libgcrypt</td>
<td>1.6.8</td>
</tr>
<tr>
<td>“best-case”</td>
<td>Arch</td>
<td>64-bit</td>
<td>OpenSSL</td>
<td>1.7.0</td>
</tr>
</tbody>
</table>
in 5 years. Note that the actual cost might be higher or lower than this value – it has to include the cost of consumed electricity and, on the other hand, can be lower by the residual price of the hardware after performing the attack (we assume that the hardware can be resold or used for a different purpose after the attack). For simplicity, our model assumes that the price of the device decreases linearly with the usage time.

Fig. 5. The attack’s initial hardware cost under “worst-case” configuration.

Fig. 6. The attack’s initial hardware cost under “best-case” configuration.

Figures 5 and 6 show the hardware costs calculated for the “worst-case” and “best-case” configurations, respectively. Notice that for the cheap NVIDIA GeForce GTX 465 GPU, the initial costs are an order of magnitude lower ($500-700K for “worst-case”) than for the other GPU devices ($2-11M for “worst-case”).

The overall values of the costs suggest that an attack on LUKS passwords would require a very large amount of resources and would be feasible only for very wealthy attackers. Also, even in the weakest configuration, the attack would require thousands of GPU devices in order to be successful after 5 years on average. Such amount of devices might pose serious challenges and incur other
costs (e.g., for the host computers, cooling, or physical storage), which we do not take into account.

5.2 Attack cost comparison

Figures 7 and 8 show the attack costs calculated for the “worst-case” and “best-case” configurations, respectively. Taking the minimum over all the GPU devices, we get a minimum cost of $1-1.5M for the “worst-case” configuration (for all hash functions). For the “best case” configuration, we get $5-7M with SHA-1 and RIPEMD-160 hash functions, $19M with SHA-256 and $43M with SHA-512.

5.3 Cryptsetup vs. VeraCrypt

Finally, we compared the cost of brute-force attacks on a LUKS partition created by Cryptsetup versions 1.6.8 and 1.7.0 to the cost of attacking a VeraCrypt partition.
VeraCrypt is the successor of a popular disk encryption software for the Windows operating system, TrueCrypt, which has been discontinued on 28 May 2014 [5,16]. Both TrueCrypt and VeraCrypt use PBKDF2 as the key derivation function. One of the most criticised problems of the original TrueCrypt was that the PBKDF2 iteration count was fixed and extremely low (1,000-2,000 iterations). In VeraCrypt, the iteration count has been raised to 327,661-655,331 iterations (depending on the hash function), but is still fixed [10]. Since version 1.12, the user can specify an optional “PIM” value to personalize the iteration count. However, since PIM has to be remembered by the user together with the password, this feature is unlikely to be used by the majority of users [8].

As seen in Figure 9, the estimated attack cost is comparable between VeraCrypt and Cryptsetup 1.7.0, while Cryptsetup 1.6.8 is falling behind both. Note that due to the fact that the iteration count is set to a fixed value in VeraCrypt, the disk unlocking time may be very high on legacy hardware (especially if the SHA-512 hash function is used). Since Cryptsetup picks the iteration count dynamically, it avoids this problem (assuming the encrypted disk is always used on the same/similar HW-SW configuration), although the attack cost may be reduced if the PBKDF2 computation is slow.

6 Discussion and conclusions

Although Argon2 is expected to supersede PBKDF2 in the next few years, currently PBKDF2 is still widely implemented to derive keys in many security-related systems. In this paper, we addressed the problem of long-term passwords secured by PBKDF2, presenting the case study of LUKS. We (a) analyzed how external factors may affect the iteration counts computation used by PBKDF2 to slow down attackers, (b) provided means of evaluating the iteration count values defined by real-world applications, and finally (c) showed that PBKDF2 is — and will still remain for years to come — capable to provide enough security margin for applications that require long-term keys.
More precisely, our testing activities identified four external factors that cannot be defined “weaknesses” (and cannot be patched) but which negatively affect the iteration count computation, improving the speed of a brute-force attack:

1. **Library versions**: Experimental results show that a security or performance bug coded into lines of an older version of the library can considerably affect the iteration count computation and, consequently, the attack cost. In our testing activities, we found old library versions in Live CD distributions, but also those available in official repositories are far from being up-to-date. This means that the majority of Linux distributions tested are affected by well-known LUKS weaknesses described in literature.

2. **Architectures (32/64-bit OS)**: We show an important gap between the iteration count values computed using 32-bit or 64-bit configurations. Experimental results suggest that, in several 32-bit Linux distributions, iteration counts can be reduced to a quarter when compared with the same 64-bit distros and, consequently, the attack cost too. Hence, from the user’s point of view, is more convenient use a 64-bit OS rather than a 32-bit OS.

3. **Backends**: The choice of cryptographic library to be used as the backend for Cryptsetup has a significant impact on the resulting iteration counts. Because the execution speed of cryptographic algorithms is due to independent optimizations coded in Libgcrypt and OpenSSL, our testing activities show that, on average, OpenSSL performs better than Libgcrypt.

4. **Linux distributions**: We analyzed the effects that a number of Linux distributions have on the iteration count computations. In some cases, the choice of a particular Linux distribution can help users to increase the iteration count values, and hence the attack cost, by about 36-40 percent. Moreover, we noted that the 32-bit Linux Mint with the Libgcrypt crypto backend performs below average, but we were not able to identify the reason behind this deviation.

Once external factors have been identified, we estimated the attack costs on the basis of the power consumed by the hardware doing the cracking and the cost of the hardware itself. Since there might be many other contributing factors — e.g., physical storage, cooling, or host computers — our estimations can only be seen as a lower bound for the actual cost.

Our results suggest that even in the weakest configuration, an attack on LUKS passwords would require a very large amount of resources — i.e., thousands of GPUs — in order to be successful in a reasonable amount of time (five years on average). In the worst-case scenario, which is based on the configurations that produce the worst iteration count value, the estimated cost is between $1M and $1.5M. In the best case scenario, which is based on the configurations that produce the best iteration count value, we get a minimum cost of $5-7M with SHA-1 and RIPEMD-160 and a maximum cost of $43M with SHA-512. Interestingly, the cost for SHA-512 is so high due to 64-bit arithmetic operations being less efficient on the GPU (as opposed to CPU).
7 Acknowledgment

Andrea Visconti was partially supported by a grant from Università degli Studi di Milano, “Piano di Sostegno alla Ricerca UNIMI 2015-2017”.

Ondrej Mosnáček and Vashek Matyáš were partially supported by the ITI centre of Czech Science Foundation, GBP202/12/G061. Ondrej Mosnáček and Milan Brož were also partially supported by Red Hat Czech. Computational resources were provided by the CESNET LM2015042 and the CERIT Scientific Cloud LM2015085, provided under the programme “Projects of Large Research, Development, and Innovations Infrastructures”.

References

2. EncFS Encrypted Filesystem, https://sites.google.com/a/arg0.net/www/encfs [Online, accessed 10-October-2016]


42. RSA Laboratories: PKCS #5 V2.1: Password Based Cryptography Standard (2012)

43. Shannon, C.E.: Prediction and entropy of printed english. Bell system technical journal 30(1), 50–64 (1951)


48. Yutaka, N.: Libgcrypt 1.7.0 Keep contexts for HMAC in GcryDigestEntry (2015), http://git.gnupg.org/cgi-bin/gitweb.cgi?p=libgcrypt.git;a=commit;h=f7505b550dd591e33d3a3fab9277c43c460f1bad [Online, accessed 10-October-2016]