MASARYK UNIVERSITY
FACULTY OF INFORMATICS

Key derivation functions and their GPU implementation

BACHELOR'S THESIS

Ondrej Mosnáček

Brno, Spring 2015

Declaration

Hereby I declare, that this paper is my original authorial work, which I have worked out by my own. All sources, references and literature used or excerpted during elaboration of this work are properly cited and listed in complete reference to the due source.

Ondrej Mosnáček

Advisor: Ing. Milan Brož

Acknowledgement

I would like to thank my supervisor for his guidance and support, and also for his extensive contributions to the Cryptsetup open-source project.

Next, I would like to thank my family for their support and patience and also to my friends who were falling behind schedule just like me and thus helped me not to panic.

Last but not least, access to computing and storage facilities owned by parties and projects contributing to the National Grid Infrastructure MetaCentrum, provided under the programme "Projects of Large Infrastructure for Research, Development, and Innovations" (LM2010005), is also greatly appreciated.

Abstract

TODO

Keywords

key derivation function, PBKDF2, GPU, OpenCL, CUDA, password hashing, disk encryption, password cracking, LUKS

Contents

1 Introduction

Encryption is the process of encoding information or data in such a way that only authorized parties can read it [12, 4]. The encryption uses a parameter – the key. The key is an information that is only known to the authorized parties and which is necessary to read the encrypted data. In general, any piece of information can be used as the key, but since it usually has to be memorized by a human, it often has the form of a password or passphrase.

Passwords and passphrases generally have the form of text (a variable-length sequence of characters), while most encryption algorithms expect a key in binary form (a long, usually fixed-size, sequence of bits or bytes). This means that for any password- or passphrase-based cryptosystem it is necessary to define the process of converting the password (passphrase) into binary form. Merely encoding the text using a common character encoding (e. g. ASCII or UTF-8) and padding it with zeroes is often not sufficient, because the resulting key might be susceptible to various attacks. An *attack* on a cryptographic key is an attempt by an unauthorized party to determine the key from publicly known information or from a certain partial information about the key (e. g. some knowledge about the domain from which the key was chosen, the first few bits of the key, etc.).

For this reason, a cryptographic primitive called *key derivation function* (KDF, plural KDFs) is used to derive encryption keys from passwords. KDFs are also often used for *password hashing* (transforming the password to a hash in such a way that it is easy to verify a given password against a hash, but infeasible to determine the original password from the hash) or *key diversification* (also *key separation*; deriving multiple keys from a master key so that it is infeasible to determine the master key or any other derived key from one or more derived keys) [14, 2].

KDFs usually have various security parameters, such as the number of iterations of an internal algorithm, which control the amount of time or memory required to perform the derivation in order to thwart brute-force attacks. Another common parameter is the cryptographic salt, which is a unique or random piece of data that is used together with the password to derive the key. Its main purpose is to protect against dictionary and rainbow table attacks and it is usually not kept secret [6].

One possible application of KDFs is key derivation from passwords in disk encryption software. Disk encryption software encrypts the contents of a storage device (such as a hard disk or a USB drive) or its part (a disk volume or *partition*) so that the data stored on the device can only be unlocked by one or more passwords or passphrases. The password/passphrase is entered when the user boots an operating system from the encrypted device or when they mount the encrypted partition to the filesystem.

An example of a disk encryption program is *cryptsetup*[1] which uses the LUKS standard as its main format for on-disk data layout. In version 1 LUKS uses PBKDF2 as the only KDF for deriving encryption keys from passwords [3]. However, PBKDF2 has a range of weaknesses, one of them being high susceptibility to brute-force and dictionary attacks using GPUs[2], as this thesis aims to demonstrate.

1.1 Goals

The goal of this work is to compare the speed of a brute-force attack on a specific key derivation function (PBKDF2) performed on standard computer processors against an attack using GPUs. Modern GPUs can be programmed using various high-level APIs (such as OpenCL[3], CUDA[4], DirectCompute or C++ AMP) and can be used not only for graphics processing but also for general purpose computation. Due to their specific architecture GPUs are suitable for parallel processing of massive amounts of data. Tasks that can be split into many small subtasks which can be run in parallel can be processed by a single GPU several times faster than by a single CPU. As was shown by Harrison and Waldron[5], using GPUs it is possible to accelerate also various algorithms of symmetric cryptography.

This work also includes analysis of susceptibility of PBKDF2 to

1. `https://gitlab.com/cryptsetup/cryptsetup/wikis/home`
2. GPU = Graphics Processing Unit
3. `https://www.khronos.org/opencl/`
4. `http://www.nvidia.com/object/cuda_home_new.html`

attacks using parallel processing and the implementation of an illustration program performing a brute-force attack on the password of a LUKS[5] encrypted partition.

1.2 Summary of results

TODO

1.3 Chapter contents

TODO

5. LUKS = Linux Unified Key Setup

2 Key derivation functions

Key derivation functions are cryptographic primitives that are used to derive encryption keys from a secret value. Depending on the application, the secret value can be another key or a password or passphrase [14]. A KDF that is designed for deriving cryptographic key from another key is called a *key-based key derivation function* (KBKDF); a KDF that is designed to take a password or passphrase as input is called a *password-based key derivation function* (PBKDF).

2.1 Key-based key derivation functions

Key-based key derivation functions are most often used to derive additional keys from a key that already has the properties of a cryptographic key – that is, it is a truly random or pseudorandom binary string that is computationally indistinguishable from one selected uniformly at random from the set of all binary strings of the same length [2].

Since the input to a KBKDF is already a cryptographic key, KBKDFs usually do not try to make brute-forcing more difficult by making the algorithm more computationally complex. A good cryptographic key has entropy of at least 128 bits, which means there are at least 2^{128} possible keys. Testing so many keys would be infeasible even with a very fast algorithm and an enormous computer cluster.

An example of a simple KBKDF is HKDF (HMAC-based extract-and-expand Key Derivation Function), which proceeds in two stages. The optional *extract* stage first extracts a suitable pseudorandom key from the (possibly low-entropy) input key material and an optional salt. Then the *expand* stage expands the extracted pseudorandom key, along with an optional context and application specific information (this can be used for key diversification), to the output key of the desired length [7, 8].

2.2 Password-based key derivation functions

As opposed to key-based key derivation functions, password-based key derivation functions are designed specifically to take low-entropy input such as a password or passphrase and to resist brute-force and dictionary attacks.

A brute-force attack is ...

A dictionary attack is ...

PBKDFs aim to reduce the feasibility of these attacks by increasing the amount of time and/or memory required to test a single key. The basic principle of this approach is that in practice, the extra time/memory requirements are only a minor inconvenience for a user (especially given that these measures provide an increased resistance against a mass attack), while for an attacker (who has to repeatedly process millions or billions of possible passwords) this means that the resources needed to perform a brute-force (or dictionary) attack increase significantly.

A well-designed PBKDF also takes into account the difference between the hardware that is used in the legitimate scenario and the hardware that the attacker might have available. A highly motivated and well-funded attacker might have access to massive amounts of computing power, might often be willing to wait even years until the password is found and might possess an expensive specialized hardware which would minimize the time and resources needed to successfully break the password. A typical user, on the other hand, will use a consumer-grade hardware (such as a personal computer or a laptop), so the PBKDF should be designed so that it performs with reasonable efficiency on the user's hardware, but at the same time it is difficult to utilize specialized hardware to gain advantage in an attack [10].

In order to protect against attacks using highly parallel architectures, modern PBKDFs use *sequential memory-hard functions*, which are designed in such a way that any time-efficient computation needs to use a certain configurable amount of memory. Requiring a certain non-trivial amount of memory to compute a single instance of the PBKDF increases the size of each compute unit, thus making any increase in parallelism more expensive (as opposed to functions that use only a small constant amount of memory) [9].

Another desirable property of PBKDFs is the ability to upgrade an existing derived key/hash to another having different (stronger) security parameters (e. g. the iteration count) without knowledge of the original password [10]. However, there are no PBKDFs having this property currently available.

The most widely used password-based key derivation functions are currently *PBKDF2* and *bcrypt*.

PBKDF2 was standardized under PKCS[1] #5, Version 2.0 in 1999 (also published as RFC 2898 in 2000 [6]) and later specified in NIST Special Publication 800-132 [13] as the only password-based key derivation function approved by the U.S. National Institute of Standards and Technology. PBKDF2 is widely used in many practical applications, such as Wi-Fi Protected Access (security protocols used to secure wireless networks), disk encryption software (Cryptsetup, TrueCrypt/VeraCrypt[2], ...) and password managers (LastPass[3], 1Password[4], ...).

Bcrypt was introduced in 1999 [11] and although it is somewhat more secure than PBKDF2, it is not as widely used.

In 2009, a new password-based key derivation function *scrypt* was introduced [9], which uses the aforementioned sequential memory-hard functions.

In 2013, an open competition called *Password Hashing Competition*[5] was announced. The competition aims to "identify new password hashing schemes in order to improve on the state-of-the-art" [1]. The competition is organized by a group of cryptography experts, not by a standardization body. It is expected, however, that it will lead to a new standard for password-based key derivation and password hashing.

1. PKCS = Public-Key Cryptography Standards; a group of standards published by RSA Security, Inc. (see https://www.emc.com/emc-plus/rsa-labs/standards-initiatives/public-key-cryptography-standards.htm)
2. https://veracrypt.codeplex.com/
3. https://lastpass.com/
4. https://agilebits.com/onepassword
5. https://password-hashing.net/

2.3 PBKDF2

TODO

2.4 Scrypt

TODO

3 Attacks on key derivation functions

4 Acceleration of algorithms using GPUs

5 Comparison of CPU and GPU attack speeds

6 Implementing PBKDF2 on GPUs

7 Conclusion

Bibliography

[1] Password hashing competition, 2015. [Online, accessed 30-April-2015, retrieved from https://password-hashing.net/].

[2] Chen, L. Recommendation for key derivation using pseudorandom functions. NIST Special Publication 800-108, The U.S. National Institute of Standards and Technology, November 2008. [Available at http://csrc.nist.gov/publications/nistpubs/800-108/sp800-108.pdf].

[3] Fruhwirth, C., and Brož, M. LUKS on-disk format specification, October 2011. [Online, accessed 27-April-2015, retrieved from https://gitlab.com/cryptsetup/cryptsetup/wikis/LUKS-standard/on-disk-format.pdf].

[4] Goldreich, O. *Foundations of Cryptography: Volume 2, Basic Applications*. Cambridge University Press, New York, NY, USA, 2004.

[5] Harrison, O., and Waldron, J. Practical symmetric key cryptography on modern graphics hardware. In *Proceedings of the 17th Conference on Security Symposium* (Berkeley, CA, USA, 2008), SS'08, USENIX Association, pp. 195–209. [Available at https://www.usenix.org/legacy/event/sec08/tech/full_papers/harrison/harrison.pdf].

[6] Kaliski, B. PKCS #5: Password-based cryptography specification. RFC 2898, RFC Editor, September 2000. [Available at https://tools.ietf.org/html/rfc2898].

[7] Krawczyk, H. Cryptographic extraction and key derivation: The HKDF scheme. Cryptology ePrint Archive, Report 2010/264, International Association for Cryptologic Research, 2010. [Available at https://eprint.iacr.org/2010/264].

[8] Krawczyk, H., and Eronen, P. HMAC-based extract-and-expand key derivation function (HKDF). RFC 5869, RFC Editor, May 2010. [Available at https://tools.ietf.org/html/rfc5869].

[9] PERCIVAL, C. Stronger key derivation via sequential memory-hard functions, May 2009. [Online, accessed 30-April-2015, retrieved from https://www.tarsnap.com/scrypt/scrypt.pdf].

[10] PESLYAK, A., AND MARECHAL, S. Password security: past, present, future. Openwall, Inc., December 2012. [Online, accessed 30-April-2015, retrieved from http://www.openwall.com/presentations/Passwords12-The-Future-Of-Hashing/Passwords12-The-Future-Of-Hashing.pdf].

[11] PROVOS, N., AND MAZIÈRES, D. A future-adaptable password scheme. In *Proceedings of the Annual Conference on USENIX Annual Technical Conference* (Berkeley, CA, USA, 1999), ATEC '99, USENIX Association, pp. 32–32. [Available at https://www.usenix.org/legacy/publications/library/proceedings/usenix99/provos/provos.pdf].

[12] SCHNEIER, B. *Applied Cryptography: Protocols, Algorithms, and Source Code in C*, 2nd ed. John Wiley & Sons, Inc., New York, NY, USA, 1996.

[13] TURAN, M. S., BARKER, E. B., BURR, W. E., AND CHEN, L. Recommendation for password-based key derivation, Part 1: Storage applications. NIST Special Publication 800-132, The U.S. National Institute of Standards and Technology, December 2010. [Available at http://csrc.nist.gov/publications/nistpubs/800-132/nist-sp800-132.pdf].

[14] WIKIPEDIA CONTRIBUTORS. Key derivation function — Wikipedia, the free encyclopedia. Wikimedia Foundation, Inc., 2015. [Online, accessed 11-April-2015, retrieved from https://en.wikipedia.org/w/index.php?title=Key_derivation_function&oldid=644734578].