INSO - Industrial Software
Institute of Computer Aided Automation | Faculty of Informatics | TU Wien

Exposé of Bachelor Thesis

Lyra2 implementation in Java

Advisor: Clemens Hlauschek
Assistance: Clemens Hlauschek

Aleksandr Lisianoi
01527346
033 534
Bachelorstudium Software & Information Engineering

Datum ....
Clemens Hlauschek

Exposé of Bachelor Thesis

1 Problem Description

Password-based authentication is the most widespread method of proving the identity of a user or approving access to a resource. It is common practice to use Password Hashing Schemes (PHS for short) in order to avoid manipulating the password in cleartext. These schemes are often built by combining other cryptographic primitives (most notably, cryptographic hash functions) in order to make the resulting hash as difficult to invert as possible. An existing PHS called Lyra2 [1] and its reference implementation are the primary focus of this thesis.

2 Expected Results

There are several goals for this thesis:

1. Provide a test harness for the existing implementation of Lyra2.
2. Port the generic single-threaded Lyra2 variant to Java.
3. Assess algorithm-level compatibility of the implementations.
4. Compare performance and other practical details of the implementations.

3 Methodological Approach

The core of Lyra2 [1] is a *cryptographic sponge*, a simple yet elegant construction which in turn heavily relies on a fixed-width permutation. Blake2b [2] or BlaMka [1] are good examples of existing cryptographic hash functions which are often used as building blocks in cryptographic sponges. However, the theoretical properties of neither cryptographic hash functions nor cryptographic sponges are the focus of this thesis. Instead, this work will focus on build and test automation and reproducibility of benchmarking results. In order to achieve that, the following will be done:

3.1 Test harness development

Current Lyra2 implementation is written in C, its build system is a single makefile and there are shell scripts that run basic benchmarks. The makefile supports six build targets (`generic-x86-64`, `linux-x86-64-sse`, `linux-x86-64-cuda`, etc.), three sponges (Blake2b, BlaMka and half-round BlaMka), two modes of parallelism (single- and multithreaded) and the number of columns of the memory matrix is also set at compile time. This results in a (really) big build matrix. So, the executables it produces should be tested automatically which is currently not done at all. As the first step of this thesis, an improved build system and a unit test system will be developed. There are many ways to do this but the following is proposed:

1. Write a `Python` script to automate the `makefile` building process. Here is its first version.
2. Write `pytest` unit tests. They allow easy parallel testing. Here are first sanity tests.
3. Write a `Python` script that would generate hash values for specific passwords/parameters.

4. Write further unit tests to verify that new Lyra2 executables generate the same hashes.

There are many reasons to introduce yet another language (`Python`) and its accompanying infrastructure to this project. For example, authors of Lyra2 are already moving away from shell scripts to `Python` scripts for benchmarking. If necessary, further reasons may be explained (i.e. good subprocess control for unit testing of executables compiled from `C` or `Java`).

3.2 Lyra2 implementation in Java

There are several scenarios when a Java implementation might be better than the original C implementation. For example, developers for Android devices might benefit from having a native library. Moreover, the porting process also means that another developer takes a look at the original implementation and can contribute improvements, see here for an example.

3.3 Algorithm level compatibility

This is the primary goal of the thesis: to implement Lyra2 in Java in such a way that it produces identical hashes (given the same password, salt and other parameters) when compared with the existing C implementation. In order to achieve that, the steps described above in 3.1 are essential. However, one must also account for unexpected discrepancies that might disallow such compatibility. If that will be the case, an explanation will be provided.

3.4 Performace comparison

This would be a logical conclusion for the thesis. Lyra2 was compared to other participants of the Password Hashing Competition in [7], [5] and this work will take a similar approach.

4 State of the Art

The most widely used PHSs are PBKDF2, bcrypt and scrypt [7][1]. Both PBKDF2 [8] and bcrypt [13] allow their users to adjust the processing costs while scrypt [10] allows to adjust both processing and memory costs. Such flexibility is motivated by the constant advances in parallel computing and dedicated hardware, as shown in [9]. In fact, the need for flexible password hashing schemes is exactly the reason why the Password Hashing Competition was announced. More than 20 different password hashing schemes were examined. The proposed schemes include the winner Argon2 [3] as well as Lyra2, Catena [6], Makawa [12], yescrypt [11] and others. Also, there is growing interest in rigorous mathematical analysis of PHSs [4].

5 Table of Contents

Preliminary structure of the thesis (about 1 page)

1. Introduction [**3 pages**]
2. Theoretical background [**7 Seiten**]
   - 2.1. Cryptographic sponges [**2 pages**]
     - 2.1.1. Definition and examples [**1 page**]
     - 2.1.2. Essential properties [**1 page**]
   - 2.2. Lyra2 algorithm [**5 pages**]
     - 2.2.1. Bootstrapping [**1 page**]
     - 2.2.2. Setup phase [**1 page**]
     - 2.2.3. Wandering phase [**1 page**]
     - 2.2.4. Wrap-up phase [**1 page**]
     - 2.2.5. Underlying sponge [**1 page**]
3. Lyra2 implementation [**2 pages**]
   - 3.1. Structure of the C implementation [**1 page**]
   - 3.2. Structure of the Java implementation [**1 page**]
4. Lyra2 comparison [**4 pages**]
   - 4.1. Structure of the comparison project [**1 page**]
   - 4.2. Comparison method, reproducibility [**1 page**]
   - 4.3. Comparison results [**2 pages**]
     - 4.3.1. Implementation compatibility [**1 page**]
     - 4.3.2. Comparative performance [**1 page**]
5. Discussion [**1 page**]

References

[1] E. R. Andrade et al. “Lyra2: Efficient Password Hashing with High Security against Time-Memory Trade-Offs”. In: *IEEE Transactions on Computers* 65.10 (2016), pp. 3096–3108. ISSN: 0018-9340. DOI: 10.1109/TC.2016.2516011.

[2] Jean-Philippe Aumasson et al. “BLAKE2: Simpler, Smaller, Fast As MD5”. In: *Proceedings of the 11th International Conference on Applied Cryptography and Network Security*. ACNS’13. Banff, AB, Canada: Springer-Verlag, 2013, pp. 119–135. ISBN: 978-3-642-38979-5. DOI: 10.1007/978-3-642-38980-1_8. URL: http://dx.doi.org/10.1007/978-3-642-38980-1_8.

[3] Alex Biryukov, Daniel Dinu, and Dmitry Khovratovich. “Argon2: new generation of memory-hard functions for password hashing and other applications”. In: *Security and Privacy (EuroS&P), 2016 IEEE European Symposium on*. IEEE. 2016, pp. 292–302.

[4] Alex Biryukov, Daniel Dinu, and Dmitry Khovratovich. *Fast and Tradeoff-Resilient Memory-Hard Functions for Cryptocurrencies and Password Hashing*. Cryptology ePrint Archive, Report 2015/430. http://eprint.iacr.org/2015/430. 2015.

[5] Donghoon Chang et al. “Performance Analysis of Some Password Hashing Schemes.” In: *IACR Cryptology ePrint Archive* 2015 (2015). http://ai2-s2-pdfs.s3.amazonaws.com/9ee2/aacb6aa8eb102503a2cfc840ced5db893777.pdf, p. 139.

[6] Christian Forler, Stefan Lucks, and Jakob Wenzel. *Catena: A memory-consuming password-scrambling framework*. Tech. rep. Citeseer, 2013.

[7] George Hatzivasilis, Ioannis Papaefstathiou, and Charalampos Manifavas. *Password Hashing Competition - Survey and Benchmark*. Cryptology ePrint Archive, Report 2015/265. http://eprint.iacr.org/2015/265. 2015.

[8] K Moriarty, B Kaliski, and A Rusch. *PKCS# 5: Password-Based Cryptography Specification Version 2.1*. Tech. rep. https://tools.ietf.org/html/rfc8018. 2017.

[9] Hilarie Orman. “Twelve Random Characters: Passwords in the Era of Massive Parallelism”. In: *IEEE Internet Computing* 17.5 (2013), pp. 91–94.

[10] Colin Percival. *Stronger Key Derivation via Sequential Memory-Hard Functions*. BSFCan09. https://www.tarsnap.com/scrypt/scrypt.pdf. 2009.

[11] Alexander Peslyak. *yescrypt - a Password Hashing Competition submission*. Tech. rep. https://password-hashing.net/submissions/specs/yescrypt-v2.pdf. 2015.

[12] Thomas Pornin. *The Makwa password hashing function*. http://www.bolet.org/makwa/makwa-spec-20150422.pdf. 2015.

[13] Niels Provos and David Mazieres. “A Future-Adaptable Password Scheme.” In: https://www.usenix.org/legacy/events/usenix99/full_papers/provos/provos.pdf. 1999.