Lyra2: Password Hashing Scheme with improved security against time-memory trade-offs*

Ewerton R. Andrade[1,2], **Marcos A. Simplicio Junior**[1]

[1] Laboratório de Arquitetura e Redes de Computadores (LARC)
Escola Politécnica da Universidade de São Paulo (Poli-USP) – São Paulo – Brasil.

[2]Instituto Federal de Rondônia (IFRO) – Rondônia – Brasil.

{eandrade,mjunior}@larc.usp.br

Abstract. *To protect against brute force attacks, modern password-based authentication systems usually employ mechanisms known as Password Hashing Schemes (PHS). Basically, a PHS is a cryptographic algorithm that generates a sequence of pseudorandom bits from a user-defined password, allowing the user to configure the computational costs involved in the process aiming to raise the costs of attackers testing multiple passwords trying to guess the correct one. In this context, the goal of this research effort is to propose a novel and superior PHS alternative. Specifically, the objective is to improve the Lyra algorithm, a PHS built upon cryptographic sponges whose project counted with the authors' participation. The resulting solution, called Lyra2, preserves the efficiency and flexibility of Lyra, and it brings important improvements when compared to its predecessor: (1) it allows a higher security level against attack venues involving time-memory trade-offs; (2) it includes tweaks for increasing the costs involved in the construction of dedicated hardware to attack; (3) it balances resistance against side-channel threats and attacks relying on cheaper (and, hence, slower) storage devices. Besides describing the algorithm's design rationale in detail, the thesis also includes a detailed analysis of its security and performance.*

1. Introduction

User authentication is one of the most vital elements in modern computer security. Even though there are authentication mechanisms based on biometric devices ("what the user is") or physical devices such as smart cards ("what the user has"), the most widespread strategy still is to rely on secret passwords ("what the user knows"). This happens because password-based authentication remains as the most cost effective and efficient method of maintaining a shared secret between a user and a computer system (Chakrabarti and Singbal 2007).

Password-based systems usually employ some cryptographic algorithm that allows the generation of a pseudorandom string of bits from the password itself, known as a Password Hashing Scheme (PHS), or Key Derivation Function (KDF) (NIST 2009). Typically, the output of the PHS is employed in one of two manners (Percival 2009): it can be locally stored in the form of a "token" for future verifications of the password or used as the secret key for encrypting and/or authenticating data. Whichever the case, such solutions employ internally a one-way (e.g., hash) function, so that recovering the password from the PHS's output is computationally infeasible (Percival 2009; Kaliski 2000).

* Full thesis and implementations available at http://lyra2.net/.

Despite the popularity of password-based authentication, the fact that most users choose quite short passwords leads to a serious issue: they commonly have much less entropy than typically required by cryptographic keys (NIST 2011). Such weak passwords greatly facilitate many kinds of "brute-force" attacks, such as dictionary attacks and exhaustive search (Chakrabarti and Singbal 2007), allowing attackers to completely bypass the non-invertibility property of the password hashing process. The feasibility of such attacks depends basically on the amount of resources available to the attacker, who can speed up the process by performing many tests in parallel.

2. Motivation

A straightforward approach for addressing this problem is to force users to choose complex passwords. This is unadvised, however, because such passwords would be harder to memorize and, thus, more easily forgotten due to the users' need of writing them down, defeating the whole purpose of authentication (Chakrabarti and Singbal 2007). For this reason, modern password hashing solutions usually employ mechanisms for increasing the *cost* of brute force attacks. Schemes such as PBKDF (Kaliski 2000) and bcrypt (Provos and Mazières 1999), for example, include a configurable parameter that controls the number of iterations performed, allowing the user to adjust the time required by the password hashing process.

A more recent proposal, scrypt (Percival 2009), allows users to control both processing time and memory usage, raising the cost of password recovery by increasing the silicon space required for running the PHS in custom hardware, or the amount of RAM required in a Graphics Processing Unit (GPU). Since this may raise the RAM costs of password cracking to unbearable levels, attackers may try to trade memory for processing time, discarding (parts of) the memory used and recomputing the discarded information when it becomes necessary (Percival 2009). The exploitation of such time-memory trade-offs (TMTO) leads to the hereby-called *low-memory attacks*. Another approach that might be used by attackers trying to reduce the costs of password cracking is to use low-cost (and, thus, slower) storage devices for keeping all memory used in the legitimate process, using the spare budget to run more tests in parallel and, thus, compensating the lower speed of each test; we call this approach a *slow-memory attack*. Besides the need for protection against low- and slow-memory attacks, there is also interest in the development of solutions that are safe against side-channel attacks, in especial the so-called *cache-timing attacks*. Basically, a cache-timing attack is possible if the attacker can observe a machine's timing behavior by monitoring its access to cache memory (e.g., the occurrence of cache-misses), building a profile of such occurrences for a legitimate password hashing process (Forler et al. 2013). Then, at least in theory, if the password being tested does not match the observed cache-timing behavior, the test could be aborted earlier, saving resources.

The considerable interest by the research community in developing new (and better) password hashing alternatives has recently even led to the creation of a cryptographic competition with this specific purpose, the Password Hashing Competition (PHC 2013).

3. Goals and Original Contributions

Aiming to address this need for stronger alternatives, our early studies led to the proposal of Lyra (Almeida et al. 2014). In this research, we propose an improved version of

Lyra, called simply Lyra2. Basically, Lyra2 preserves the flexibility and efficiency of Lyra, including: (1) the ability to configure the desired amount of memory and processing time to be used by the algorithm; (2) the capacity of providing a higher memory usage than what is obtained with scrypt for a similar processing time. In addition, it brings important security improvements when compared to its predecessor: (1) it allows a higher security level against attack venues involving time-memory trade-offs (TMTO); (2) it includes tweaks to increase the costs involved in the construction of dedicated hardware for attacking the algorithm (e.g., FPGAs or ASICs); (3) it balances resistance against side-channel threats and attacks relying on cheaper (and, hence, slower) storage devices.

4. Brief overview of the state-of-the-art

Arguably, the main password hashing solutions currently in use are (PHC 2013): PBKDF2 (Kaliski 2000), bcrypt (Provos and Mazières 1999) and scrypt (Percival 2009). Nevertheless, until recently scrypt was the only solution available in the literature exploring both memory and processing costs. This scenario changed with Lyra (Almeida et al. 2014), which also introduced improvements such as: the decoupling of the memory and processing parameters, giving further flexibility to users; the usage of a single underlying function (a cryptographic sponge) rather than the two employed in scrypt; higher resistance against attacks exploiting time-memory trade-offs; and higher performance, allowing for a higher memory usage with a similar processing time.

Over the last years, some candidates submitted to the Password Hashing Competition (namely, Argon2, Catena, and yescrypt, besides Lyra2 itself, to cite only algorithms deemed as recommended) also allow memory and processing time to be configured, and some of them (especially Catena, Argon2 and Lyra2) try to provide a thorough security analysis. Due to space limitations, we do not analyze each solution in this document, limiting the discussion to the comparative performance and security assessment in Sections 5.1 and 5.2. Nevertheless, we refer the interested reader to (Almeida et al. 2014) for a discussion on scrypt and improvements proposed in Lyra (and inherited by Lyra2), the PHC official website (PHC 2013) for details on each submission, and also to our thesis.

5. Lyra2

As any PHS, Lyra2 takes as input a salt and a password to create a pseudorandom output. Internally, its memory is organized as a two-dimensional array whose cells are iteratively read and written, called simply the *memory matrix*. Hence, discarding a cell for saving memory leads to the need of recomputing it, until the point it was last modified, whenever it is accessed again. The matrix's initialization and visitation uses a stateful combination of the underlying sponge's absorbing, squeezing and duplexing operations, and was designed specifically to avoid password cracking using commom attack platforms such as GPUs and dedicated hardware. Also, users can define the matrix's size and the number of times its cells are revisited after initialization, allowing Lyra2's resource usage to be fine-tuned. Due to space limitations, here we limit the discussion of Lyra2's design to this brief overview, whereas the detailed design, some possible variants, as well as an extended security and performance analysis can be found at Lyra2's reference guide and PhD thesis — available at `www.lyra2.net` —, and published papers (Andrade et al. 2016).

Table 1. Security overview of the PHSs considered the state of art. In the TMTO analysis, the time parameter is denoted by T and the memory is denoted by R.

Algorithm	Time-Memory trade-offs (TMTO)	Slow Memory	Side Channel	Hardware and GPUs	Garbage Collector
Argon2i	for R' $R/6$ $\approx 2^{15.5} \cdot T \cdot R$	—	✓	✓	✓
Argon2d	for R' $R/6$ $\approx 2^{19.6} \cdot T \cdot R$	✓	✗	✓	✓
battcrypt	—	✓	✗	✓	✓
Catena	O 1 Θ R^{T+1}	—	✓	✓	✓
Lyra2 [ours]	for R' $R/2^{n+2}$, where $n \geq 0$ O $2^{2nT}R^{2+n}$, for $n \gg 1$	✓	!	✓	✓
POMELO	—	✓	!	✓	✓
yescrypt	O 1 O R^{T+1}	✓	✗	✓	✓[1]
scrypt	O 1 O R^2	✓	✗	!	✗

✓- Has protection; ✗- Has no protection; ! - Partial protection; — - Nothing declared.

5.1. Security

Table 1 summarizes the Lyra2's security analysis, providing a comparison with other Password Hashing Schemes that are considered part of the state of the art. We note that, among the results appearing in this table, only the data related to Lyra2 were actually analyzed and presented in our thesis, while the rest of the data shown were collected on the reference guides, manuals and articles describing and/or analyzing the other solutions, including also third-party analysis. As can be seen in this table, Lyra2 is the only solution that provides at least partial protection against all known attacks.

5.2. Benchmarks

In our assessment of Lyra2's performance, we used an SSE-enabled implementation of Blake2b's (Aumasson et al. 2013) and BlaMka's (Andrade 2016, Sec. 4.4.1) compression function as the underlying sponge's f function of Lyra2's Algorithm. The actual implementations, as well as test vectors, are available at `www.lyra2.net`. Figure 1 shows the results for Lyra2 using Blake2b's as compression function, with C 256, ρ 1, and different T and R settings. As depicted, Lyra2 is able to execute in less than 1 s when using up to 1 GB of memory. The testbed employed is an Intel Xeon E5-2430 (2.20 GHz, W 64 bits) equipped with 48 GB of DRAM, running Ubuntu 14.04 LTS 64 bits. The source code was compiled using gcc and g++ 4.9.2.

The same Figure 1 also compares Lyra2 with scrypt's SSE-enabled implementation available at `www.tarsnap.com/scrypt.html`, using the parameters suggested by scrypt's author in (Percival 2009). We also performed tests aiming to compare the performance of Lyra2 and the other 5 memory-hard PHC finalists: Argon2, battcrypt, Catena, POMELO, and yescrypt. The results, which basically confirm existing analysis done in (Broz 2014) and show that Lyra2 is a very competitive solution in terms of performance, are depicted in Figure 1.

[1] Provides protection when in its "read-write" mode (`YESCRYPT_RW`) (Forler et al. 2014).

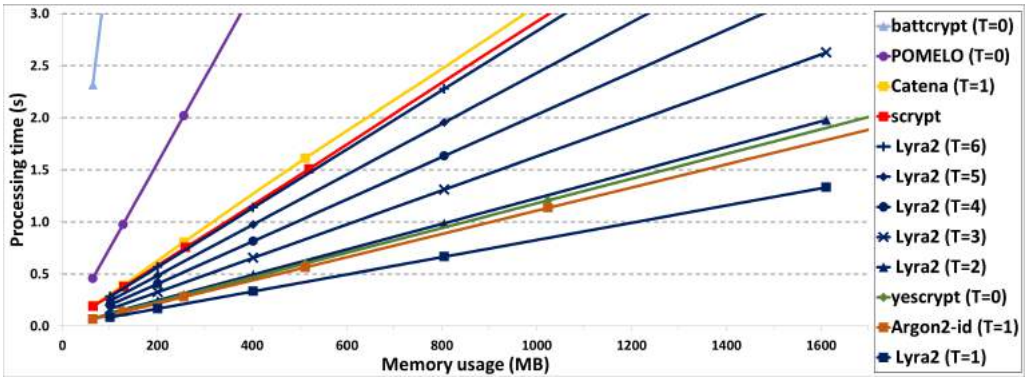

Figure 1. Performance of Lyra2, for C 256, ρ 1, p 1, and different T and R settings, compared with scrypt and memory-hard PHC finalists with minimum parameters.

6. Final Considerations

In this document, we presented the main aspects of Lyra2, a password hashing scheme that allows legitimate users to fine tune memory and processing costs according to the desired level of security and resources available in the target user platform. In summary, the combination of a strictly sequential design for thwarting attacks with parallel platforms (e.g. GPUs or ASICs), the high costs of exploring time-memory trade-offs, and the ability to raise the memory usage beyond what is attainable with similar-purpose solutions for a similar security level and processing time make Lyra2 an appealing PHS solution.

6.1. Publications and other results

The following adoptions, publications, presentations, submissions and envisioned papers are a direct or indirect result of the research effort carried out during this thesis:

- *PHC special recognition*: Lyra2 received a special recognition for its elegant sponge-based design, and alternative approach to side-channel resistance combined with resistance to slow-memory attacks (PHC 2013).
- *BlaMka's adoption*: the winner of the PHC, Argon2, adopts BlaMka by default as its permutation function (Biryukov et al. 2016).
- *Lyra2's adoptions*: the Vertcoin electronic currency announced that it is migrating from scrypt to Lyra2 (a432511 2014). Furthermore, one of the major bitcoin mining software on GPU, the Sgminer, added support to Lyra2 in its distribution package (Crypto Mining 2015). Recently, ZCoin adopted Lyra2 due its resistance against ASIC mining algorithms (ZCoin 2016).
- *Journal Articles*: in (Almeida et al. 2014), we present the Lyra algorithm, describing the preliminary ideas that gave rise to Lyra2; in (Andrade et al. 2016), we describe Lyra2 and provide a brief security and performance analysis.
- *Extended abstract*: in (Andrade and Simplicio Jr 2014b), we present the initial ideas and results of Lyra2; in (Andrade and Simplicio Jr 2014a), we gave an oral presentation at LatinCrypt'14 with some preliminary results; and in (ICISSP 2016; SBSeg16 2016) we present the current status of our project.
- *Award*: recently, we received a best PhD project award (ICISSP 2016), best PhD thesis award (SBSeg16 2016).

Acknowledgements

This work was supported by CNPq (305350/2013-7 and 473916/2013-4), FAPESP (2015/50520-6 and 2013/25977-9), CAPES (79414400249), ErasmusMundus, and FDTE.

References

a432511 (2014). PoW Algorithm Upgrade: Lyra2 – Vertcoin. https://vertcoin.org/.

Almeida, L., Andrade, E., Barreto, P., and Simplicio, M. (2014). Lyra: Password-based key derivation with tunable memory and processing costs. *JCE*, 4(2):75–89.

Andrade, E. R. (2016). *Lyra2: Password Hashing Scheme with improved security against time-memory trade-offs*. PhD thesis, Escola Politécnica da Universidade de São Paulo (Poli-USP), São Paulo. Available from: http://lyra2.net/.

Andrade, E. R. and Simplicio Jr, M. A. (2014a). Lyra2: a password hashing schemes with tunable memory and processing costs. LATINCRYPT'14. Brazil.

Andrade, E. R. and Simplicio Jr, M. A. (2014b). Lyra2: Um Esquema de Hash de Senhas com custos de memória e processamento ajustáveis. 53–55, October, III WPG-EC. São Paulo, SP, Brazil. http://www2.pcs.usp.br/wpgec/2014/index.htm.

Andrade, E. R., Simplicio Jr, M. A., Barreto, P. S. L. M., and Santos, P. C. F. d. (2016). Lyra2: efficient password hashing with high security against time-memory trade-offs. *IEEE Transactions on Computers*, PP(99).

Aumasson, J.-P., Neves, S., Wilcox-O'Hearn, Z., and Winnerlein, C. (2013). BLAKE2: simpler, smaller, fast as MD5. https://blake2.net/.

Biryukov, A., Dinu, D., and Khovratovich, D. (2016). *Argon2: the memory-hard function for password hashing and other applications*. PHC, v1.3 of Argon2 edition.

Broz, M. (2014). Another PHC candidates "mechanical" tests – pub archives of PHC list.

Chakrabarti, S. and Singbal, M. (2007). Password-based authentication: Preventing dictionary attacks. *Computer*, 40(6):68–74.

Crypto Mining (2015). Updated Windows Binary of sgminer 5.1.1 With Fixed Lyra2Re Support – Crypto Mining Blog. http://cryptomining-blog.com/.

Forler, C., List, E., Lucks, S., and Wenzel, J. (2014). Overview of the Candidates for the Password Hashing Competition - And Their Resistance Against Garbage-Collector Attacks. Cryptology ePrint Archive. http://eprint.iacr.org/2014/881.

Forler, C., Lucks, S., and Wenzel, J. (2013). Catena: A memory-consuming password scrambler. Cryptology ePrint Archive. eprint.iacr.org/2013/525.

ICISSP (2016). Previous awards. International Conference on Information Systems Security and Privacy – website. http://www.icissp.org/PreviousAwards.aspx.

Kaliski, B. (2000). *PKCS#5: Password-Based Cryptography Specif. v2.0 (RFC 2898)*.

NIST (2009). *Special Publication 800-18 – Recommendation for Key Derivation Using Pseudorandom Functions*. NIST, USA.

NIST (2011). *Special Publication 800-63-1 – Electronic Authentication Guideline*. NIST.

Percival, C. (2009). Stronger key derivation via sequential memory-hard functions. In *BSD Conference*.

PHC (2013). Password Hashing Competition. https://password-hashing.net/.

Provos, N. and Mazières, D. (1999). A future-adaptable password scheme. In *USENIX'99*.

SBSeg16 (2016). Trabalhos premiados. XVI SBSeg. http://sbseg2016.ic.uff.br/pt/trabalhos-premiados.php.

ZCoin (2016). Lyra2 Mining switch: Update your wallet. http://blog.zcoin.tech/lyra2-mining-update/.