View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by Repositorio Institucional de la Universidad de Alicante

symmetry

MDPI

Article

Optimizing a Password Hashing Function with Hardware-Accelerated Symmetric Encryption

Rafael Álvarez [1,*], Alicia Andrade [2] and Antonio Zamora [3]

1 Departamento de Ciencia de la Computación e Inteligencia Artificial (DCCIA), Universidad de Alicante, 03690 Alicante, Spain

2 Fac. Ingeniería, Ciencias Físicas y Matemática, Universidad Central, Quito 170129, Ecuador; aandrade@uce.edu.ec

3 Departamento de Ciencia de la Computación e Inteligencia Artificial (DCCIA), Universidad de Alicante, 03690 Alicante, Spain; zamora@dccia.ua.es

* Correspondence: ralvarez@dccia.ua.es

Received: 2 November 2018; Accepted: 22 November 2018; Published: 3 December 2018

Abstract: Password-based key derivation functions (PBKDFs) are commonly used to transform user passwords into keys for symmetric encryption, as well as for user authentication, password hashing, and preventing attacks based on custom hardware. We propose two optimized alternatives that enhance the performance of a previously published PBKDF. This design is based on (1) employing a symmetric cipher, the Advanced Encryption Standard (AES), as a pseudo-random generator and (2) taking advantage of the support for the hardware acceleration for AES that is available on many common platforms in order to mitigate common attacks to password-based user authentication systems. We also analyze their security characteristics, establishing that they are equivalent to the security of the core primitive (AES), and we compare their performance with well-known PBKDF algorithms, such as Scrypt and Argon2, with favorable results.

Keywords: symmetric; encryption; password; hash; cryptography; PBKDF

1. Introduction

Key derivation functions are employed to obtain one or more keys from a master secret. This is especially useful in the case of user passwords, which can be of arbitrary length and are unsuitable to be used directly as fixed-size cipher keys, so, there must be a process for converting passwords into secret keys. This process is performed by password-based key derivation functions (PBKDFs). PBKDFs are also called password hashing functions, and they are commonly employed in user authentication since they have certain advantages over other password processing methods: they are capable of accepting a salt, preventing precalculated table attacks; they are one-way functions (much as ordinary cryptographic hash functions), so the hashed password database cannot be reversed if it is stolen; and they can usually be parameterized in terms of temporal and memory cost to prevent attacks based on massively parallel hardware, like general-purpose graphical processing units (GPGPU) or custom hardware. Password hashing is a field of active research (see [1,2]), with several recent publications [3–10] that improve the current industry standard (PBKDF2, see [11]). Besides password hashing and key derivation, PBKDFs have found applications in the field of cryptocurrencies and blockchain algorithms, where they are used as proof-of-work functions for such designs (see [12]).

Symmetric encryption (see [13]) is a type of cryptography that employs the same (or easily derivable from one another) keys for encryption and decryption, hence the establishment of a symmetric process from *cleartext* to *ciphertext* and, back again, from *ciphertext* to *cleartext*. There are two basic kinds of symmetric cryptosystems: block ciphers and stream ciphers; they differ in

that block ciphers have no internal state and usually process data in blocks, while stream ciphers do have an internal state and process data element by element (an element is usually a bit or a byte of data). Nevertheless, most block ciphers can be run in operation modes that make them work as stream ciphers; such is the case in our proposal, where we employ the current advanced encryption standard (AES, [14]) in counter (CTR) mode to work as a stream cipher in the role of a pseudo-random number generator (PRNG); the use of AES as a PRNG has been proposed by the United States National Institute of Standards and Technology (NIST, see [15]). Besides having been independently tested for almost two decades and considered secure by the community, AES has the advantage of being accelerated in hardware on most common modern processors, like those found on laptop, desktop or server machines we use nowadays.

The main contributions of this paper are two different optimizations of a previously proposed PBKDF (see [3]) that favorably compare in performance to the original version and widely employed PBKDFs, such as Scrypt [7] and Argon2 [5]. This is significant for user authentication applications that are based on passwords, as well as in blockchain applications where a proof-of-work algorithm is required. Taking advantage of the fact that hardware acceleration support is available for AES, our proposed PBKDF design reduces the performance advantage of attackers employing GPGPU or custom hardware since its main core primitive is also run on hardware.

The rest of the paper is structured as follows: a short review on related work is included in Section 2, the proposed optimized algorithms are described in Section 3, the results obtained by our studies are presented in Section 4, the significance of the results are discussed in Section 5, while the testing methodology is detailed in Section 6, followed by some conclusions and future lines of work in Section 7.

2. Related Work

There is abundant recent literature on the connection between symmetry and cryptography. Chang et al. proposed a mobility network authentication scheme based on elliptic-curve cryptography (see [16]) that ensures anonymity, security, and convenience. Hung et al. designed a lattice-based revocable certificateless signature scheme (see [17]) that aims to resist cryptanalysis, even in the post-quantum era. Sakalauskas et al. improved upon an asymmetric cipher based on the matrix power function (see [18]) to avoid a successful discrete logarithm attack against the original version. Ramadan et al. published a survey of public key infrastructure (PKI)-based security for mobile systems (see [19]) covering aspects such as authentication, key agreement, and privacy.

Qiao et al. described a black-box traceable ciphertext-policy attribute-based encryption (CP-ABE, see [20,21]) that is scalable and efficient and, therefore, better suited for cryptographic cloud storage. Zhu et al. cryptanalyzed an image encryption algorithm based on a chaos s-box (see [22]), proposing an improved version with better security and performance. Park et al. described the use cases, challenges, and solutions involved in the application of blockchain-based security technologies to cloud computing (see [23]).

Chang et al. proposed password-authenticated key exchange and protected password change protocols that do not involve symmetric or asymmetric cryptosystems (see [24]), basing the security on the computational Diffie–Hellman assumption in the random oracle model. Nam et al. presented a provably secure three-party password-only authenticated key exchange protocol (see [25]) that can run in only two rounds of communication.

Regarding PBKDF and password hashing functions, the finalists to the Password Hashing Competition (*Argon2* [5], *Catena* [6], *Makwa* [8], *Lyra2* [9], and *Yescrypt* [10]) are highly relevant functions related to our proposed optimized PBKDF. Also, there is previous work by the authors in this field, including the original version of the proposal ([3]) and others (see [4,26]).

3. Description

In the following, we describe the parameters and elements, as well as the design, of the original PBKDF function and the proposed optimized versions. There are three variants of our proposal: the original version (*AESCTR-o*, published in [3]); the intermediate optimization step (*AESCTR-i*); and the final optimization (*AESCTR-f*). We use a pseudocode notation loosely based on the Go language to describe the initialization and output stages of the proposal.

3.1. Parameters

Our proposal and most PBKDF designs share a very similar set of parameters, mainly including the user password (*pass[]*) and random salt (*salt[]*) byte strings to be hashed, the length of the output hash to be generated (*plen*), and some kinds of cost parameters. Those PBKDFs that have been designed to slow down attackers employing GPGPU or specialized hardware usually involve two cost parameters: a time cost (*ptime*) and a memory cost (*pmem*) that, due to its nature, tends to influence the time cost as well. These parameters are the same as in the original version (see [3] for more details).

Most of the variables of the algorithm are unchanged as well; *M[]* is the main memory buffer that is parameterized by *plen* and *pmem*, while *out[]* constitutes the output hash of the function. Also, *M64[]* and *out64[]* are employed in the final optimization (*AESCTR-f*) to perform 64-bit native operations for performance reasons.

The algorithm employs SHA3-256 as a secure cryptographic hash function (see [27]) during the initial seeding phase, and AES-128 (see [14]) is used in CTR mode as a pseudo-random generator; both of these could be swapped for different, equivalent primitives were it necessary in the future.

3.2. Initialization

The initialization stage is unchanged from the original version of the algorithm (see [3]) and is reproduced here in Figure 1. We have added comments to detail the seeding and buffer initialization steps.

```
// Generate a 256-bit hash (seed) from the password and salt
fH := sha3.New256()
fH.Write(pass)
fH.Write(0)
fH.Write(salt)
seed := fH.Sum(nil)

// Build an AES-128-CTR instance from the seed value
blk := aes.NewCipher(seed[0:16])
fC := cipher.NewCTR(blk, seed[16:32])

// Allocate and fill output buffer
out := make([]byte, plen)
fC.XORKeyStream(out, out)

// Allocate and fill memory buffer
M := make([]byte, pmem*plen)
fC.XORKeyStream(M, M)
```

Figure 1. Initialization stage pseudocode (common to all variants).

3.3. Output

The intermediate optimization attempts to improve the performance of the original version (see [3]) by avoiding AES encryption steps inside the loops and just encrypting the *out[]* buffer as the last step. For this reason, there is a second inner loop that generates a new index so that a different row

from *M[]* can be processed into *out[]*. These modifications are shown in Figure 2 (pseudocode) and in Figure 3 (flow diagram).

```
for t := 0; t < ptime; t++ {
for m := 0; m < pmem; m++ {
i := (int(out[0:8])%pmem) * plen
for o := 0; o < len(out); o++ {
M[i+o] -= out[o]
}
i = (i*i) % pmem
for o := 0; o < len(out); o++ {
out[o] -= (M[i+o] ^ out[o])
}
}
}
fC.XORKeyStream(out, out)
```

Figure 2. Intermediate (*AESCTR-i*) output stage pseudocode.

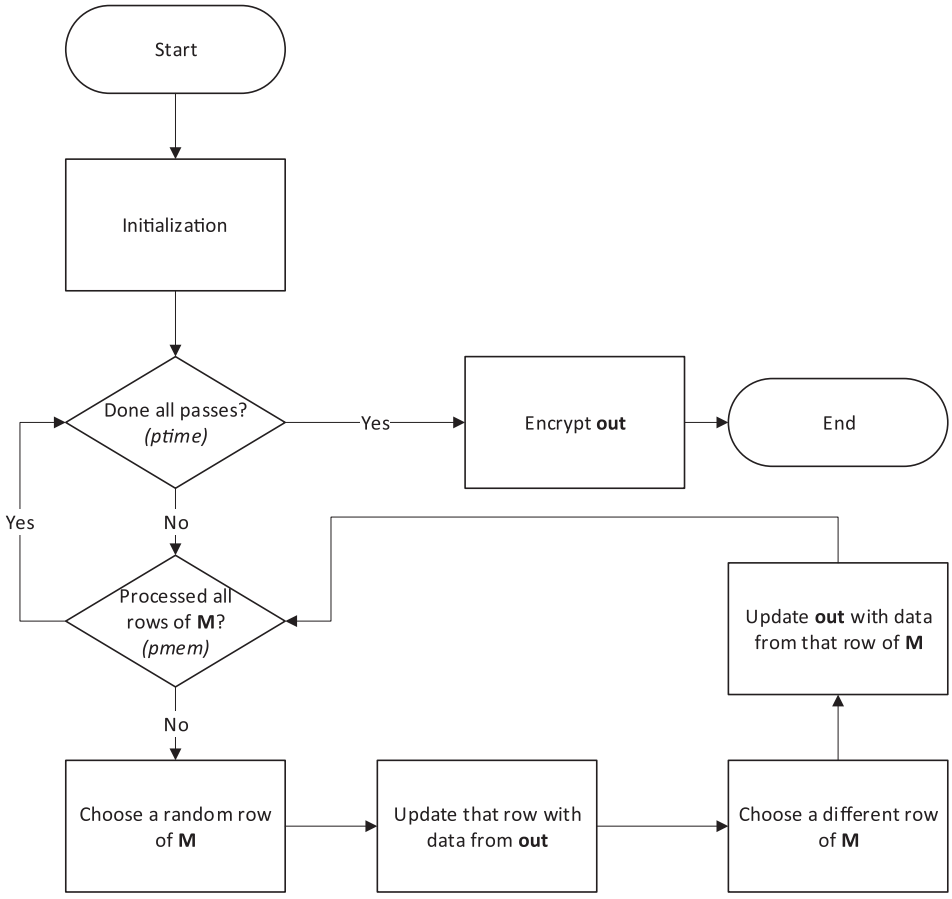

Figure 3. Flow diagram for the intermediate (*AESCTR-i*) output stage.

The final optimization further improves the performance by avoiding writing back to *M[]* and having a second inner loop. Also, memory access and operations are performed in 64-bit. The differences between the final and intermediate optimizations are shown in Figure 4 (pseudocode) and in Figure 5 (flow diagram).

```
for t := 0; t < ptime; t++ {
for m := 0; m < pmem; m++ {
i := (int(out[0:8])%pmem) * plen/8
for o := 0; o < len(out); o++ {
out64[o] -= (M64[i+o] ^ out64[o])
}
}
}
fC.XORKeyStream(out, out)
```

Figure 4. Final (*AESCTR-f*) output stage pseudocode.

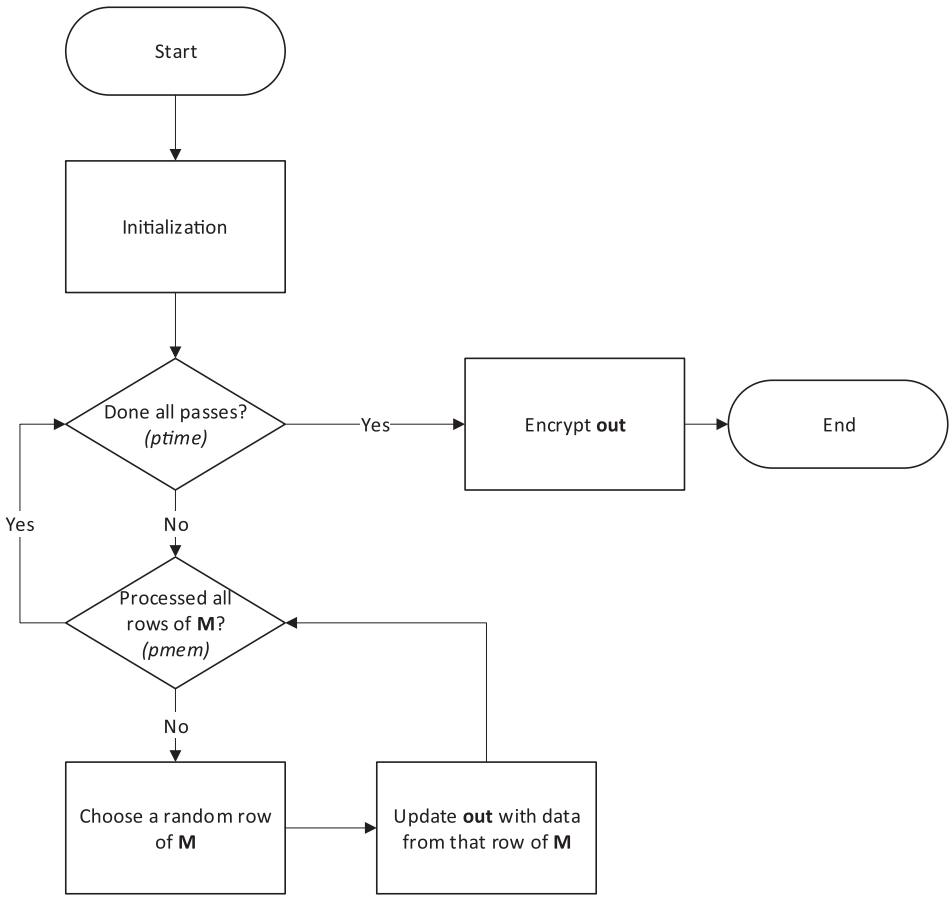

Figure 5. Flow diagram for the final (*AESCTR-f*) output stage.

4. Results

Regarding performance, we benchmarked in the following the optimized proposals, together with the original version ([3]), when modulating the time (*ptime*) and space (*pmem*) complexity parameters. The testing methodology is detailed in Section 6.

In Figure 6, we present the computational cost (execution time in logarithmic scale) of all three variants of the proposal as the *pmem* parameter modulation ranges from 2^8 to 2^{23} entries of 32 bytes (a memory usage varying from 8 KB to 256 MB) and with *ptime* = 1. We can see that the intermediate optimization is a significant improvement over the original function, but it is clearly overtaken by the final optimization, which is about 5 times faster than the original version.

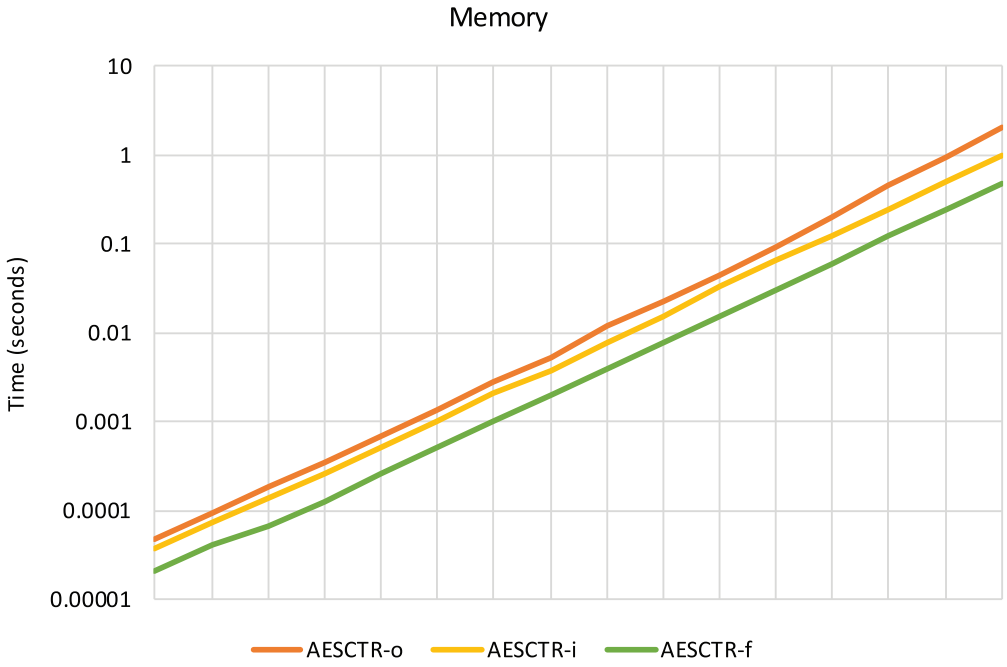

Figure 6. Performance while modulating the spatial cost parameter (*pmem*).

Figure 7 shows the behavior of the proposed optimizations when the time parameter is modulated. In this test, *ptime* values range from 1 to 2^{15} passes and memory usage (*pmem*) is kept constant at 2^8 entries of *plen* bytes (8 KB of memory). In this case, the difference in performance of the final optimization (*AESCTR-f*) is more pronounced than in the case of the memory parameter.

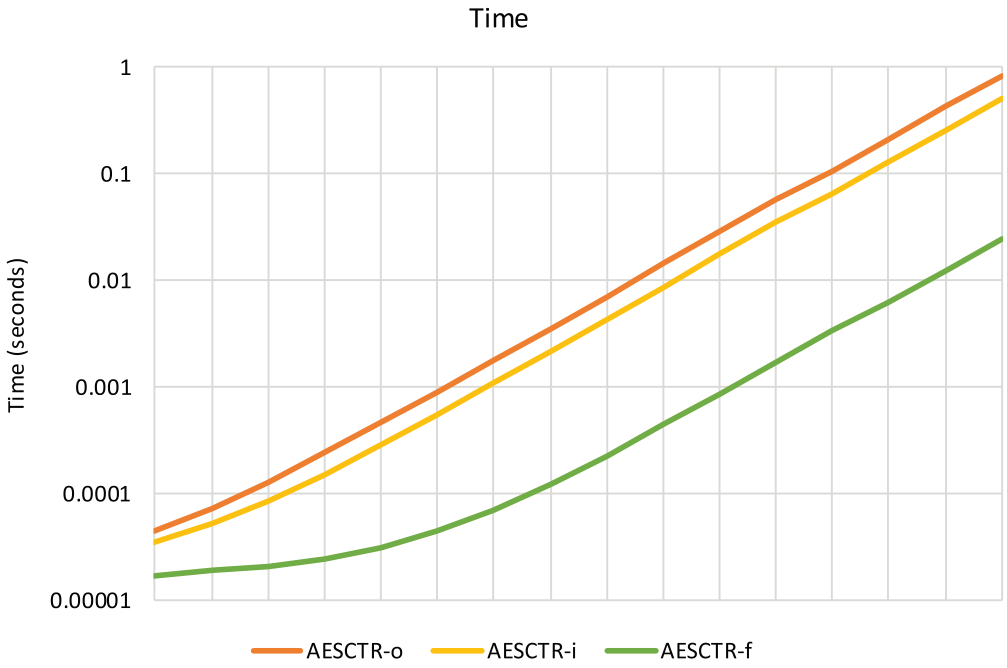

Figure 7. Performance while modulating the temporal cost parameter (*ptime*).

Figure 8 represents the execution time (in logarithmic scale) when both parameters, *pmem* and *ptime*, are simultaneously modulated in a double *for* loop. The outer loop is *pmem*, corresponding to the

number of entries in the main memory buffer, *M[]*, going from 2^8 to 2^{15}, and the inner loop corresponds to *ptime*, ranging from 1 to 2^7; the maximum amount of memory usage is 128 MB and occurs when pmem = 2^{15} and ptime = 2^7. The sawtooth shapes are expected in a double-loop arrangement. In this combined benchmark, the performance gain achieved with the final optimization is readily apparent.

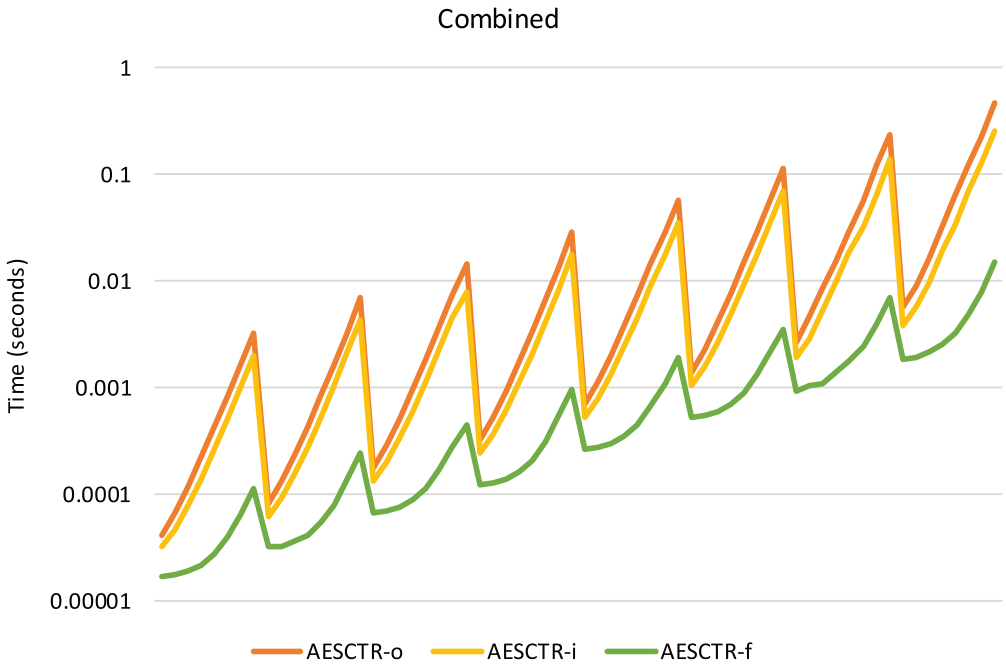

Figure 8. Performance while simultaneously modulating both *ptime* and *pmem* cost parameters.

5. Discussion

In the following, we discuss the security characteristics of our proposal and compare all three variants to Scrypt and Argon2 in terms of performance.

5.1. Comparison with Scrypt and Argon2

Scrypt (see [7]) is a PBKDF that was designed by Colin Percival in 2009. It has been employed for many services and applications, acting as a de facto standard in recent years. It has also been employed as a proof-of-work algorithm in some blockchain implementations.

Argon2 (see [5]) became the Password Hashing Competition (PHC) winner in 2015, overcoming finalists such as Catena [6], Lyra2 [9], Yescrypt [10], and Makwa [8]. It provides very interesting options, such as data-dependent or data-independent memory access, and is gaining traction in many new secure authentication implementations.

As shown in Figure 9, for an equal amount of memory, Scrypt is slower than all three variants of the proposed AES-CTR algorithm, while Argon2 is a very efficient algorithm, but the final optimization (*AESCTR-f*) is slightly faster than Argon2. Moreover, the *speedup* between our proposal and Argon2 increases with memory usage, as shown in Table 1.

It is interesting to note that while Argon2 has been implemented using Intel's SSE4 instruction set and our proposal takes advantage of the native AES instructions, Scrypt does not benefit directly from the hardware acceleration available in modern processors. More details regarding the testing methodology are included in Section 6.

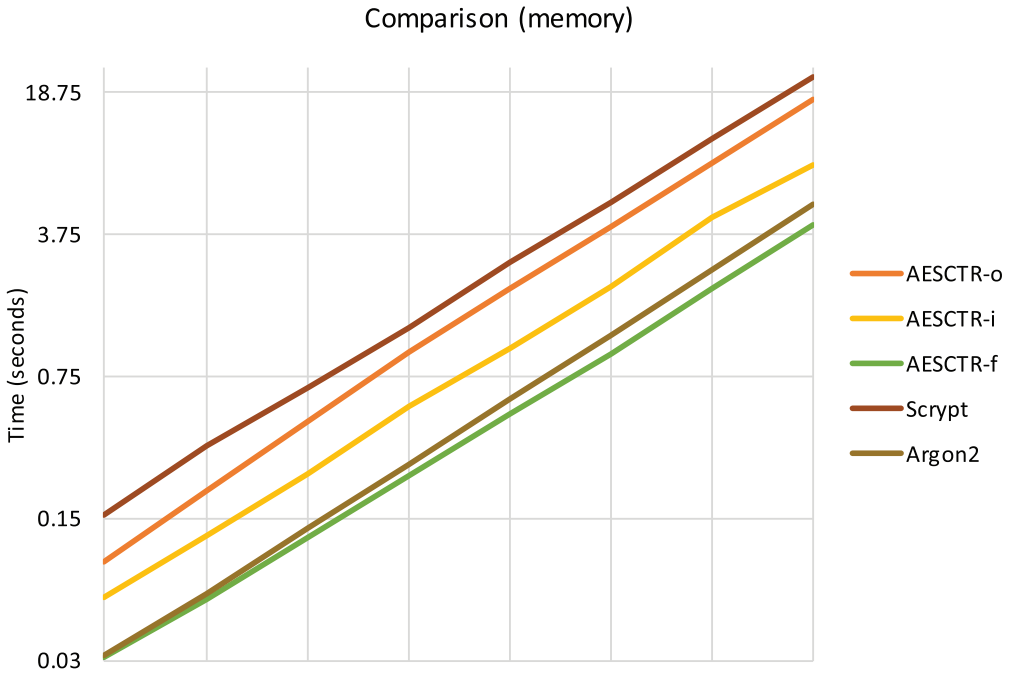

Figure 9. Comparison with Scrypt and Argon2.

Table 1. Speedup between the final optimization and Argon2.

pmem	AESCTR-f (s)	Argon2 (s)	Speedup
8	0.03	0.03	1.04
9	0.06	0.06	1.07
10	0.12	0.13	1.11
11	0.24	0.28	1.16
12	0.49	0.58	1.19
13	0.98	1.21	1.24
14	2.02	2.51	1.25
15	4.15	5.27	1.27

5.2. Security

This design is based on two different cryptographic primitives: a pseudo-random generator that provides the initial contents of the memory and output buffers, and a hash function that processes the user password and salt and produces a seed for this pseudo-random generator. We employed a symmetric cipher, AES-128 [14] in CTR mode, as the pseudo-random generator and SHA3-256 [27] as the hash function that provides the seed (128-bit key and initialization vector) for AES.

Both of these cryptographic primitives are well-known, independently tested current standards. Nevertheless, if they were to be deemed insecure in the future, they could be replaced by different, size-compatible, secure alternatives.

The proposed optimized design should be on par with the original version and these primitives at a minimum-security level of 128 bits against brute force attacks since the output of the PBKDF (in all three variants) comes directly from the output of the AES-128 symmetric cipher.

6. Methods

We employed the Go programming language (version 1.11.1, see [28]) for the implementation of all tested algorithms; Go is an excellent choice for cryptography testing since it is a very efficient, compiled language and includes most security standards and algorithms in its standard library. All benchmarks were run on the same computer, a desktop PC with an Intel i7 CPU (6950X, 3.5

GHz and with AES Native Instructions support) and 32 GB of RAM, running Microsoft Windows 10 (1803 release). The length for passwords, salts, and output was chosen as 32 bytes (256 bits), and all tests were run 100 times to avoid the interference from external processes as much as possible.

In the case of the comparison benchmarks, the official implementations available in the golang.org/x/crypto packages were used for Scrypt and Argon2 testing. It should be noted that this implementation of Argon2 supports hardware acceleration via SSE4 instructions and that the recommended parameters (three passes) and a single thread were used. To ensure fair testing, an equal amount of RAM usage was chosen for each algorithm in all comparison tests.

7. Conclusions

We optimized a previously published password-based key derivation function that employs the Advanced Encryption Standard (AES) in counter mode as a core primitive, proposing two new algorithms based on the original design: a more conservative optimization and a fully optimized one. The design philosophy is based on taking advantage of the custom AES instructions available on most modern processors that enable hardware support to defend against brute force password cracking attacks mounted on specialized hardware or general-purpose graphical processing units.

We have also analyzed the performance of all three variants in comparison with Scrypt and Argon2, which are the current industry standards in terms of password hashing functions. The final optimization version of the algorithm (*AESCTR-f*) is faster than Argon2 for an equal amount of memory usage, showing that AES can be an excellent candidate for the design of password hashing functions. Moreover, since the design is based primarily on AES in counter mode as a pseudo-random generator and the final output is directly encrypted with AES, we can establish that our proposal is equivalent, in terms of security, to this extensively analyzed encryption standard.

For future research, there are several possible interesting topics, such as server-side ROM, client-independent update, server relief, parallelism, or specialized implementations, among others.

Server-side ROM is an extra security measure that involves employing a very big random file on the server as part of the hashing process. In this way, an attacker would have to produce the same random file, and the large amount of memory required would further deter the use of specialized hardware. Our proposal can be adapted to accept such a file as part of the algorithm without compromising its performance or security.

Client-independent update is the capability of changing PBKDF parameters without the need for the user to reenter the password. This is a convenient feature in password authentication since the server can increase the security of the PBKDF as necessary but without any friction for the end users. This can be performed without modification to the proposed algorithm by multiple-step processing, but other optimized methods might be possible.

Server relief implies delegating part of the PBKDF computation to the end user so that the server is less impacted by computational requirements of password-authenticating a large number of users simultaneously. It is, in essence, a way of increasing the parallelism of the server and reducing the advantage that attackers might have by using general-purpose graphical processing units or other specialized hardware. This usually involves some kind of protocol in order to share the computational load between the server and the user node in a secure manner.

Multiple-thread parallelism can also be studied and incorporated by modifying the proposed final optimization (*AESCTR-f*). This can be useful in situations where server relief is not possible or when further parallelism is desired. Since Argon2 allows for multiple-thread parallelism, a comparison between the parallel performance scalability of the final optimization and Argon2 might be possible.

Specialized implementation on hardware platforms, such as general-purpose graphic processing units or field programmable gate arrays, could be very useful for further performance testing and optimization of the proposed PBKDF algorithm.

Author Contributions: All three authors contributed equally in the conceptualization, validation, research and writing of this paper.

Funding: Research partially supported by the Spanish Government under Project Grant TEC2014-54110-R (CASUS).

Conflicts of Interest: the authors declare no conflict of interest.

References

1. Hellman, M. A Cryptanalytic Time-memory Trade-off. *IEEE Trans. Inf. Theory* **2006**, *26*, 401–406, doi:10.1109/TIT.1980.1056220. [CrossRef]
2. Provos, N.; Mazieres, D. A Future-Adaptable Password Scheme. In Proceedings of the 1999 USENIX Annual Technical Conference, FREENIX Track, Berkeley, CA, USA, 23–26 August 1999; pp. 81–91.
3. Álvarez-Sánchez, R.; Andrade-Bazurto, A.; Santos-González, I.; Zamora-Gómez, A. AES-CTR as a Password-Hashing Function. In Proceedings of the International Joint Conference SOCO'17-CISIS'17-ICEUTE'17, León, Spain, 6–8 September 2017; Pérez García, H., Alfonso-Cendón, J., Sánchez González, L., Quintián, H., Corchado, E., Eds.; Springer International Publishing: Cham, Switzerland, 2018; pp. 610–617, ISBN 978-3-319-67180-2._59. [CrossRef]
4. Álvarez, R.; Zamora, A. Using Spritz as a Password-Based Key Derivation Function. In Proceedings of the International Joint Conference SOCO'16-CISIS'16-ICEUTE'16, San Sebastián, Spain, 19–21 October 2016.
5. Biryukov, A.; Dinu, D.; Khovratovich, D. Argon2: New Generation of Memory-Hard Functions for Password Hashing and Other Applications. In Proceedings of the IEEE 2016 IEEE European Symposium on Security and Privacy, Saarbrucken, Germany, 21–24 March 2016; pp. 292–302, ISBN 978-1-5090-1751-5. [CrossRef]
6. Forler, C.; Lucks, S.; Wenzel, J. The Catena Password-Scrambling Framework. 2015. Available online: https://password-hashing.net/submissions/specs/Catena-v5.pdf (accessed on 20 November 2018).
7. Percival, C. Stronger Key Derivation via Sequential Memory-Hard Functions. 2009. Available online: http://www.bsdcan.org/2009/schedule/attachments/87_scrypt.pdf (accessed on 20 November 2018).
8. Pornin, T. The Makwa Password Hashing Function. 2015. Available online: http://www.bolet.org/makwa/makwa-spec-20150422.pdf (accessed on 20 November 2018).
9. Simplício, M.A., Jr.; Almeida, L.C.; Andrade, E.R.; dos Santos, P.C.; Barreto, P.S. Lyra2: Password Hashing Scheme with improved security against time-memory trade-offs. *IEEE Trans. Comput.* **2016**, *65*, 3096–3108, doi:10.1109/TC.2016.2516011. [CrossRef]
10. Peslyak, A. yescrypt—A Password Hashing Competition Submission. 2015. Available online: https://password-hashing.net/submissions/specs/yescrypt-v2.pdf (accessed on 20 November 2018).
11. Moriarty, K.; Kaliski, B.; Rusch, A. *PKCS# 5: Password-Based Cryptography Specification Version 2.1*; Technical Report; IETF: Fremont, CA, USA, 2017; doi:10.17487/RFC8018.
12. Biryukov, A.; Dinu, D.; Khovratovich, D. Fast and Tradeoff-Resilient Memory-Hard Functions for Cryptocurrencies and Password Hashing. *IACR Cryptol. ePrint Arch.* **2015**, *2015*, 430:1–430:15.
13. Ferguson, N.; Schneier, B.; Kohno, T. *Cryptography Engineering: Design Principles and Practical Applications*; Wiley Publishing: Hoboken, NJ, USA, 2010, ISBN 978-0-470-47424-2.
14. Daemen, J.; Rijmen, V. AES Proposal: Rijndael. 1999. Available online: http://www.cs.miami.edu/home/burt/learning/Csc688.012/rijndael/rijndael_doc_V2.pdf (accessed on 20 November 2018).
15. Keller, S.S. NIST-Recommended Random Number Generator Based on ANSI X9.31 Appendix A.2.4 Using the 3-Key Triple DES and AES Algorithms. 2005. Available online: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.210.70&rep=rep1&type=pdf (accessed on 20 November 2018).
16. Chang, Y.F.; Tai, W.L.; Hsu, M.H. A Secure Mobility Network Authentication Scheme Ensuring User Anonymity. *Symmetry* **2017**, *9*, 307, doi:10.3390/sym9120307. [CrossRef]
17. Hung, Y.H.; Tseng, Y.M.; Huang, S.S. Lattice-Based Revocable Certificateless Signature. *Symmetry* **2017**, *9*, 242, doi:10.3390/sym9100242. [CrossRef]
18. Sakalauskas, E.; Mihalkovich, A.; Venčkauskas, A. Improved Asymmetric Cipher Based on Matrix Power Function with Provable Security. *Symmetry* **2017**, *9*, 9, doi:10.3390/sym9010009. [CrossRef]
19. Ramadan, M.; Du, G.; Li, F.; Xu, C. A Survey of Public Key Infrastructure-Based Security for Mobile Communication Systems. *Symmetry* **2016**, *8*, 85, doi:10.3390/sym8090085. [CrossRef]
20. Qiao, H.; Ba, H.; Zhou, H.; Wang, Z.; Ren, J.; Hu, Y. Practical, Provably Secure, and Black-Box Traceable CP-ABE for Cryptographic Cloud Storage. *Symmetry* **2018**, *10*, 482, doi:10.3390/sym10100482. [CrossRef]

21. Ba, H.; Zhou, H.; Mei, S.; Qiao, H.; Hong, T.; Wang, Z.; Ren, J. Astrape: An Efficient Concurrent Cloud Attestation with Ciphertext-Policy Attribute-Based Encryption. *Symmetry* **2018**, *10*, 425, doi:10.3390/sym10100425. [CrossRef]
22. Zhu, C.; Wang, G.; Sun, K. Cryptanalysis and Improvement on an Image Encryption Algorithm Design Using a Novel Chaos Based S-Box. *Symmetry* **2018**, *10*, 399, doi:10.3390/sym10090399. [CrossRef]
23. Park, J.H.; Park, J.H. Blockchain Security in Cloud Computing: Use Cases, Challenges, and Solutions. *Symmetry* **2017**, *9*, 164, doi:10.3390/sym9080164. [CrossRef]
24. Chang, T.Y.; Hwang, M.S.; Yang, C.C. Password Authenticated Key Exchange and Protected Password Change Protocols. *Symmetry* **2017**, *9*, 134, doi:10.3390/sym9080134. [CrossRef]
25. Nam, J.; Choo, K.K.R.; Han, S.; Paik, J.; Won, D. Two-Round Password-Only Authenticated Key Exchange in the Three-Party Setting. *Symmetry* **2015**, *7*, 105–124, doi:10.3390/sym7010105. [CrossRef]
26. Alvarez, R.; Caballero-Gil, C.; Santonja, J.; Zamora, A. Algorithms for Lightweight Key Exchange. *Sensors* **2017**, *17*, 1517, doi:10.3390/s17071517. [CrossRef] [PubMed]
27. Bertoni, G.; Daemen, J.; Peeters, M.; Van Assche, G. Cryptographic Sponge Functions. 2011. Available online: https://keccak.team/files/CSF-0.1.pdf (accessed on 20 November 2018).
28. The Go Programming Language. Available online: http://www.golang.org (accessed on 20 November 2018).

© 2018 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (http://creativecommons.org/licenses/by/4.0/).