CERIAS Tech Report 2023-5

Closing the Gap: Leveraging AES-NI to Balance Adversarial Advantage and Honest User Performance in Argon2i

by Nicholas Harrell and Nathaniel Krakauer
Center for Education and Research
Information Assurance and Security
Purdue University, West Lafayette, IN 47907-2086

Closing the Gap: Leveraging AES-NI to Balance Adversarial Advantage and Honest User Performance in Argon2i

Nicholas Harrell, *Purdue University,* Nathaniel Krakauer, *Purdue University,*

***Abstract*—The challenge of providing data privacy and integrity while maintaining efficient performance for honest users is a persistent concern in cryptography. Attackers exploit advances in parallel hardware and custom circuit hardware to gain an advantage over regular users. One such method is the use of Application-Specific Integrated Circuits (ASICs) to optimize key derivation function (KDF) algorithms, giving adversaries a significant advantage in password guessing and recovery attacks. Other examples include using graphical processing units (GPUs) and field programmable gate arrays (FPGAs).**

We propose a focused approach to close the gap between adversarial advantage and honest user performance by leveraging the hardware optimization AES-NI (Advanced Encryption Standard New Instructions). AES-NI is widely available in modern x86 architecture microprocessors. Honest users can negate the adversary advantage by diminishing the utility of their computational power. We explore the impact of AES-NI on the Argon2i KDF algorithm, a widely-used and recommended password hashing function.

Through our analysis, we demonstrate the effectiveness of incorporating AES-NI in reducing the advantage gained by attackers using ASICs. We also discuss the security and performance trade-offs to provide guidelines for practical implementation in deployed cryptosystems.

***Index Terms*—cryptography, hardware optimization, argon2i, key derivative function, kdf**

I. Introduction

Despite numerous efforts to transition away from password-based authentication, it remains the most prevalent form of authentication [1]. The persistence of password-based systems, even in the presence of multi-factor authentication and various security protocols has contributed to security breaches that have exposed billions of passwords to offline password cracking over the past decade [2]. Techniques such as balloon attacks, multi-instance security, auxiliary input and bit-fixing attacks, password guessing, and password unrecoverability have been employed to demonstrate the limits of password cracking.

The research goal of this paper is to demonstrate the impact of enabling honest users to compute hashes more efficiently, thereby reducing an attacker's advantage. As honest users can compute hashes more quickly, they can execute more iterations of a hash function without incurring additional costs, significantly increasing the attacker's workload. Furthermore, using unique random salts for each password in a database would decrease the attacker's likelihood of successfully cracking passwords. This heightened level of assurance constitutes a highly sought-after security guarantee. The primary motivation for this research question in [2] illustrates that even when using ASICs, which are the most effective manufacturing process to implement algorithms in hardware, Memory Hard Functions (MHFs) still consume significant amounts of RAM when deployed on an ASIC architecture. This mathematical property of MHFs is a desirable security characteristic. However, many Key Derivative Functions (KDFs), such as Argon2i, bcrypt, scrypt, DRSample, and the currently NIST-approved algorithms, KDF and PBKDF2, do not allow average users to take advantage of modern hardware designs. Although modern software optimizations have improved the speed of hash calculations, the attacker is assumed to possess a memory advantage. Consequently, users expend more resources due to less efficient hash calculations.

II. Previous Work

Because passwords are still the most used form of authentication, there is a wide array of prior works dedicated to advancing the theoretical understanding of generating intractable hashes no matter the computational efficiency of the adversary. This includes specialized ASIC and GPU hardware, that an adversary has available. The problem formulation is where an attacker wants to solve multiple hashed passwords using preprocessed methods, such as rainbow tables, to solve the hashes faster. Even though, theoretically, the hash functions are hard problems, everyday users of computers frequently choose predictable low-entropy passwords, which diminish the usefulness of these hard functions [3].

A. Preprocessing Attack

The main avenue for an adversary is to trade space for time. Hellman introduced the concept of time memory trade-off to cryptoanalyze any N key cryptosystem in $N^{2/3}$ operations with $N^{2/3}$ words of memory on average. The adversary precomputes a table of plaintext encrypted by N keys and stored in the table. The attack introduced by Hellman successfully compromises DES encryption but not block chaining encryption schemes that use a random initialization vector (IV) to prevent guessing the first block of text correctly [4].

B. Mitigations

One way to compensate for low entropy passwords is to add iterations to the hashing process. [5] outlines in specification PKCS#5, that using iteration can make it more time inefficient

for an adversary to evaluate the hash function in the forward direction. In this standard, the authors recommend using 1000 iterations to increase the cost to an adversary but limit the impact from an honest broker deriving the individual keys [6].

[7] introduces a novel system implementation that allows for password security to keep pace with increased hardware performance using iteration to make a forward pass of the hashing process intractable. The paper's motivation is that password length and entropy do not scale with computing power. The fact that most passwords are low entropy does not help this fact. Their algorithm, bcrypt, creates 128-bit salt to encrypt 192-bit values. Bcrypt uses their eksblowfish algorithm, a cost-parameterizable and salted block cipher used to create resistant ciphertext against the provided keys [8]. The changing intermediate arrays in the eksblowfish algorithm prevent the possibility of new optimizations making the ciphertext of the key crackable and limits the utility of encoding the Feistel network used in the block cipher being encoded and run in hardware. Eksblowfish is used to store the key of the passwords on disk securely. Bcrypt's 128-bit salt makes precomputing dictionary attacks intractable. This also includes if the attacker already knows the salt. Bit slicing attacks are also intractable since bcrypt S-boxes change while the algorithm runs [9].

Another avenue in the literature is to make computing memory-hard. The prior works discussed so far have a major flaw. Each is susceptible to attackers implementing custom hardware to crack the key derivation functions. In response, scrypt was developed to mitigate parallel attacks [10]. The goal is to make the attacker use up an intractable amount of memory for the duration of the password cracking. The key security assumption of scrypt is that BlockMix with Salsa20 on an eight-count iteration has no optimizations to allow iterations to be hacked more easily than using a random oracle [11].

The authors of [12] contribute new theoretical tools to prove the lower bounds for functions that run in parallel that provide the hard guarantees for scrypt. The key advancement is applying the pebbling reduction to memory hard functions that underpin the security of key derivative functions. Pebbling on graphs is a two-player game. The game starts when player 1 picks any vertex as the root vertex and sets pebbles on any vertex besides the root. Player two can decide to make a move by picking two pebbles at a vertex and can win if player two places a pebble at the root. The pebbling number is the least number of pebbles where player two can win no matter how the pebbles are placed and any vertex is root [13]. This reduction supports the lower bound complexity of memory-hard functions (MHF) on parallel computation tasks such as GPUs. The pebble path is a directed acyclic graph (DAG), the longest path of pebbles throughout the graph. This path represents the space needed to store enough information to crack the MHF and is used to represent the lower bound of a parallel attack. The authors expand the use of pebbling reductions from only sequential computation models to similar computation models by modifying the pebbling game for player 2 to move multiple pebbles during their turn. Using this new game, the paper proves a pebbling reduction from the game to MHFs to prove security from a parallel random oracle model adversary [14]. This new theoretical paradigm provides stronger security guarantees than the previous works discussed above since this is the first theory tool to reason about attackers use parallel hardware. The theoretical assumption in prior works used a sequential computer processing model. The rest of the paper will motivate the extension of efficiently using Argon2, whose theoretical security assumption is based on the parallel lower bounds proof using pebbling reductions.

III. Our Contribution: Hardware and Software Optimization

This paper addresses the threat posed by an adversary possessing considerable computational resources for password cracking, with a focus on the Argon2i Key Derivation Function (KDF). The ultimate aim is to propose a method for mitigating the adversarial advantage to improve user security. We demonstrate that, given comparable memory and processing capabilities, the performance of an AES-NI implementation is similar to an FPGA or ASIC implementation of the Advanced Encryption Standard (AES) algorithm. Since little prior work in the literature supports this hypothesis, this paper seeks to examine known implementations of AES on FPGAs and ASICs to establish baseline measurements.

A. ASIC and FPGA

Efficient cryptographic processing relies heavily on the algorithm and the associated operations being executed. In the context of hardware optimization, features such as pipelining, iterative looping, and loop unrolling are important, but still susceptible to pipeline stalls from structural dependencies. While this paper does not delve into these aspects in-depth, it explores some associated trade-offs. Pipelining is particularly advantageous in the context of iterative processes. As [15] elucidates, the number of rounds K that can be unrolled without surpassing the available circuit area is desirable for hardware optimization. This necessitates the consideration of ciphers without feedback options, as their inclusion would constrain the circuit, thereby reducing the available area. [15] demonstrates the suitability of AES for FPGA implementation, highlighting the iterative nature of the AES round as an ideal cipher for hardware integration. The intrinsic functions of AES, such as MixColumns and SubBytes, possess mutually inverse properties that allow for the parallel execution of multiple rounds in forward and reverse directions. AES's design facilitates the exploitation of techniques that leverage various iterations.

The authors in [16] investigate optimization by examining circuit area employing the term "Gate Equivalent [GE]." The study uses a baseline throughput of 50 MHz to illustrate that increasing the circuit area leads to a reduction in clock cycles and an increase in throughput. Consequently, the only countermeasure available involves deploying an algorithm that demands substantial memory to consume the entire circuit area. In comparing ASICs and FPGAs, ASICs have historically exhibited a frequency advantage on the order of four to five. Typically, ASICs possess a processing advantage due to the static nature of their integrated circuits and the utilization of

hard cores [17]. Moreover, the cost of FPGAs and ASICs must be considered, as the significant expense could deter the average attacker.

B. AES-NI

We analyze heterogeneous systems encompassing a central processing unit (CPU) and a graphics processing unit (GPU) to execute parallel processing tasks more effectively. Contemporary designs often support multi-threading or hyper-threading features, as evidenced by the prevalence of cryptographic implementations. Generally, web servers employ a Key Derivative Function (KDF) for password-hashing processes.

A simplified explanation of modern-day password hashing involves a server generating a unique salt for each hash. This salt is known only to the server and constitutes a crucial aspect of the KDF implementation. Ideally, the server stores the salt in a location that minimizes the likelihood of disclosure. On a Linux system the salt is stored in the */etc/passwd* file. We assume an ideal server for our experiment setup. Consequently, even if a user's password is exposed and an adversary knows the hashing algorithm, the attacker must spend time t to crack the salt before utilizing the password maliciously. Furthermore, suppose the server generates a random salt for each password. In that case, the time required for an adversary to crack all passwords in the database increases to t^k, where k represents the total number of passwords. [5] refers to this phenomenon as multi-instance security.

The significance of this relationship stems from the fact that not every company or user can afford application-specific integrated circuit (ASIC) hardware or field-programmable gate arrays (FPGAs) for KDF processing. The time t required for an honest user to process k passwords should correlate with the adversary's ability to crack passwords. This relationship is vital to security from a performance perspective, so hardware optimizations provide the most cost-effective solution. Future research may explore additional hardware optimizations for cryptographic processes.

The AES-NI extension comprises six instructions that optimize AES implementations in software [18]. Modern systems generally enable cryptographic hardware optimizations by default, suggesting that incorporating encryption methods that leverage these optimizations could enhance the honest user's performance.

Users have alternative options for optimizing cryptographic schemes. One might consider implementing these on GPUs, as most modern heterogeneous systems possess GPUs. Although powerful GPUs are generally less expensive than ASICs and FPGAs in terms of cost, [19] compares the performance of both in various settings, such as multiple GPUs. While [19] demonstrates competitiveness with GPUs, AES-NI outperforms the GPU in nearly all settings with multiple cores and threads when multiple GPUs are used. From a cost perspective, AES-NI emerges as the superior candidate, as the expense of multiple GPUs is not justified when similar or better performance can be achieved with pre-existing hardware optimizations.

IV. Experimental Setup

The design of this experiment aims to address several critical aspects related to the performance of AES_NI. Firstly, we seek to analyze the performance of AES_NI in a general-purpose register setting, which emulates systems constrained to AES_NI optimization only. Secondly, we aim to evaluate an optimized platform capable of harnessing additional hardware instruction sets and larger registers.

To provide a meaningful comparison, we run the Argon2i baseline algorithm in an optimized manner to assess the performance differences between the AES_NI variants and Argon2i. By utilizing general registers, we determine if AES_NI performance is comparable to optimized Argon2i to elucidate the potential benefits of hardware optimizations.

A. Measures

In evaluating the performance of an essential derivation function (KDF), it is crucial to consider various factors, particularly those relating to the parallel execution of subroutines within the KDF's structure. To conduct a comprehensive analysis, we examine the performance of the KDF over 1 to 14 thread counts. This range is expected to provide sufficient evidence for identifying the critical performance differences.

Another critical aspect to consider is memory usage. Memory-hard KDFs, such as Argon2i, rely on consuming large amounts of memory to increase the computational complexity of hash calculations. Given that recent studies on Argon2i have utilized 4 gigabytes of memory, we increase memory to 8 gigabytes to reflect the memory capacities of modern computing systems and offer readers additional insight into performance at higher memory levels.

We will employ three primary metrics to assess the KDF's performance: millicycles (Mcycles), computations per byte (cpb), and runtime, representing the total time required for the algorithm to complete its calculations. For each parameter change, we will calculate the average values over three iterations to ensure the reliability and accuracy of our findings. Future studies will demonstrate the overall power consumption of each algorithm.

B. Constraints

The deliverable of this experiment had a time constraint of two months, which required us to scope down the experiment to a more manageable experiment. However, the authors are confident the results give researchers insights into the potential for pre-embedded hardware optimizations.

C. Limitations

This experiment only uses one personal computer under a specific configuration. We attempted to suppress as many background processes and services will allow. We make the assumption that only dedicating half of the available memory to the experiment and running several iterations will eliminate concerns for extraneous factors.

D. Experimental Prerequisites

We used a Windows 11 computer to perform the experiments. Given the memory-intensive nature of Windows 11, where background processes can utilize up to 50% of available memory due to scheduled operations, a minimum of 16 GB of memory is required. This allocation ensures that the algorithms can efficiently utilize up to 8 GB of memory for computational purposes.

E. Hardware

- AMD Ryzen 9 5900HS Personal Computer
- 8 Cores, 16 Threads (14 threads utilized for the experiment)
- 8 x 32 KBytes L1 Data Cache, 8 x 32 KBytes L1 Instruction Cache
- 8 x 512 KBytes L2 Cache
- 16 MBytes L3 Cache
- 16 GBytes DDR4 DRAM (8 GBytes allocated for the experiment)
- Zen 3 processors with 3 GHz to 4.6 GHz (turbo) clock speed

F. Software

- Argon2i bench routine
- Three algorithm variants:
  1) Argon2i AVX2 with 256 registers (baseline)
  2) AES_NI General Register (GR)
  3) AES_NI AVX2

V. Results

A. Runtime

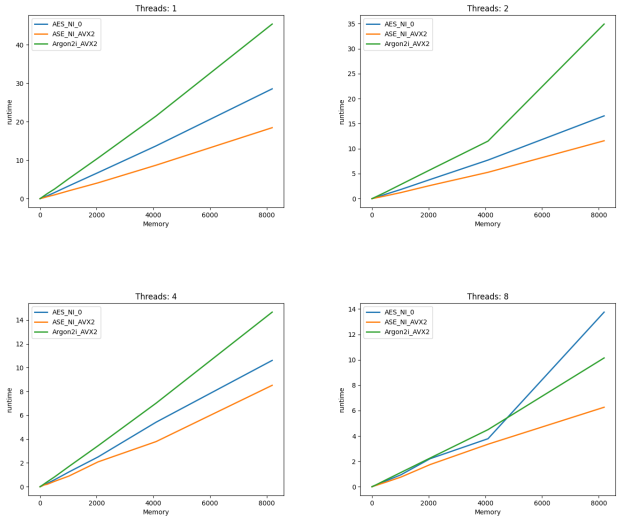

Fig. 1. runtime/Memory for 1,2,4,8 Threads

Our investigation's results indicate that the AES_NI variants, encompassing both the AVX2 and General Purpose Register (GPR) derivatives, exhibited superior performance in comparison to the Argon2i AVX2 variant. Among the assessed derivatives, the AES_NI AVX2 variant emerged as the most efficient, in accordance with our hypothesized outcomes delineated in prior sections of this manuscript. This result can be ascribed to the inherent efficiencies of the AES_NI AVX2 variant, which leverages the benefits of dedicated hardware instructions and optimized performance.

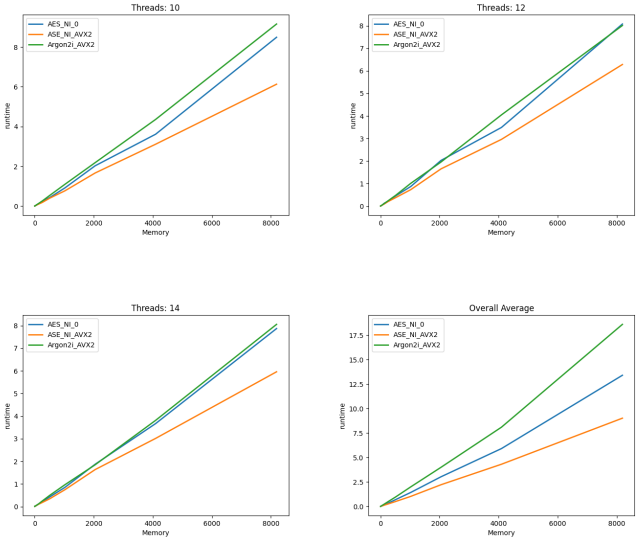

Fig. 2. runtime/Memory for 10,12,14,Overall Threads

An observed anomaly in the GPR AES_NI variant was a marked increase in runtime beyond 4GB. Although rerunning the test could potentially mitigate this issue, we opted to retain it in order to demonstrate how the utilization of GPR could potentially compromise performance quality. This irregularity may be attributable to cache misses or pipeline stalls, signifying resource competition. By examining the consistent linear nature of the other two variants, we highlight the advantages of dedicated hardware, which include the delivery of consistent performance. While this investigation does not provide a direct comparison with ASICs or FPGAs, it does illustrate the benefits of dedicated circuits.

It is worth noting that, unlike AES_NI, Argon2i does not employ a lookup table. The additional cost associated with utilizing a lookup table does not significantly impact performance, as the table is stored locally within the processor. Furthermore, we substituted two multiplicative and additive operators from Argon2i's G function with an entire round of AES_encrypt, which consists of SubBytes, MixColumns, ShiftRows, and AddRoundKey. This introduces a considerable number of supplementary operations. Despite the added complexity, the AES_NI variant outperforms the optimized Argon2i, with the AES_NI AVX2 version surpassing the Argon2i AVX2 by a factor of two when utilizing 8GB of memory.

B. CPB

Upon examining cycles per byte (cpb), an unexpected observation arose when analyzing the AES_NI General Pur-

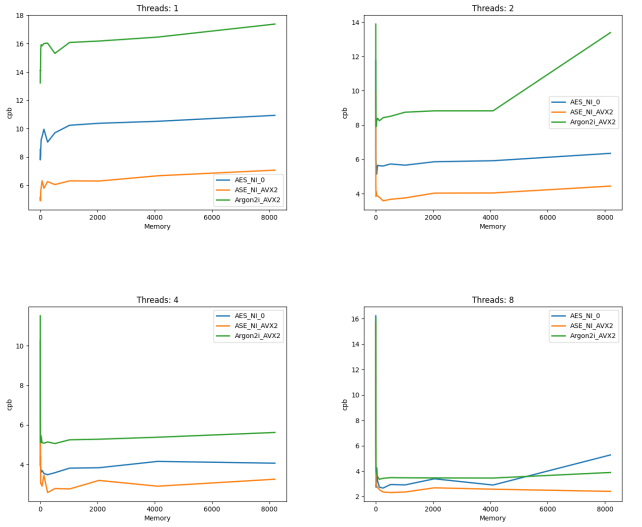

Fig. 3. CPB/Memory for 1,2,4,8 Threads

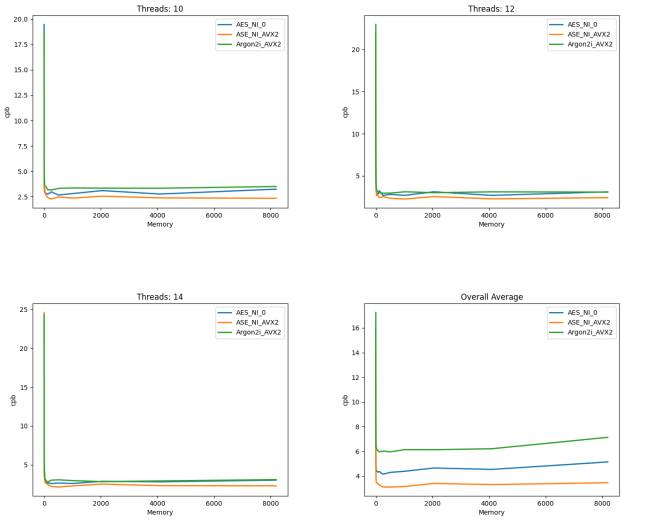

Fig. 4. CPB/Memory for 10,12,14,Overall Threads

pose Register (GPR) variant. Evidently, after reaching 4 GB at 8 threads, AES_NI begins to consume more cpbs. We hypothesize that this phenomenon may be ascribed to cache misses resulting from contention between Argon2i's memory fill function and the AES_NI instructions. Our reasoning is that the AES_NI intrinsic function necessitates access to the L1 cache to execute lookups in the subbytes operations, while Argon2i's memory fill operation concurrently requires access to local resources. As memory saturation intensifies at the 4 GB threshold, a heightened frequency of memory lookup operations transpires, leading to an escalation in cpbs. Conversely, the AVX2 variants capitalize on larger vectorized registers, enabling the functions to operate more efficiently through unrolling techniques and an augmented pipeline for data processing. As these instruction sets consistently function locally, resource competition is attenuated.

VI. Conclusion

In summary, the integration of AES_NI into Argon2i as a permutation within the G function serves to enhance the overall performance of Argon2i with both GPR and AVX2 instruction sets. This improvement provides evidence of potential for bridging the gap between adversaries and honest users concerning Key Derivative Functions.

References

[1] P. Khatiwada and B. Yang, "An access control and authentication scheme for secure data sharing in the decentralized cloud storage system," in *2022 5th Conference on Cloud and Internet of Things (CIoT)*. IEEE, 2022, pp. 137–144.

[2] W. Bai, J. Blocki, and M. H. Ameri, "Cost-asymmetric memory hard password hashing," in *Security and Cryptography for Networks: 13th International Conference, SCN 2022, Amalfi (SA), Italy, September 12–14, 2022, Proceedings*. Springer, 2022, pp. 21–44.

[3] S. Pearman, S. A. Zhang, L. Bauer, N. Christin, and L. F. Cranor, "Why people (don't) use password managers effectively," in *Fifteenth Symposium on Usable Privacy and Security (SOUPS 2019)*, 2019, pp. 319–338.

[4] E. G. AbdAllah, Y. R. Kuang, and C. Huang, "Advanced encryption standard new instructions (aes-ni) analysis: Security, performance, and power consumption," in *Proceedings of the 2020 12th International Conference on Computer and Automation Engineering*, ser. ICCAE 2020. New York, NY, USA: Association for Computing Machinery, 2020, p. 167–172. [Online]. Available: https://doi.org/10.1145/3384613.3384648

[5] P. Farshim and S. Tessaro, "Password hashing and preprocessing," in *Advances in Cryptology–EUROCRYPT 2021: 40th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, October 17–21, 2021, Proceedings, Part II*. Springer, 2021, pp. 64–91.

[6] B. Kaliski, "Rfc2898: Pkcs 5: Password-based cryptography specification version 2.0," USA, 2000.

[7] N. Provos and D. Mazieres, "A future-adaptable password scheme." in *USENIX Annual Technical Conference, FREENIX Track*, vol. 1999, 1999, pp. 81–91.

[8] L. Ertaul, M. Kaur, and V. A. K. R. Gudise, "Implementation and performance analysis of pbkdf2, bcrypt, scrypt algorithms," in *Proceedings of the international conference on wireless networks (ICWN)*. The Steering Committee of The World Congress in Computer Science, Computer . . . , 2016, p. 66.

[9] N. Provos and D. Mazières, "A Future-Adaptable password scheme," in *1999 USENIX Annual Technical Conference (USENIX ATC 99)*. Monterey, CA: USENIX Association, June 1999. [Online]. Available: https://www.usenix.org/conference/1999-usenix-annual-technical-conference/future-adaptable-password-scheme

[10] A. Srinivasan, A. Nguyen, and R. Tarlecki, "Stump-stalling offline password attacks using pre-hash manipulations," in *2015 IEEE 21st International Conference on Parallel and Distributed Systems (ICPADS)*. IEEE, 2015, pp. 306–313.

[11] C. Percival and S. Josefsson, "The scrypt Password-Based Key Derivation Function," RFC 7914, Aug. 2016. [Online]. Available: https://www.rfc-editor.org/info/rfc7914

[12] J. Alwen, B. Chen, K. Pietrzak, L. Reyzin, and S. Tessaro, "Scrypt is maximally memory-hard," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2017, pp. 33–62.

[13] J. Asplund, G. Hurlbert, and F. Kenter, "Pebbling on graph products and other binary graph constructions," 2018.

[14] J. Alwen and V. Serbinenko, "High parallel complexity graphs and memory-hard functions," in *Proceedings of the Forty-Seventh Annual ACM Symposium on Theory of Computing*, ser. STOC '15. New York, NY, USA: Association for Computing Machinery, 2015, p. 595–603. [Online]. Available: https://doi.org/10.1145/2746539.2746622

[15] K. Gaj and P. Chodowiec, "Fpga and asic implementations of aes," *Cryptographic engineering*, pp. 235–294, 2009.

[16] N. Pramstaller, S. Mangard, S. Dominikus, and J. Wolkerstorfer, "Efficient aes implementations on asics and fpgas," in *Advanced Encryption Standard–AES: 4th International Conference, AES 2004, Bonn, Germany, May 10-12, 2004, Revised Selected and Invited Papers 4*. Springer, 2005, pp. 98–112.

[17] T. Güneysu, "Utilizing hard cores of modern fpga devices for high-performance cryptography," *Journal of Cryptographic Engineering*, vol. 1, pp. 37–55, 2011.

[18] G. Hofemeier and R. Chesebrough, "Introduction to intel aes-ni and intel secure key instructions," *Intel, White Paper*, vol. 62, 2012.

[19] V. Sanz, A. Pousa, M. Naiouf, and A. De Giusti, "Performance analysis of aes on cpu-gpu heterogeneous systems," in *Cloud Computing, Big Data & Emerging Topics: 10th Conference, JCC-BD&ET 2022, La Plata, Argentina, June 28–30, 2022, Proceedings*. Springer, 2022, pp. 31–42.