IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 23 June 2024, accepted 8 July 2024, date of publication 11 July 2024, date of current version 22 July 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3426605

RESEARCH ARTICLE

Efficient Optimization of MS Office 2013+ Password Cracking and PBKDF2-HMAC-SHA2 on GPUs

DONGCHEON KIM, (Student Member, IEEE), AND SEOG CHUNG SEO, (Member, IEEE)
Department of Financial Information Security, Kookmin University, Seoul 02707, South Korea

Corresponding author: Seog Chung Seo (scseo@kookmin.ac.kr)

This work was supported by the National Research Foundation of Korea (NRF) grant funded by Korean Government [Ministry of Science and Information & Communications Technology (MSIT)] under Grant 2022R1C1C1013368.

ABSTRACT The increasing sensitivity of personal information has led to the widespread adoption of encryption technology for unstructured files and various applications. However, this presents significant challenges for law enforcement agencies tasked with gathering evidence for crime investigations and prosecutions. The use of encryption to impede forensic efforts poses difficulties, as passwords to encrypted files cannot be easily obtained. Consequently, there is a pressing need to develop high-speed techniques for cracking encrypted files to access critical evidence. To address this challenge, our research focused on optimizing the PBKDF2-HMAC-SHA2 algorithm, commonly used for password protection, and enhancing methods for cracking MS Office 2013+ files, a widely used office suite. We implemented optimizations at both the algorithmic and architectural levels. Algorithmically, we concentrated on reducing memory accesses and eliminating redundant computations within SHA2 and HMAC. Architecturally, we optimized memory access patterns, utilized specialized memory, and improved thread placement to enhance throughput performance. As a result, we achieved significant performance improvements. For PBKDF2-HMAC-SHA2, we saw a 29% improvement on the RTX3090 and a 33% improvement on the RTX4090 compared to Hashcat, the most popular open-source password recovery tool. For cracking MS Office 2013+ files, we achieved performance gains of up to 15% on the RTX3090 and up to 9% on the RTX4090 compared to commercial software like Passware and Elcomsoft. Since our software is open source, it can be used to accelerate password cracking efforts involving HMAC and hash functions. Moreover, its architecture can serve as a reference for developing parallel password-cracking tools.

INDEX TERMS HMAC, SHA2, PBKDF2, MS Office 2013+, password cracking, efficient implementation, software optimization, parallel computing, GPU, CUDA.

I. INTRODUCTION

With the recent advancements in data privacy laws, the application of encryption to personal information has become mandatory, and guidelines for encryption have been widely disseminated among individuals, leading to a broad adoption of encryption technologies across society [1], [2]. Additionally, as sensitivity regarding personal information increases, law enforcement agencies are unable to compel the disclosure of passwords for encrypted files. This inability can be exploited for criminal purposes, causing significant obstacles in crime profit recovery and response, as well as societal issues like undermining public safety and weakening social security, resulting in substantial social harm [3]. Consequently, in situations where cooperation from individuals under investigation is unlikely, and without violating data protection laws, research into high-speed cryptographic decryption methods for legitimately accessing criminal evidence is urgently needed.

Cracking methods for encrypted files can essentially be divided into brute force cracking (or exhaustive key search) and table-based methods such as dictionary cracking and

The associate editor coordinating the review of this manuscript and approving it for publication was Sedat Akleylek.

rainbow table cracking [4]. In many applications, the strength of a password is gauged by the time it takes to crack it, which is influenced by the incorporation of special characters, the mixture of upper and lowercase letters, and the length of the password [5]. However, cracking a password that uses a combination of various character types with a dictionary or table becomes challenging. If a password's strength is high, the time to create such a table could be as long as the time required for a brute force attack. Additionally, it is impossible to find every random password as a hash in the table. Therefore, to search the entire possible password space, accelerating brute force cracking is necessary [6].

In the field of password cracking research, high-performance parallel computing capabilities, particularly GPUs (Graphic Processing Unit), are primarily employed to simultaneously search through large volumes of passwords. As a result, password hashing functions are designed to require significant time and memory expenses to thwart cracking attempts that leverage specialized hardware capable of efficient parallel operations, such as GPUs [7], [8]. File encryption ensures security through a combination of various cryptographic algorithms and iterations. Therefore, to achieve efficient cracking, optimization of the cryptographic algorithms is necessary. Additionally, each GPU comes with its own set of architectural constraints that can degrade performance, necessitating optimized implementations for better performance even within identical environments. The performance of these methods depends on the cipher design and the optimizations of the implementers [9]. Consequently, there has been limited research on optimizing for an efficient exhaustive search of passwords for encrypted files.

Therefore, to apply efficient brute force cracking in a parallel processing environment, optimization must be applied both to the algorithm itself and by leveraging the characteristics of GPUs. Additionally, research into distributed password cracking that utilizes the parallel processing capabilities of GPUs has been conducted. In this manner, to perform exhaustive searches, there is a need for optimized algorithms and flexible cracking libraries that utilize computing power [10]. However, commercial password cracking tools such as Passware [11] and Elcomsoft [12] offer their services for a fee and do not disclose their internal cracking methods or the structure of their algorithms, presenting challenges for further research.

Therefore, this paper aims to analyze unstructured files MS Office 2013+ (versions of Microsoft Office after 2013) and the hash algorithm PBKDF2-HMAC-SHA2 (Password-Based Key Derivation Function 2, Hash-based Message Authentication Code, Secure Hash Algorithm 2), presenting an optimized brute force cracking methodology through high-speed implementation techniques based on CPU (Central Processing Unit)/GPU, and to establish performance benchmarks for it. Moreover, to overcome the limitations of existing commercial cracking tools, it discloses code that applies optimization strategies enabling the invocation of each algorithm in the desired environment. Although multiple algorithms can be employed in file encryption, the security of most file encryptions is based on repeated hash functions. Our optimization method has the advantage of being applicable even if different algorithms are used. The ultimate goal of our research is the cracking of encrypted unstructured files. We address the optimization methodologies for cryptographic algorithms and cracking processes in GPU environments to handle this efficiently. By not only parallelizing the cracking process itself but also accelerating the cryptographic operations used in file encryption, we can significantly enhance the overall cracking speed. Furthermore, the core research of this study, the high-speed implementation technology for GPU-based cryptographic algorithms, can be applied as a strategy for high-speed/parallelization of algorithms in information security products for cloud security, network security, ransomware recovery, and can be utilized in various security application technologies.

- ***Optimization of PBKDF2-HMAC-SHA2 on GPUs*** We describe an optimization methodology at the algorithmic (SHA2, HMAC) level for PBKDF2, an algorithm for increasing the security of user-entered passwords. In addition, we implemented the algorithm in a GPU environment to parallelize the input of many passwords, and performed implementation optimizations on PBKDF2 from a memory and computation perspective to achieve the highest efficiency. This resulted in performance improvements of up to 29% on RTX 3090 (Ray Tracing Texel eXtreme 3090) and up to 33% on RTX 4090 compared to Hashcat, the world's leading password recovery software.
- ***Optimization Strategies for Cracking MS Office 2013+ on GPUs*** To undertake cracking research on the encrypted structures of Microsoft Office files, which are among the most extensively used worldwide, this study analyzed the file encryption structure of these documents and incorporated optimization research on the utilized hashing algorithms. Furthermore, the parallel computing capabilities of GPUs were harnessed for exhaustive password searches, and a structure for key generation and verification was designed to efficiently use GPU resources. The methodology also presents a way to conduct exhaustive searches for passwords of variable lengths. Our optimization methods are applicable to all versions of MS Office 2013 and beyond, allowing for the customization of the algorithm and iteration counts to create the desired password cracking tool. Our research achieved a performance improvement of up to 15% on RTX 3090 and 9% on RTX 4090 environments compared to existing commercial cracking software (Passware, Elcomsoft).

Furthermore, the key GPU-based cryptographic algorithm acceleration techniques highlighted in this study can be applied as algorithms for high-speed/parallelization in information security products for cloud security, network security,

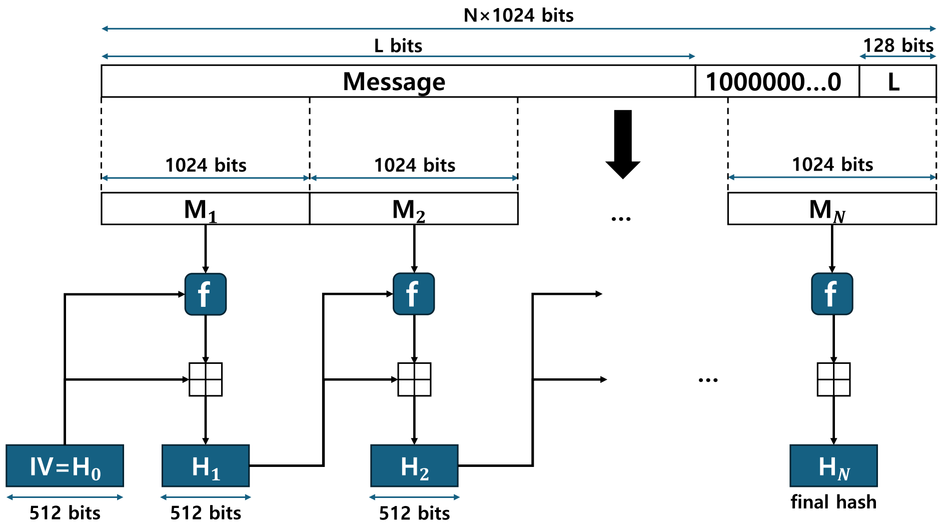


FIGURE 1. Overall computational structure of SHA512.

ransomware recovery, and various other security applications [13], [14], [15]. As previously explained, in order to overcome the limitations of existing research and contribute to the advancement of related studies, we will release the code as open-source at 'https://github.com/kindongsy/Access-Cracking_GPU' to allow others to utilize our research.

Organization: The rest of this article is structured as follows: Section II explains SHA2 and PBKDF2 for understanding the main algorithms. Section III introduces the PBKDF2 algorithm. Section IV describes the file encryption structure of MS Office 2013+. Section V introduces the architecture of GPU where our research is conducted. Building on the earlier explanations, Section VI describes optimization methodologies commonly applied, focusing on GPU environment optimization techniques and optimization of SHA2. Section VII explains optimization of PBKDF2-HMAC-SHA2 on GPUs. Section VIII proposes optimization strategies for cracking MS office 2013+ on GPUs. Afterwards, Section VIII-B provides a performance analysis of PBKDF2-HMAC-SHA2 and MS Office 2013+ cracking, while Section X concludes the paper.

II. PRELIMINARIES

In this section, we explain SHA2, a commonly used algorithm internally, and SHA2-based HMAC, which serves as the PRF (Pseudo-Random Function) of PBKDF2, to provide an understanding of PBKDF2-HMAC-SHA2 and MS Office 2013+.

A. SHA2

SHA is a cryptographic hash function adopted as a standard by NIST (National Institute of Standards and Technology) [16]. It began with SHA-0 in 1993 under FIPS PUB 180 (Federal Information Processing Standard Publication), but due to cryptographic vulnerabilities, it evolved over time, leading to the current SHA3 specified in FIPS PUB 202 [17]. Among them, SHA-2 refers to SHA-224, SHA-256, SHA-384, SHA-512/224, and SHA-512/256, and it is widely used in various fields such as integrity verification, message authentication codes, key exchange and generation algorithms, and key derivation functions. SHA-2 is a function based on the Merkle-Damgard structure, which applies a compression function repeatedly to input message blocks to produce a fixed-size output, known as a digest. It is broadly divided into two stages: Preprocessing and Hash Computation [18]. The preprocessing stage involves message padding, parsing of the padded message, and setup of initialization values. Then, Hash Computation involves generating a message schedule from the padded message, followed by iterative hashing using constant and word operations through a compression function to produce a fixed hash value. The

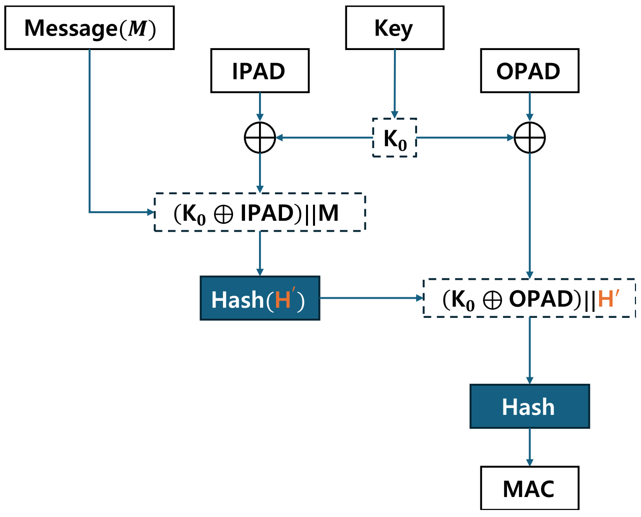


FIGURE 2. Overview of HMAC.

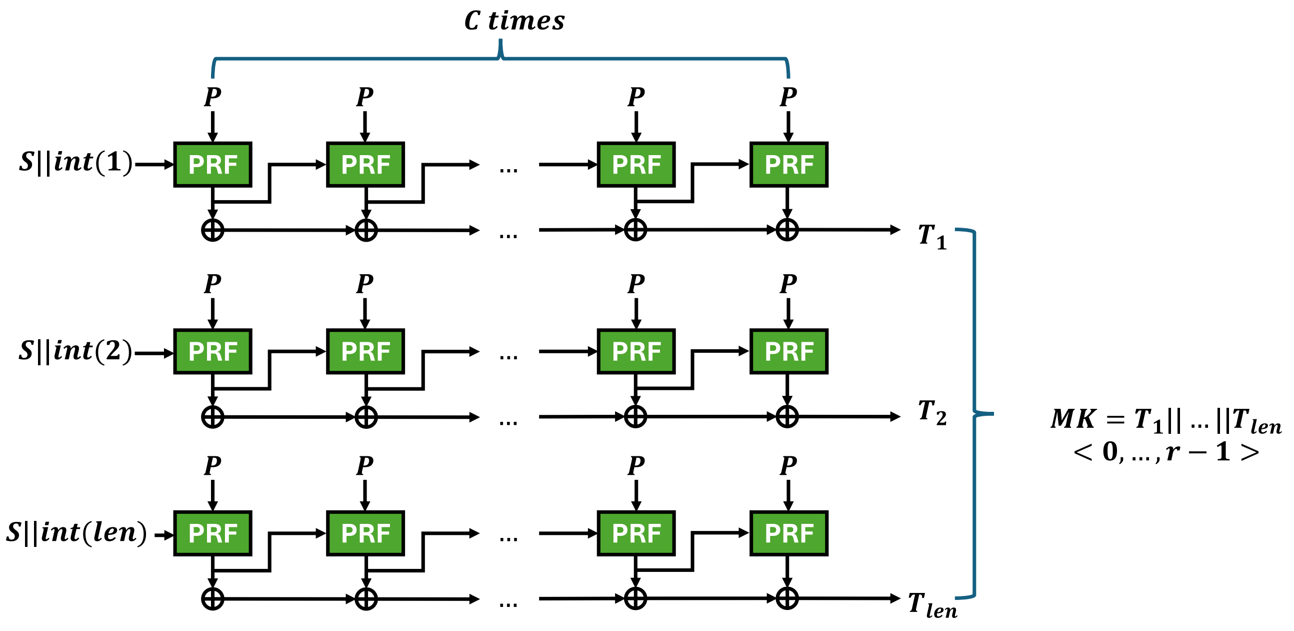


FIGURE 3. Overall computational structure of PBKDF2.

overall computational structure of SHA2 is illustrated in the Figure 1.

B. HMAC

MAC (Message Authentication Code) provides integrity verification for information exchanged between two parties sharing a secret key for authentication [19]. Differing from hash functions, MACs employ a secret key to verify message integrity and prevent third-party impersonation. HMAC is an algorithm that constructs MACs using a hash function, with its structure illustrated in Figure 2.

III. PBKDF2-HMAC-SHA2

Password systems are crucial in numerous applications for protecting users' personal information [20]. These systems ensure access permissions by matching the data entered by the user with the stored password. However, due to the low entropy and randomness of most user-generated passwords, they are ill-suited for use as direct cryptographic keys. Addressing this, key derivation cryptographic algorithms like the Password-Based Key Derivation Function (PBKDF) have been developed, which leverage the user's secret key, key length, salt, iteration count, and a PRF to produce cryptographic keys [21]. Originally, PBKDF1, employing PRFs such as SHA-1 to generate fixed-size (160 bits) outputs, prompted security concerns. Consequently, PBKDF2, which uses a more sophisticated level of PRF, was recommended. In this research, the HMAC algorithm based on SHA2 is selected as the PRF [22], [23]. The comprehensive structure of PBKDF2 is depicted in Figure 3, demonstrating our selection of the SHA2-based HMAC as the PRF for enhanced security.

TABLE 1. MS Office version and configuration algorithms.

MS Office version (Year)	Encryption Algorithm	Hash function	Repeat count of the hash function
12.0 (2007)	AES-128	SHA-1	50,000
14.0 (2010)	AES-128	SHA-1	100,000
15.0 (2013)	AES-128	SHA-512	100,000
16.0 (2016)	AES-256	SHA-512	100,000

IV. MS OFFICE 2013+

Microsoft Office, developed by Microsoft Corporation, stands as the most renowned office software suite, encompassing applications like Word, Excel, and PowerPoint [24]. The term *MS Office 2013+* encompasses MS Office 2013 and all subsequent versions, including the contemporary MS Office 365.

For encryption applications and the protection of files in non-structured formats, as outlined in Table 1, there has been a shift from AES-128 to AES-256 (Advanced Encryption Standard) in block cipher algorithms, which effectively doubles the key length. Additionally, there's an increase in both the output length of hash functions and the number of iterations. The complexity of the encryption system's internal structure and the user password input interfaces have significantly advanced, potentially enhancing security by up to 2^{256} times. The imperative to secure data within feasible investigation periods necessitates high-speed decryption and computational capabilities. This requirement motivated our research into cracking optimization, focusing on the consistent file encryption structure in MS Office

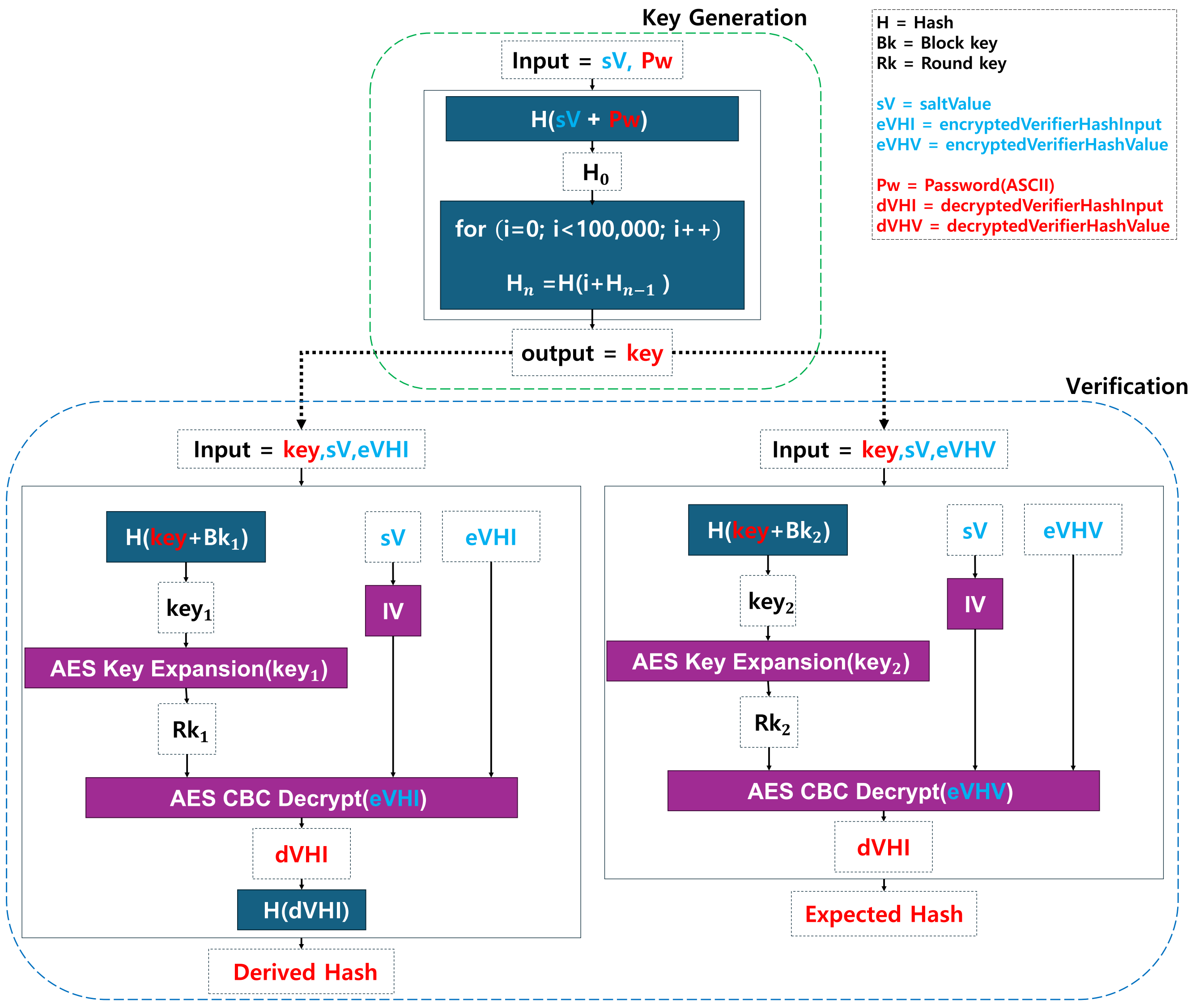
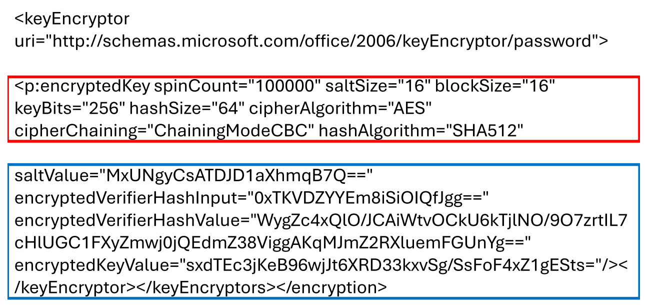


FIGURE 4. MS office 2013+ password encryption mechanism.



```
<keyEncryptor
uri="http://schemas.microsoft.com/office/2006/keyEncryptor/password">

<p:encryptedKey spinCount="100000" saltSize="16" blockSize="16"
keyBits="256" hashSize="64" cipherAlgorithm="AES"
cipherChaining="ChainingModeCBC" hashAlgorithm="SHA512"

saltValue="MxUNgyCsATDJD1aXhmqB7Q=="
encryptedVerifierHashInput="0xTKVDZYYEm8iSiOIQfJgg=="
encryptedVerifierHashValue="WygZc4xQlO/JCAiWtvOCkU6kTjlNO/9O7zrtIL7
cHIUGC1FXyZmwj0jQEdmZ38ViggAKqMJmZ2RXluemFGUnYg=="
encryptedKeyValue="sxdTEc3jKeB96wjJt6XRD33kxvSg/SsFoF4xZ1gESts="/><
/keyEncryptor></keyEncryptors></encryption>
```

FIGURE 5. Encryption information of MS office 365.

versions from 2013, including MS Office 2016 and MS Office 365.

The password encryption structure for MS Office 2013+ documents is outlined in [25]. This can be illustrated as shown in Figure 4. Following this configuration, renaming a MS Office 2013+ document file with a.zip extension and decompressing it yields EncryptionInfo and EncryptedPackage files. The EncryptionInfo file, in particular, allows for the inspection of encryption information.

Figure 5 displays information extracted from an actual MS Office 365 Word document, using the previously mentioned methods. The data, highlighted in red, shows the encryption and hash algorithm's configuration for MS Office 2013+ documents. Additionally, indicated in blue, the input and output values of the encryption and hash algorithm can be examined. This examination confirms that MS Office 365 (and versions from MS Office 2013 onwards) uses AES-256 as the encryption algorithm and performs 100,000 iterations of the SHA-512 hash function.

The detailed password encryption and decryption process of actual MS Office 2013+ is as follows. First, the initial hash function is performed by concatenating the *saltValue*

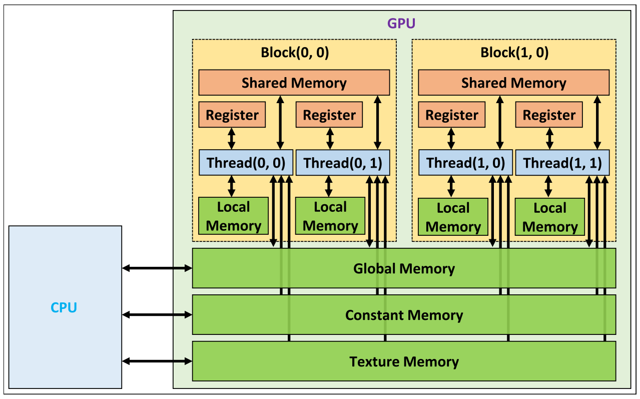


FIGURE 6. Memory structure of NVIDIA GPUs [26].

(sV) extracted from the encryption information with the user-inputted ASCII (American Standard Code for Information Interchange) form of the *Password (Pw)*. This value is then repeatedly hashed 100,000 times by concatenating it with the increasing index value and the result of the previous hash. In the subsequent verification process, the *key* and *block* key_1 (Bk_1) are concatenated and hashed to calculate key_1. Based on this key_1, the AES Key Expansion is performed to generate *Round* key_1 (Rk_1). Then, the *sV* is used to create the *IV (initialization Vector)* for AES CBC (Cipher Block Chaining mode) Decryption. AES CBC Decryption is then performed on the *encryptedVerifierHashInput (eVHI)* to obtain the *decryptedVerifierHashInput (dVHI)*. This *dVHI* is then hashed to compute the *Derived Hash*. A similar process is followed for the *encryptedVerifierHashValue (eVHV)*, but the final hash step is omitted. If the *Derived Hash* matches the *Expected Hash (dVHV)*, it is considered a correct password.

V. GPU

Originally, GPUs were architectures designed for high quality graphical processing and design tasks through parallel processing based on a large number of threads. As the performance of GPUs continues to evolve, General Purpose Graphic Processing Unit (GPGPU) technology has been developed to harness the computing power of GPUs for general purpose data operations. NVIDIA has developed and provided CUDA (Compute Unified Device Architecture) [27], a development framework for GPU computing, to enable ordinary users to make versatile use of GPU computing technology. With this in mind, research into the use of GPUs as cryptographic accelerators in server environments is actively underway [28], [29]. Taking advantage of the highly parallelizable computational capabilities of GPUs, research on cracking encrypted data is also being conducted in various environments, including this one.

- **Memory Structure of NVIDIA GPUs**
  The GPU memory architecture, shown in Figure 6, is organised into on-chip and off-chip memory, corresponding to the GPU's Streaming Multiprocessor (SM). On-Chip memory, located within the processor, consists of shared memory and registers, providing fast access speeds but with limitations on access methods and storage capacity. Conversely, off-chip memory, located outside the processor, includes global, constant and local memory types. This external memory offers more flexible access at the cost of slower speeds compared to on-chip memory.
- **Computational Structure of NVIDIA GPUs**
  First, the CPU passes the data to be processed on the GPU through kernel functions, and then the SM performs operations on the received data at the warp level, which is the basic computational unit of the GPU. These warps typically consist of 32 threads in most GPU architectures, and the threads within a warp follow the Single Instruction Multiple Thread (SIMT) structure, where they all perform the same operation.

VI. COMMON OPTIMIZATION METHOD

In this section, we explain optimisation methods that exploit the characteristics of the NVIDIA GPU [30], which are commonly used in the PBKDF2-HMAC-SHA2 and MS Office 2013+ cracking algorithms. We also discuss approaches to algorithmic optimisation of SHA2, which is commonly used in both algorithms.

A. OPTIMIZATION STRATEGIES IN GPU ENVIRONMENTS

1) COALESCED MEMORY ACCESS PATTERN

When the GPU receives data from the CPU, it typically uses global memory as its main memory. Global memory has a large capacity, but suffers from slow memory access speeds. Therefore, when threads running at warp level need to access required data, accessing data serially results in accessing a limited amount of data per cache line, as shown in the upper part of Figure 7. However, by reorganising the data as shown in the lower part of the Figure 7, more data can be accessed per cache line. This memory access technique is called Coalesced Memory Access Pattern, which is a prominent memory access optimisation technique in GPUs.

2) SHARED MEMORY

Shared memory is memory that can be allocated per block, which is a logical grouping of threads, and is on-chip memory, which means it has limited capacity but provides low latency when accessing data. In addition, threads within an allocated block can communicate efficiently through shared memory, and they can read from and write to shared memory within kernel functions. Therefore, to take advantage of the fast access speed of shared memory, optimization from a memory perspective can be achieved within the GPU by fetching and ordering data from global memory or by storing frequently accessed data in shared memory during computation.

To achieve high bandwidth, shared memory is divided into modules called 'Banks', each of which is of the same size (32-bit), allowing simultaneous access by threads within a warp (hence banks are typically composed of

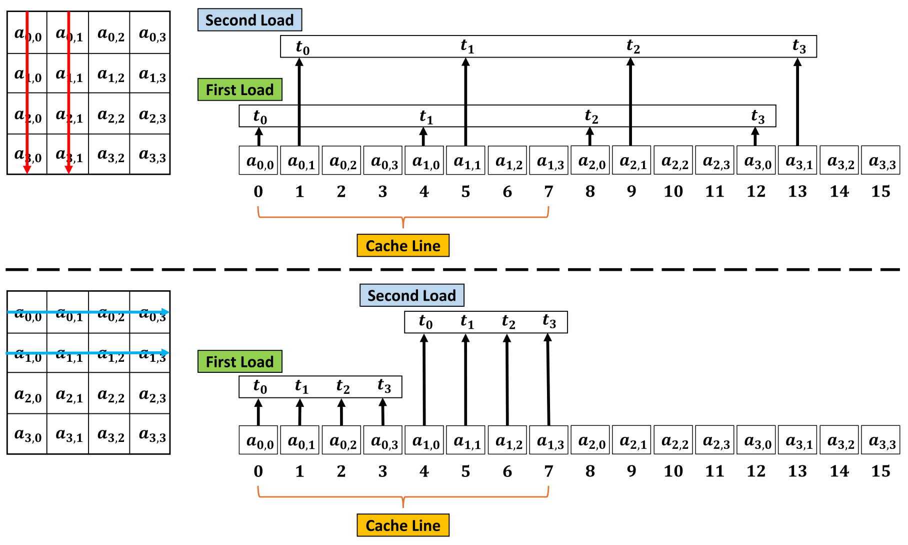

Second Load
First Load
Cache Line
t_0 t_1 t_2 t_3
$a_{0,0}$ $a_{0,1}$ $a_{0,2}$ $a_{0,3}$ $a_{1,0}$ $a_{1,1}$ $a_{1,2}$ $a_{1,3}$ $a_{2,0}$ $a_{2,1}$ $a_{2,2}$ $a_{2,3}$ $a_{3,0}$ $a_{3,1}$ $a_{3,2}$ $a_{3,3}$
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

FIGURE 7. GPU coalesced memory access pattern.

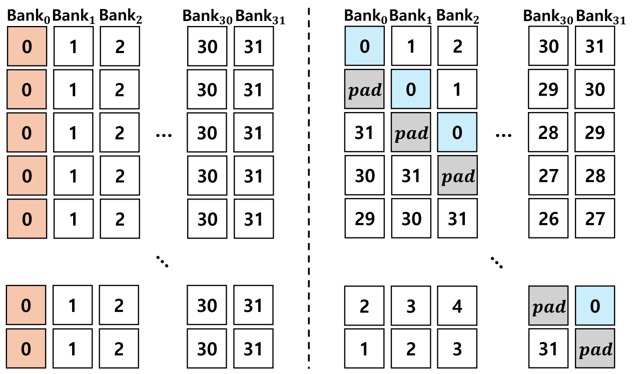


FIGURE 8. Approaches to resolve bank conflicts in shared memory.

32 modules). However, if two or more memory requests map to the same memory bank, a bank conflict occurs, resulting in serialization of access to that bank and a reduction in throughput proportional to the number of bank accesses. Therefore, minimizing bank conflicts is essential to achieve maximum performance when using shared memory. To eliminate such bank conflicts, we have partitioned the data accessing shared memory into 32-bit units and adjusted the padding with garbage values to ensure that different threads within the same warp do not access the same bank, as shown in Figure 8.

3) CONSTANT MEMORY

Constants are a small amount of off-chip memory per GPU, typically 64KB in total, that is accessible to all threads within the device. However, it is read-only; once data is copied from the CPU to the GPU, it cannot be modified within GPU functions. Nevertheless, constant memory behaves like cache memory, allowing fast access when all threads within a given warp are accessing the same memory address.

4) WARP DIVERGENCE

GPU, a hardware based on SIMT architecture, executes a single instruction across all threads within a warp for efficient computation. For optimal operation, all 32 threads within a warp should follow the same execution path. When threads within a warp diverge due to branching instructions, certain threads become idle, resulting in serialization of operations. This phenomenon is known as warp divergence, and it is therefore more efficient from a parallel processing perspective to avoid branching instructions to ensure that threads within the same warp execute the same operations.

5) LOCAL MEMORY

When a kernel exceeds the hardware limit on the number of registers it can use, the data corresponding to the excess registers spills over into local memory. This register spilling degrades performance because accessing local memory is just as expensive as accessing global memory. Although each thread can have a maximum of 255 32-bit registers, regardless of the GPU architecture's computational capability, the maximum usage can vary depending on the maximum number of registers per thread block. Therefore, efficient management of block and thread configurations and register usage is required.

6) OCCUPANCY

Occupancy is the ratio of the number of active warps per multiprocessor to the maximum number of active warps possible. Higher occupancy does not always equate to higher performance-there is a point beyond which additional occupancy does not improve performance. However, low occupancy always degrades the ability to hide memory latency, resulting in performance degradation. Factors that affect this occupancy include warps per SM, blocks per SM, registers per SM, and shared memory per SM. Therefore,

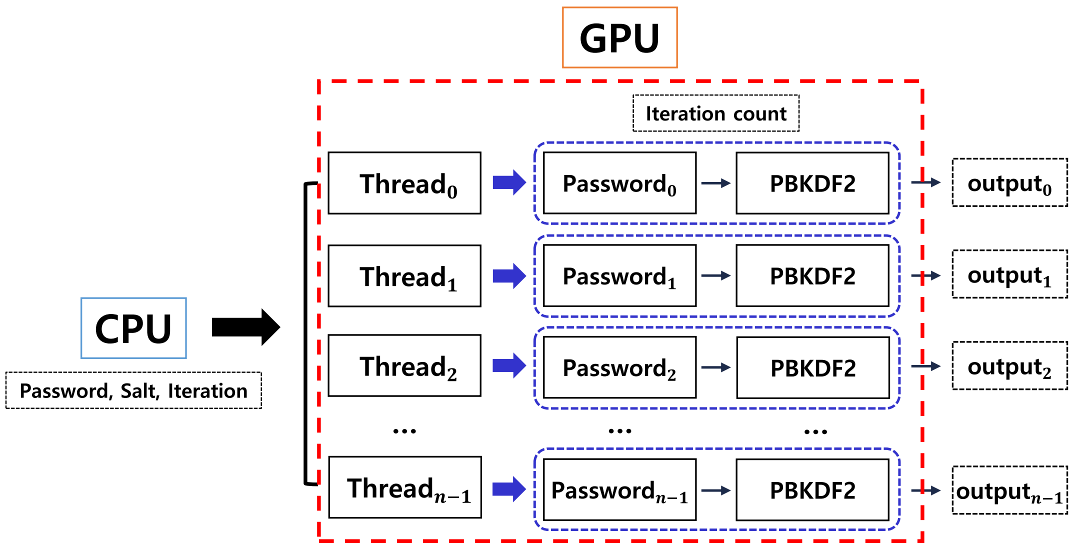


FIGURE 9. Parallel structure of PBKDF2.

TABLE 2. Block Operation Process in SHA256 [34].

Variables	Round 0	Round 1	Round 2	Round 3
a	$U_{(0,1)}+U_{(0,2)}$	$U_{(1,1)}+U_{(1,2)}$	$U_{(2,1)}+U_{(2,2)}$	$U_{(3,1)}+U_{(3,2)}$
b	a	$U_{(0,1)}+U_{(0,2)}$	$U_{(1,1)}+U_{(1,2)}$	$U_{(2,1)}+U_{(2,2)}$
c	b	a	$U_{(0,1)}+U_{(0,2)}$	$U_{(1,1)}+U_{(1,2)}$
d	c	b	a	$U_{(0,1)}+U_{(0,2)}$
e	$U_{(0,1)}+d$	$U_{(1,1)}+c$	$U_{(2,1)}+b$	$U_{(3,1)}+a$
f	e	$U_{(0,1)}+d$	$U_{(1,1)}+c$	$U_{(2,1)}+b$
g	f	e	$U_{(0,1)}+d$	$U_{(1,1)}+c$
h	g	f	e	$U_{(0,1)}+d$

achieving optimal computational efficiency in GPU environments requires consideration of the above computational and memory characteristics.

B. OPTIMIZATION OF SHA2

Most hash functions have a chaining operation structure [31], [32], [33], which makes it difficult to parallelize internal operations. Therefore, in SHA2 optimization research, the Block Operation (BO) and Zero-Based Optimization (ZB) methods used in [34] have been applied.

The BO technique is an optimization method that reduces unnecessary memory accesses during internal block operations of SHA2. When applied, it can reduce the 32 memory allocations of 4 rounds to 8 memory allocations based on SHA256 as shown in Table 2. Therefore, when applied to 64 rounds, it can be processed with 16 memory allocation operations. This optimization method is suitable not only for CPU environments, but also for GPUs, where memory access overhead is relatively high. In addition, it can reduce register usage per thread, making it a beneficial approach from the perspective of occupancy and thread/block heuristics.

$$W_t = \begin{cases} M_t^{(i)} & 0 \leq t \leq 15 \\ \sigma_1^{\{256\}}(W_{t-2}) + W_{t-7} + \sigma_0^{\{256\}}(W_{t-15}) + W_{t-16} \\ & 16 \leq t \leq 63 \end{cases} \tag{1}$$

The ZB technique is an optimization method for the message scheduling part of SHA2 operations. In this method, if the calculated $W[t]$ is zero during the word expansion process, as shown in equation 1, the corresponding operation is skipped to reduce the computational load. While this approach may not significantly reduce computation for typical data, it can result in a reduction in computation proportional to the fixed length of zeros when applied to scenarios involving one-zero padding.

VII. OPTIMIZATION OF PBKDF2-HMAC-SHA2 ON GPUS

In this section, we present the optimization techniques used to achieve the highest throughput performance for PBKDF2-HMAC-SHA2. The PBKDF2 algorithm is characterized by the iterative computation of the PRF and its input-output, which makes internal parallelization impossible. Therefore, from the perspective of throughput, research has been conducted by performing one PBKDF2 operation per GPU thread, as shown in Figure 9.

A. OPTIMIZATION OF HMAC

Furthermore, since the PRF chosen for this study, the HMAC algorithm, is based on SHA2, not only the optimization methods for SHA2 described earlier can be applied, but also the HMAC optimization methods proposed in [34]. The first method proposed in that paper is Block-Reduction (BR), which takes advantage of the fact that the password, a primary

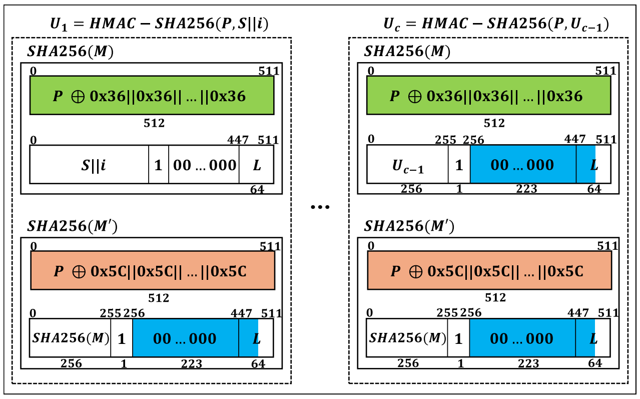

$U_1 = HMAC - SHA256(P, S||i)$
$U_c = HMAC - SHA256(P, U_{c-1})$
$SHA256(M)$
$P \oplus$ 0x36||0x36|| ... ||0x36
$S||i$
1
00 ... 000
L
U_{c-1}
$SHA256(M')$
$P \oplus$ 0x5C||0x5C|| ... ||0x5C
$SHA256(M)$
0
511
512
447
255
256
223
64

FIGURE 10. Block operation reduce method for PBKDF2-HMAC [35].

input of PBKDF2, remains unchanged during computation, thereby reducing the computational burden.

Looking at the first input block in HMAC, as shown in Figure 10, Password is XORed with *IPAD* or *OPAD*. Since Password remains unchanged after being entered by the user, it is possible to precompute this part, which can then be reused in all PRF operations. Therefore, instead of performing the operations for each iteration over the existing 4 blocks ($4 \times$ *iteration count*), the workload can be reduced to the operations for precomputing the 2 blocks plus the operations for the 2 blocks for the iteration count ($2+2\times$ *iteration count*).

The second optimization method concerns the input size (IS), as shown in Figure 10. Each hash function, as shown, processes two block operations. The second block contains the message length information and padding. Due to the characteristics of the hash function during this process, the message length information and padding become fixed, eliminating the need for block division and message length verification for this data. Therefore, this optimization method eliminates the aforementioned verification. Furthermore, building on this, we have eliminated branching instructions and block division based on length information within the hash function to further improve performance.

B. OPTIMIZATION USING GPU MEMORY CHARACTERISTICS

For PBKDF2, it takes iteration count, password, and salt as inputs. For the salt, since it's initial value remains unchanged and is only used as the first input during the HMAC-SHA2 process, we used constant memory, which shows fast speeds when accessed by all threads simultaneously. As for the password, after it is entered into the hash function, it is iteratively updated according to the number of iterations. This results in a computational overhead that is proportional to the number of iterations. To solve this problem, we used BO. Since password access and updates occur mainly during hash function iterations, we first fixed the password length and applied coalesced memory access patterns during memory copying between CPU and GPU. Later, we transferred intermediate data positions during these operations to shared memory to further improve access speeds. To avoid potential bank conflicts during this process, we applied padding techniques.

C. NVIDIA NSIGHT COMPUTE

In addition, for detailed analysis, we examined our PBKDF2-HMAC-SHA2 algorithm using NVIDIA Nsight Compute (version 2023.3.1) [36], an interactive profiler provided by NVIDIA. With a compute SM throughput of 93.15%, we achieved high throughput (compute throughput over 80%), demonstrating significant processing efficiency in our implementation. By applying optimization strategies to the hash functions, our workload analysis showed that the SM Busy, which indicates the pipeline used for the ALU (Arithmetic Logic Unit), reached 95.34%. In addition, we achieved a Theoretical Occupancy of 22.25%, which is close to the upper bound represented by Occupancy, which is 25%. This indicates that our implementation is highly optimized from an occupancy perspective as well.

VIII. OPTIMIZATION STRATEGIES FOR CRACKING MS OFFICE 2013+ ON GPUS

In this section, we present the optimization strategies used for efficient implementation of MS Office 2013+ cracking. Similar to PBKDF2, the MS Office 2013+ algorithm relies on a high-iteration hash function to ensure security. Therefore, we designed a structure to perform exhaustive search by cracking one password per thread, as shown in Figure 11. Since the password length in MS Office 2013+ files is not fixed, we applied a structure to cover all possible lengths for exhaustive search. After that, the MS Office 2013+ cracking process is divided into key generation and verification stages. Among them, the key generation stage, which involves iterative computation of the hash function, imposes the largest computational burden. Therefore, we focus on optimizing this stage to improve the overall performance. Our cracking method, as shown in Table 1, can be applied to crack all encrypted files starting from MS Office 2013. In addition, our optimization method allows you to include the desired algorithms, making it usable as the desired password cracking tool.

A. APPROACHES FOR EXHAUSTIVE PASSWORD SEARCH ON GPUS

We considered how to efficiently perform exhaustive password search for password cracking. Since not all threads used in cracking need to work on the same password, we declared an array in the kernel for the password length and used blockIdx, which is the CUDA-specific index of the thread block, and threadIdx within a thread block to perform an exhaustive search for all possible characters (96 characters excluding control characters) represented in ASCII, as shown in Figure 11. However, with this approach, it was only possible to examine passwords up to two characters in length using the thread block and thread index. So we extended it to three characters using CUDA Streams. Originally,

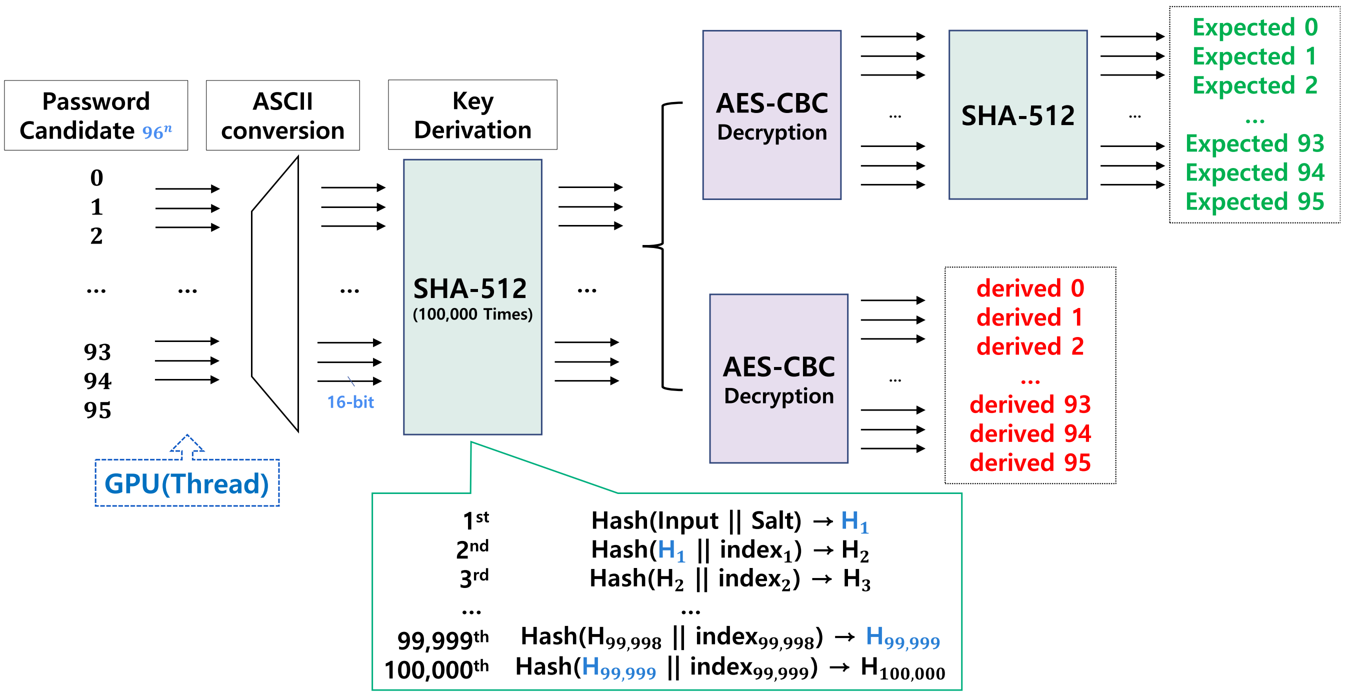


FIGURE 11. Proposed structure for parallel cracking of MS Office 2013+.

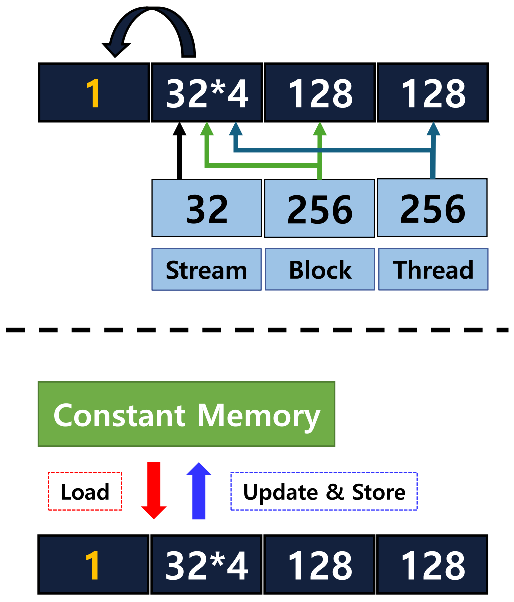


FIGURE 12. Password update methodology.

CUDA streams are used to asynchronously execute logic on a stream-by-stream basis for memory copies and kernel operations to reduce data copy and operation times. However, by using the index of CUDA streams along with blockIdx and threadIdx, we can increase the password length to three characters and perform calculations within a single kernel. Therefore, we perform an exhaustive search for passwords of length three using blockIdx, threadIdx, and the stream index, as shown in Figure 12.

B. OPTIMIZATION USING GPU MEMORY CHARACTERISTICS

After cracking three-character passwords using the approach described above, we used constant memory for passwords of four or more characters. Once the cracking for three-character passwords is complete (if the password is not three characters long), the value of the next array (with an index of 3 or greater) is incremented outside the kernel. Then, this incremented value (count) is passed to constant memory and another kernel call is made to continue the cracking process. We chose this approach to take advantage of constant memory. Since constant memory is read-only, its values cannot be changed within the kernel. However, it has similar characteristics to cache memory, making it highly efficient when accessed repeatedly by all threads. Therefore, constant memory was chosen as the memory for entering operand values into the structure, where the incremented count value is applied uniformly.

For the calculation of hash values for passwords and the verification process to check if the correct password is found, certain values that are used repeatedly but remain fixed are also stored in constant memory. These fixed values include salts and encryptedVerifierHashInput parsed from encryption information for password cracking, as well as salts from the actual encrypted files for verification and encryptedVerifierHashValue used for AES CBC mode decryption. In the key generation part, which is the most computationally

TABLE 3. PBKDF2-HMAC-SHA2 Performance Analysis (Unit: hash/s, performance improvement rate(%)).

Classification		RTX 3090		RTX 4090	
Algorithms	Iteration count	Hashcat	**Our Works**	Hashcat	**Our Works**
PBKDF2 HMAC (SHA256)	1,000	3,442.3k	**3,980.3k**(15% ↑)	8,222.2k	**8,658.8k**(5% ↑)
	10,000	356.2k	**391.2k**(10% ↑)	833.2k	**862.3k**(3% ↑)
	100,000	30,133	**38,881**(29% ↑)	81,612	**86,493**(6% ↑)
	600,000	5,514	**6,684**(21% ↑)	14,418	**14,841**(3% ↑)
PBKDF2 HMAC (SHA512)	1,000	1,337.3k	**1,484.6k**(11% ↑)	2,938.0k	**3,298.0k**(12% ↑)
	10,000	131.2k	**145.3k**(10% ↑)	295.3k	**331.6k**(12% ↑)
	100,000	11,983	**14,417**(20% ↑)	28,491	**33,156**(16% ↑)
	210,000	5,647	**6,721**(20% ↑)	11,778	**15,763**(33% ↑)

TABLE 4. Performance Measurement Environment.

Classification	Architecture
OS	Windows 10 Pro
CPU	13th Gen Intel(R) Core(TM) i7-13700K 3.40 GHz(x64)
RAM	32.0 GB
GPU	NVIDIA RTX 3090, RTX 4090
IDE	Visual Studio 2022
CUDA version	12.2

TABLE 5. GPU Architecture specifications [37].

Classification	RTX 3090	RTX 4090
CUDA cores	10,496	16,384
Architecture	Ampere	Ada Lovelace
Graphic Processor	GA102	AD102
Streaming Multiprocessor	82	128
Bandwidth	936.2 GB/s	1,008.0 GB/s
Base Clock	1,395 MHz	2,235 MHz
Compute Capability	8.6	8.9
Power Consumption	350W	450W

intensive, we took advantage of shared memory and existing optimization research on the SHA2 series [34]. Specifically, for the input and output used when the hash function is called over 100,000 times, there is a significant amount of memory access during the hash function call and internal operations. Therefore, we transferred this data to shared memory to improve memory access speed and padded garbage values in the input data to prevent potential bank conflicts that may occur during shared memory usage.

IX. PERFORMANCE ANALYSIS

In this study, we conducted a performance analysis by comparing PBKDF2-HMAC-SHA2 with the world-renowned cracking software Hashcat (version 6.2.6) from a throughput perspective. For cracking MS Office 2013+ documents, we compared performance using tools provided by Passware (version 2023.3.1) and Elcomsoft (version 2023.4.50), both of which are widely available services. In the case of coin mining, which uses repetitive hash functions similar to file cracking, current GPU architectures are not as efficient in terms of cost and power consumption compared to their mining performance. As a result, coin mining has predominantly shifted to ASIC (Application Specific Integrated Circuit) implementations, and there has been little development in efficient software implementations for mining using GPUs. Therefore, to the best of our knowledge, Hashcat, which optimizes hash functions for cracking, has achieved the world's best performance in CPU/GPU environments. Consequently, we selected Hashcat as our benchmark for comparison. The detailed performance measurement environment is outlined in Table 4. In addition, Table 5 provides the specifications of the main architectures used in our study, namely the RTX 3090 and RTX 4090. Furthermore, we would like to point out that the performance evaluation of our PBKDF2-HMAC-SHA2 and MS Office 2013+ cracking has received official certification through an assessment conducted by the Korea Security Evaluation Laboratory (KSEL).

In the case of PBKDF2-HMAC-SHA2, security can be adjusted through the iteration count. Therefore, performance measurements were performed by setting iteration counts suitable for different environments. In addition, performance analysis was performed up to the PBKDF2-HMAC-SHA2 iteration counts recommended by the Open Worldwide Application Security Project (OWASP), which are 600,000 iterations for SHA256 and 210,000 iterations for SHA512. For PBKDF2-HMAC-SHA256, the most significant performance improvement was observed at about 100,000 iterations. Compared to Hashcat, there was a maximum improvement of 29% and 6% in the RTX 3090 and RTX 4090 environments, respectively. In the case of the RTX 4090, the performance improvement was slightly lower than that of the RTX 3090, likely due to the improvement in core performance and memory access speed. For PBKDF2-HMAC-SHA512, the most significant performance improvement was achieved at 210,000 iterations, with 20% and 33% performance improvements in the RTX 3090 and RTX 4090 environments, respectively. Since the SHA512 iteration, which is the most

TABLE 6. **MS Office 2013+ Cracking Performance Analysis (Unit: Performance(pass/s), Percentage(%)).**

Classification	RTX 3090		RTX 4090	
	Performance	Percentage	Performance	Percentage
Passware	25,500	100	60,872	100
Elcomsoft	25,700	101	61,000	101
Our Work	**29,493**	**115**(15% ↑)	**66,216**	**109**(9% ↑)

intensive, requires more than twice as much computation as SHA256, it is judged that the optimization efficiency is better reflected even in the RTX 4090 environment. Furthermore, based on the Table 3 above, it can be speculated that the range of 100,000 to 200,000 iterations is the most efficient compared to Hashcat. Beyond this range, as the number of iterations increases, the load on internal hash function operations becomes significant, leading to less significant performance improvements.

Performance measurements were conducted using Microsoft Office 365's Word as the target for MS Office 2013+ cracking. As shown in Figure 5, the number of hash function iterations for cracking MS Office 2013+ files is determined by the encryption information of the encrypted file and is set according to MS Office's security policy. Therefore, in the case of MS Office 2013+, cracking was performed through 100,000 hash function iterations, as shown in Table 1. Performance was measured by running Passware's and Elcomsoft's cracking tools, achieving approximately 15% and 9% performance improvements on RTX 3090 and RTX 4090 environments, respectively, as shown in Table 6.

One of the advantages of our research is the ability to perform computations on arbitrary inputs, allowing each GPU in a multi-GPU environment to process different arbitrary inputs. Through additional research, we confirmed that our method functions correctly in a multi-GPU setup, accommodating user inputs appropriately. When utilizing multiple GPUs, considering the Compute Capability of the micro-architecture within each GPU, it is possible to achieve proportional performance improvements as presented in our performance analysis. This means that the performance scales linearly with the number of GPUs, yielding the same performance per GPU as with a single GPU.

Consequently, the performance improvement achieved through our research has the following practical significance. By saving up to 16% of the time, we can reduce the actual cost of cracking by 16%. For example, if it takes 10 hours to crack a particular encrypted file, a 16% performance improvement might not seem substantial. However, if it takes 100 days, reducing 16 days is a significant enhancement. As the password length increases, the shortened execution time for these computations will provide even greater benefits. Additionally, since GPUs are high-cost architectures, these time savings translate into tangible material benefits. Furthermore, our research enables computations for arbitrary user inputs in both single and multi-GPU environments, consistently delivering performance close to the maximum in password search. This amplifies the aforementioned advantages. Moreover, the existing cracking products we referenced in our paper (Hashcat, Passware, Elcomsoft) are all designed based on GPU architectures. Therefore, the 16% (or 9%) performance improvement we achieved in the same GPU environment is by no means insignificant.

X. CONCLUSION

In conclusion, our research focuses on efficient cracking of encrypted unstructured files, addressing optimization methodologies for cryptographic algorithms and cracking processes in GPU environments. Compared to the world's leading cracking software (Hashcat, Passware, Elcomsoft), we achieved remarkable performance improvements of up to 29% and 15% for PBKDF2-HMAC-SHA2 and MS Office 2013+, respectively, on the RTX 3090, and up to 33% and 9% on the RTX 4090. This demonstrates significant advancements in high-speed encryption cracking techniques, not only through parallel processing of the cracking itself but also by accelerating the cryptographic operations used in file encryption. These findings highlight the potential of our approach to provide substantial time and cost savings as the duration of file cracking increases, contributing to criminal investigations by enabling faster access to encrypted data. Furthermore, by providing open access to our proposed optimization methods and open-source implementations, our research allows for further improvements through subsequent public research. Our optimization techniques are also applicable to other algorithms, offering versatility and broader applicability. This, in turn, lays a foundation for implementing high-speed and parallelized encryption algorithms in various information security applications, such as cloud security, network security, and ransomware recovery.

CONFLICT OF INTEREST

All authors have no conflict of interest.

REFERENCES

[1] P. G. Kelley, S. Komanduri, M. L. Mazurek, R. Shay, T. Vidas, L. Bauer, N. Christin, L. F. Cranor, and J. Lopez, "Guess again (and again and again): Measuring password strength by simulating password-cracking algorithms," in *Proc. IEEE Symp. Secur. Privacy*, May 2012, pp. 523–537.

[2] L. Han, "Password cracking and countermeasures in computer security: A survey," 2014, *arXiv:1411.7803*.

[3] K. Ilker, "Password cracking methods and techniques in computer forensic investigation," *Acta Infologica*, vol. 5, no. 1, pp. 27–38, 2021.

[4] F. Yu and Y. Huang, "An overview of study of passowrd cracking," in *Proc. Int. Conf. Comput. Sci. Mech. Autom. (CSMA)*, Oct. 2015, pp. 25–29.

[5] A.-D. Vu, J.-I. Han, H.-A. Nguyen, Y.-M. Kim, and E.-J. Im, "A homogeneous parallel brute force cracking algorithm on the GPU," in *Proc. ICTC*, Sep. 2011, pp. 561–564.

[6] K.-W. Kim, S.-S. Lee, D.-W. Hong, and J.-C. Ryou, "GPU-accelerated password cracking of PDF files," *KSII Trans. Internet Inf. Syst. (TIIS)*, vol. 5, no. 11, pp. 2235–2253, 2011.

[7] R. Álvarez, A. Andrade, and A. Zamora, "Optimizing a password hashing function with hardware-accelerated symmetric encryption," *Symmetry*, vol. 10, no. 12, p. 705, Dec. 2018.

[8] I. L. S. Hendarto and Y. Kurniawan, "Performance factors of a CUDA GPU parallel program: A case study on a PDF password cracking brute-force algorithm," in *Proc. Int. Conf. Comput., Control, Informat. Appl. (IC3INA)*, Oct. 2017, pp. 35–40.

[9] C. Tezcan, "Key lengths revisited: GPU-based brute force cryptanalysis of DES, 3DES, and PRESENT," *J. Syst. Archit.*, vol. 124, Mar. 2022, Art. no. 102402.
[10] R. Hranický, L. Zobal, V. Večera, and P. Matoušek, "Distributed password cracking in a hybrid environment," in *Proc. SPI*, 2017, pp. 75–90.
[11] Passware. (2023). *Passware Kit Forensic*. Accessed: Mar. 21, 2024. [Online]. Available: https://www.passware.com/kit-forensic/
[12] MCF Elcomsoft Desktop. (2023). *Elcomsoft Products and Solutions*. Accessed: Mar. 21, 2024. [Online]. Available: https://www.elcomsoft.com/products.html/
[13] W.-K. Lee, X.-F. Wong, B.-M. Goi, and R. C.-W. Phan, "CUDA-SSL: SSL/TLS accelerated by GPU," in *Proc. Int. Carnahan Conf. Secur. Technol. (ICCST)*, Oct. 2017, pp. 1–6.
[14] S. Morishima and H. Matsutani, "Accelerating blockchain search of full nodes using GPUs," in *Proc. 26th Euromicro Int. Conf. Parallel, Distrib. Network-Based Process. (PDP)*, Mar. 2018, pp. 244–248.
[15] W. Pan, F. Zheng, Y. Zhao, W.-T. Zhu, and J. Jing, "An efficient elliptic curve cryptography signature server with GPU acceleration," *IEEE Trans. Inf. Forensics Security*, vol. 12, no. 1, pp. 111–122, Jan. 2017.
[16] National Institute of Standards and Technology. (2017). *Hash Functions*. Accessed: Mar. 7, 2024. [Online]. Available: https://csrc.nist.gov/projects/hash-functions
[17] National Institute of Standards and Technology, *SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions*, SFederal Inf. Process., Standards 202, Aug. 2015, U.S. Dept. Commerce, Washington, DC, USA, 2015.
[18] National Institute of Standards and Technology, *Secure Hash Standard*, Federal Inf. Process., Standards 180-4, U.S. Dept. Commerce, Washington, DC, USA, 2015.
[19] National Institute of Standards and Technology, *The Keyed-Hash Message Authentication Code (HMAC)*, Federal Inf. Process., Standards 198-1, Jul. 2008, U.S. Dept. Commerce, Washington, DC, USA, 2008.
[20] R. Khande, S. Ramaswami, C. Naidu, and N. Patel, "An effective mechanism for securing and managing password using AES-256 encryption & PBKDF2," *Int. J. Electr. Eng. Technol.*, vol. 12, no. 5, pp. 1–7, May 2021.
[21] National Institute of Standards and Technology, *Recommendation for Password-Based Key Derivation*, Special, Standard 800-132, U.S. Dept. Commerce, Washington, DC, USA, Dec. 2010.
[22] M. Dürmuth and T. Kranz, "On password guessing with GPUs and FPGAs," in *Technology and Practice of Passwords*, Trondheim, Norway. Cham, Switzerland: Springer, 2015, pp. 19–38.
[23] F. Yu, Y. Shi, and H. Yin, "A fast recovery of encrypted message database of WeChat on GPU," in *Proc. IEEE 5th Int. Conf. Signal Image Process. (ICSIP)*, Oct. 2020, pp. 980–984.
[24] R. Hranický, "Digital forensics: The acceleration of password cracking," Ph.D. thesis, Fac. Inf. Technol., Brno Univ. Technol., Brno, Czechia, 2022. [Online]. Available: https://www.fit.vut.cz/study/phd-thesis/890/
[25] J. Hong, Z. Chen, and J. Hu, "Analysis of encryption mechanism in office 2013," in *Proc. IEEE 9th Int. Conf. Anti-Counterfeiting, Secur., Identificat. (ASID)*, Sep. 2013, pp. 29–32.
[26] A. Corana, "Architectural evolution of NVIDIA GPUs for high-performance computing," IEIIT-CNR, Genoa, Italy, Tech. Rep. IEIIT-CNR-150212, 2015, doi: 10.13140/RG.2.1.1496.1042.
[27] NVIDIA. (2024). *Cuda C++ Programming Guide V12.4*. Accessed: Mar. 15, 2024. [Online]. Available: https://docs.nvidia.com/cuda/cuda-c-programming-guide
[28] A. A. Badawi, B. Veeravalli, C. F. Mun, and K. M. M. Aung, "High-performance FV somewhat homomorphic encryption on GPUs: An implementation using CUDA," *IACR Trans. Cryptograph. Hardw. Embedded Syst.*, vol. 2018, pp. 70–95, May 2018.
[29] W. Jung, S. Kim, J. H. Ahn, J. H. Cheon, and Y. Lee, "Over 100x faster bootstrapping in fully homomorphic encryption through memory-centric optimization with GPUs," *IACR Trans. Cryptograph. Hardw. Embedded Syst.*, vol. 2021, pp. 114–148, Aug. 2021.
[30] NVIDIA. (2015). *Achieved Occupancy*. Accessed: Mar. 12, 2024. [Online]. Available: https://docs.nvidia.com/gameworks/content/developertools/desktop/analysis/report/cudaexperiments/kernellevel/achievedoccupancy.htm
[31] C. Ge, L. Xu, W. Qiu, Z. Huang, J. Guo, G. Liu, and Z. Gong, "Optimized password recovery for SHA-512 on GPUs," in *Proc. IEEE Int. Conf. Comput. Sci. Eng. (CSE) IEEE Int. Conf. Embedded Ubiquitous Comput. (EUC)*, vol. 2, Jul. 2017, pp. 226–229.
[32] R. Saputra and B. Noranita, "Analysis of GPGPU-based brute-force and dictionary attack on SHA-1 password hash," in *Proc. 3rd Int. Conf. Informat. Comput. Sci. (ICICoS)*, Oct. 2019, pp. 1–4.
[33] W.-D, Qiu, Z. Gong, Y.-D. Guo, B.-Z. Liu, X.-M. Tang, and Y.-H. Yuan, "GPU-based high-performance password recovery technique for hash functions," *J. Inf. Sci. Eng.*, vol. 32, no. 1, pp. 97–112, 2016, doi: 10.6688/JISE.2016.32.1.6.
[34] H. Choi and S. C. Seo, "Optimization of PBKDF2 using HMAC-SHA2 and HMAC-LSH families in CPU environment," *IEEE Access*, vol. 9, pp. 40165–40177, 2021.
[35] A. Visconti and F. Gorla, "Exploiting an HMAC-SHA-1 optimization to speed up PBKDF2," *IEEE Trans. Dependable Secure Comput.*, vol. 17, no. 4, pp. 775–781, Jul. 2020.
[36] NVIDIA. (2023). *NVIDIA Nsight Compute V2023.3.1*. Accessed: Mar. 15, 2024. [Online]. Available: https://developer.nvidia.com/tools-overview/nsight-compute
[37] N. Cooperation. (2023). *NVIDIA GPU Specs*. Accessed: Aug. 7, 2023. [Online]. Available: https://www.nvidia.com/en-us/geforce/graphics-cards/compare/?section=compare-20/

DONGCHEON KIM (Student Member, IEEE) received the B.S. degree from the Department of Information Security, Cryptology, and Mathematics, Kookmin University, where he is currently pursuing the master's degree in financial information security. His research interests include parallel programming environments, such as GPUs, post quantum cryptography (PQC), and cryptographic module validation program (CMVP).

SEOG CHUNG SEO (Member, IEEE) received the B.S. degree in information and computer engineering from Ajou University, Suwon, South Korea, in 2005, the M.S. degree in information and communications from Gwangju Institute of Science and Technology (GIST), Gwangju, South Korea, in 2007, and the Ph.D. degree from Korea University, Seoul, South Korea, in 2011. He was a Research Staff Member with Samsung Advanced Institute of Technology (SAIT) and Samsung DMC Research and Development Center, from September 2011 to April 2014. He was a Senior Research Member with the Affiliated Institute of ETRI, South Korea, from 2014 to 2018. Currently, he is an Associate Professor with Kookmin University, South Korea. His research interests include public-key cryptography, its efficient implementations on various IT devices, cryptographic module validation program, network security, and data authentication algorithms.