WPA-WPA2 PSK Cracking Implementation on Parallel Platforms

A. Abdelrahman, H. Khaled and Eman Shaaban
Computer Systems Department
Faculty of Computer & Information Science, Ain Shams University
Cairo, Egypt
aismail.abdelra@gmail.com,
heba.khaled@cis.asu.edu.eg, eman.shaaban@cis.asu.edu.eg

Wail S. Elkilani
Department of Computer Science
Faculty of Applied Computer Science, King Saud University
Riyadh, KSA
welkilani@ksu.edu.sa

Abstract*—Today, WLANs are the most outstanding systems to connect the devices of home and business environments. WPA/WPA2 PSK protocol is the dominant authentication protocol for most of WLANs, represented in small or non-professional networks. Cracking WPA/WPA2 PSK efficiency is mainly depending on the cracking speed. The shared memory parallel computing model is one of the used methods in enhancing WPA/WPA2 PSK cracking. In this paper, we will adapt our implemented single threaded cracking tool on GPU and Multi-Core processor to make use of their parallel platforms. The measurements result shows that the cracking speed is enhanced to be 16X using multi threaded code and 41x using GPU code on a normal machine parallel platforms.

Keywords*— graphics processing unit (GPU); multi-core CPU; wi-fi protected access (WPA); pre-shared key (PSK); password cracking; wireless local area network(WLAN);

I. INTRODUCTION

IEEE 802.11 standards define the specifications of MAC and PHY layer of WLAN. 802.11 security amendments present the security protocols WPA and WPA2 and the personal Pre Shared Key (PSK) mode is one of the main authentication modes. In PSK the user configures the access point (AP) with a password which will be used by client devices for the authentication process. The predefined password is used in deriving the authentication keys and the authentication keys are used in deriving the encryption keys. Because of the easy steps of WPA/WPA2 PSK authentication, it became the most used authentication method and cracking PSK password became a very important vulnerability study.

Three main attack types can be applied on PSK password; Brute Force, Dictionary and Rainbow Tables. Brute Force attack is based on trying all the possible password permutations with the calculation steps which will be declared in the following sections to find the real used password. This method has a very high time cost factor because of the number of the possible permutation. In PSK mode the password characters can be any character of the ASCII 95 characters, the valid password length is from 8 to 63 characters. So, the minimum number of permutations to in case of cracking a password with the minimum allowed length is 6,634,204,312,890,625 which will take unacceptable time based on any reached cracking speeds [1-3]. Dictionary attack looks like the brute force attack but it depends on the most used passwords listed in a file instead of trying non needed permutations in brute force. The dictionary size can reach gigabytes. Many fields of science can help in building the cracking dictionaries like AI and social engineering [2-3]. Rainbow Tables attack is based on building an offline calculated database of PMK (Pairwise Master Key) keys which will be illustrated in the following sections. PMK calculation is the most expensive operation in terms of the needed steps, as it is composed of hashing the fetched password and a commonly used SSID for 4096 [2][4]. In this paper, our cracking implementation is depending on Dictionary attack. Dictionary attack is mainly depending on the speed of trying the maximum number of pre shared keys\passwords per second (PSK\Sec), using parallel platform presents a great edge in enhancing the cracking calculation.

The new parallel programming models are categorized to two main types Shared Memory and Distributed Memory. In the shared memory model, the computing nodes can be multiple processors or single processor of multiple cores. Those computing nodes are sharing a global memory. In Distributed Memory the architecture is composed of separate computing nodes and each one has its own memory and address. The communication between the nodes is handled by interconnect network. In this paper we use the shared memory model [5]. We can divide the shared memory model to two trends Multi-Core Trend and Many Threads Trend. The Multi-Core trend keeps enhancing processing speed of sequential programs besides migrating to multiple cores architecture. For example, Intel® Core™ i5-750 Processor has 4 cores. The Many Threads trend aims to enhance the throughput of the parallel application. For example, NVIDIA GTX680 graphics processing unit (GPU) has 16,384 threads [6-7].

Based on the understanding of the single-threaded implementation of our WPA/WPA2 PSK cracking tool, adapting the implementation on any parallel platform is very easy after understanding the platform architecture. Shared Memory concepts can be implemented using Pthread and OpenMP on Multi-Core processors, CUDA on GPU [5]. The rest of this paper will have the following structure. Section 2

talks about the previous related work. Section 3 talks about the structure of our implemented single-threaded WPA/WPA2 PSK tool. Section 4 illustrates how the cracking implementation is adapted on GPU platform. Section 5 illustrates how the cracking implementation is adapted on Multi-Core platform. Section 6 clarifies the obtained results of different implementations and our work next steps.

II. Relative Work

PSK cracking is mainly adapted on the following platforms GPU, Multi-Core CPU, FPGA and Cell BE. In [1], Tyler used the FPGA in cracking WPA\WPA2 passwords. The working platform is Convey's HC-2, with 2 different processors. The first processor is Intel x86 host processor. The second processor is the co processor, contains four Virtex-5 LX330 configurable FPGAs with 150MHZ and 300MHZ supported clocks. In [8], Markus introduced new optimized and fully pipelined FPGA WPA2 cracker to be compared against the top marketed FPGA WPA2 Elcomsoft cracker and GPU cudaHashcat crackers. In [9], Martin used Cell BE to perform many experiments to show how this platform can enhance the results of WPA/WPA Cracking. In [10], Khai used CPU (Intel Core i5 430m) and GPU (ATI Mobility Radeon HD 5470) to evaluate a bench mark on Elcomsoft and Pyrit tools. In [11], Visan performed a comparison between Aircrack on CPU Intel Core i7 and Pyrit on GPU NVIDIA GeForce GTX 280. In [12], Mathew used Pyrit with NVIDIA GeForce 9400 GT GPU and AMD Phenom II X4 840 CPU to measure the speed of 4091 keys dictionary. In [13], Thomas performed a bench mark using Pyrit V288 to build a database with pre computed PMKs on Amazon "Cluster GPU instance" and "Cluster Compute instance". In [14], Liang introduced a new method of PSK cracking based on parallel random search over GPU platform.

It can be noticed from the listed related work that few contributions are based on implementing their own cracking steps, many of them are depending on using a ready implemented online hacking tools without being exposed to the technical details of the cracking steps. Beside our edge in implementing the cracking steps ourselves, our work has a unique edge represented in understanding relation between the cracking steps input and 802.11 standards. All the contributions are concentrating on the cracking steps with stubbing how the input parameters are sent inside the raw frames in the air. To gain this unique edge, we study the 802.11 MAC layer structure and the EAP authentication protocol to get the knowhow of extracting the cracking steps inputs from the raw exchanged data between the AP and the client without depending on ready implemented tools. This edge is located and discussed in "Parsing CAP File" in the following section.

III. Single Thread Cracking Implementation

WPA/WPA2 PSK authentication standards are based on Data-Link layer, where the data is exchanged over 802.11 MAC frames in a process called Four Way Handshake. As illustrated in **Fig. 1,** the Four Way Handshake information is exchanged over four messages in the packet body part of EAPOL-Key frame. In Message-1, the AP sends a random generated number ANonce to the client. Then the client generates a random number SNonce to derive the PTK (Pairwise Transient Key) using the predefined user password. In Message-2, the client sends the generated SNonce and uses KCK (Key Confirmation Key) the first 16 bytes of PTK to calculate and send the MIC of the 2nd EAPOL-Key frame. Then the AP derives the PTK to validate that Message 2 MIC is calculated from the user defined password. In Message-3, The AP sends the Anonce, Information Elements which may contain the derived GTK if necessary and the MIC of the 3rd EAPOL-Key frame. Then the client calculates its own MIC of Message-3 to validate that the AP is a trusted point. In Message-4, the client acknowledges that the temporal keys are installed.

Our single-threaded implementation is illustrated in Fig 2 based on Dictionary Attack. The implementation is based on a passwords dictionary file and a CAP file of a completed Four Way Handshake authentication. The first step is processing the CAP file to extract the needed input parameters for the cracking. The second step is to calculate the PMK key using the extracted input parameters and a fetched password from the used dictionary. The third step is calculating PTK key from the PMK key. The fourth step is to calculate the EAPOL-Key frame MIC of the third message. The last step is comparing the calculated MIC with the extracted MIC from step 1.

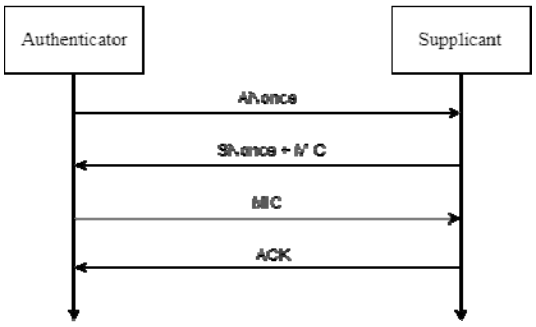

Fig. 1. Four Way Handshake Process

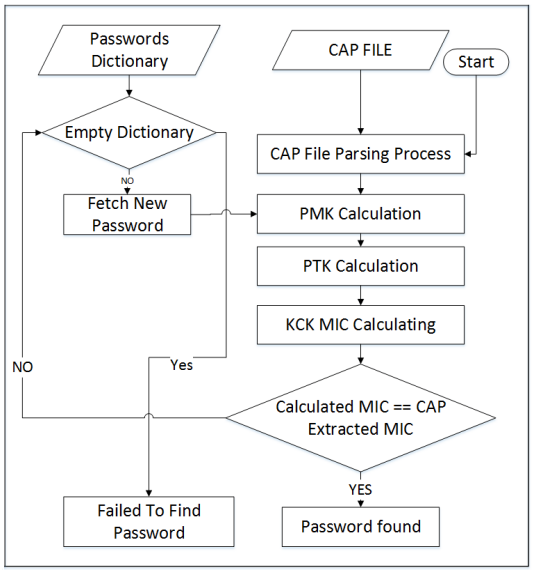

Fig. 2. Single Thread Cracking Phases

A. Parsing CAP File

After applying a De-authentication attack with a tool like Aireplay-ng, we can save the re-authentication process between the client and the AP (containing the cracking input parameters) using any packet capturing tool as Airodump-ng. The raw data of the re-authentication process can be saved in a packet capture file (CAP File) and can be browsed by Wireshark or any text editor program.

In this phase the implementation is depending on understanding 802.11 MAC layer frame standards and it's relation with the authentication process based on EAP (Extensible Authentication Protocol) to extract the input parameters. The input parameters are SSID (Service Set Identifier) representing WLAN network name, Bssid (Basic Service Set ID) represents MAC address of AP, client MAC address, EAPOL-Key frame, MIC from the third Eapol-Key, key version from the EAPOL-Key frame, Snonce and Anonce. We depend on the third message in the Four Way Handshake to get the EAPOL-Key frame and MIC, as sending the third message from the AP confirms that the client has the right user pre-defined password. The input parameters can be found in the Four Way Handshake except the SSID. The SSID can be extracted from the exchanged information between the AP and the client before the authentication process over the following MAC frames types Beacon, Probe Request, Association Request and Association Response.

B. PMK Calculation

PMK is a DK (Derived Key) calculated from the user password in WPA\WPA2 PSK using the following function [1]

$$PMK=PBKDF2\ (Passphrase, SSID, 4096, 256) \quad (1)$$

PBKDF2 (Password-Based Key Derivation Function 2) in rfc2898 is used where the initial key is the user pre-defined password, the used salt is SSID, calculation iteration number is 4096 and the derived key (PMK) length is 25bits. HMAC-SHA1 is the used underlying function.

C. PTK Calculation

PTK is an expansion of PMK to derive several keys from PTK using PRF (Pseudo Random Function) as following [15]:

$$PTK= PRF\text{-}512(PMK, \text{"Pairwise key expansion"}, MAC1||MAC2||Nonce1||Nonce2) \quad (2)$$

MAC1 is the numeric smaller MAC from client and AP MAC addresses, MAC2 is the bigger one. Nonce1 is the smaller number from Anonce and Snonce, Nonce2 is the bigger one.

D. MIC Calculating and obtaining the Password

The KCK is used to calculate the MIC field of the real fetched EAPOL-Key frame sent over the third message, and then the calculated MIC is compared to the fetched MIC from the real Four Way Handshake. If the two MICs are identical then the dictionary used password is the real user used password. The MIC calculating hash function is depending on Key Version parameter inside EAPOL-Key frame,

If 1: $MIC = HMAC\text{-}SHA1(KCK, EAPOL\text{-}Key\ Frame)$ (3)
If 2: $MIC = HMAC\text{-}MD5(KCK, EAPOL\text{-}Key\ Frame)$ (4)
In both versions the HMAC Key is KCK input to be hashed is the EAPOL-Key frame after setting it's MIC field with 0s.

IV. CRACKING IMPLEMENTATION ON GPU

The main building block of Nvidia GPU is a row of streaming multiprocessors (SMs). Each SM consists of number of streaming processors (SPs) sharing control logic and instruction cache. All SMs share multiple gigabytes of GDDR DRAM called Global Memory, which can be used for general computing as an off ship memory. The GPU device is linked with the CPU through PCI-Express interface [6]. CUDA C is an extension of standard C, It enables the programmer to harness the computation power of the CPU and GPU from one application program. A typical computing platform consists of CPU called Host and Cuda GPU called Device. Cuda program structure is a mirror of the platform structure. Two main parts compose the program and each part has its own DRAM memory. The first part runs on the host CPU and implemented with standard C and the second part runs on one or more GPU devices and labeled with Cuda keywords. On the device part there are 3 layers of the memory to be accessed from the kernel. Local Memory is a private memory assigned and available per each thread. Shared Memory is assigned and visible for all threads inside each block. Global Memory is assigned and visible for all the device threads [16].

The main building block of The CUDA program is the Kernel; it is a C function runs on the GPU labeled by Cuda extended keywords. Any kernel can run by X numbers of Cuda threads in parallel with unique thread ID [16].

Using the thread ID we can form one, two, or three dimensional block of threads. We can use the same idea to use more than one block of the same thread to form one, two, or three dimensional grid of blocks as shown in Fig 3. At any certain point of time any block can be assigned to any available single SM, therefore the number of threads per block is limited as they will share the limited resources of the SM. The number of threads per block is depending on the GPU family and version, normally it reaches 1024 [16].

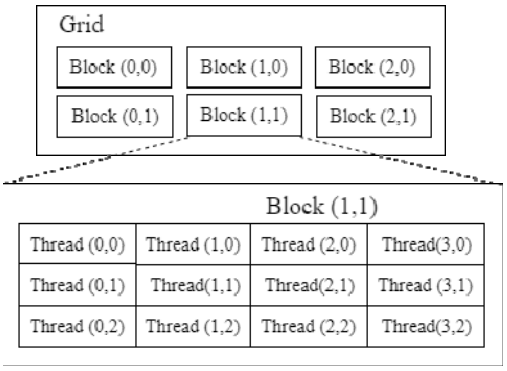

Fig. 3. Cuda Program Thread, Block and Grid [16]

The block is assigned to only one SM. More than one block can be assigned to same SM. Each SM may handle hundreds of threads concurrently based on SIMT Architecture (Single Instruction Multiple Threads). The SM execute warps, warp is groups of 32 threads to be executed in parallel. All the warp threads start to be executed together from the same kernel line by the warp scheduler. Each warp thread has its own program address, registers and local memory to execute independently [16].

The CPU starts then it launches the kernel with the Grid configuration on the host. When all threads are done, The CPU part continues its execution until another kernel is launched on the host. So the host and device execution don't overlap [6].

Our cracking implementation on the Single thread CPU is adapted to make use of the discussed GPU parallel platform. Only the keys calculation parts are what is moved to the GPU side due to the computation complexity of this part. As declared in Fig 4, the host part of the implementation is illustrated in the left and the device part is illustrated in the right. Generally, The CPU part is mainly responsible for extracting the input parameters to be exported to device global memory, then it calculates the optimized number of blocks to be launched on the device. The GPU part is responsible of the keys calculation for each passed password.

The flow of Cuda Implementation is like the single thread implementation the different functions in Fig 4. are:

"Prepare for PTK", for all the dictionary passwords the input of PTK are the same excepting the PMK part as following:
("*Pairwise key expansion "||MAC1||MAC2||Nonce1||Nonce2)* It can be calculated for one time in CPU instead of calculating it for each password.

"Prepare GPU Kernel Inputs" and "Global Memory Initialization", those functions perform the main steps of initializing the GPU with the Input parameters to enable the GPU threads which will be launched to go through the keys calculation. The inputs are allocated in GPU global memory to be visible for all working threads representing dictionary passwords, SSID, prepared common PTK input, EAPOL-Key frame, captured MIC and key version.

"Calculate Blocks Number", this function is responsible for calculating the optimized number of blocks to be launched on the GPU, taking into consideration that the blocks will be executed in warps of 32 threads.

"Launch Cracking Kernel", the CPU launches the key calculation Kernel on the GPU with the calculated configuration in the previous step.

The GPU is mainly used to execute the key calculation cracking kernel. For the launched Kernel, each thread will copy the needed input from the global to local memory then it will begin in key calculation using the fetched password. If the fetched password calculates the same MIC of EAPOL-Key frame, then the password is found.

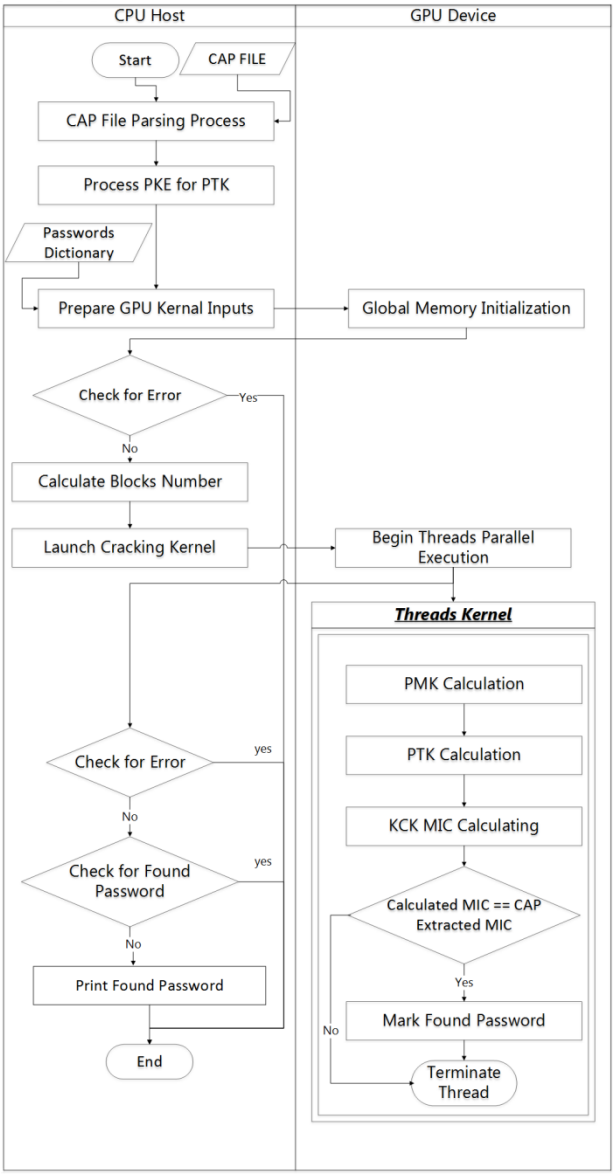

Fig. 4. Cracking Implementation on CUDA

V. CRACKING IMPLEMENTATION ON MULTI-CORE

The CPU is the key player of the processing performance. As shown in Fig 5 It contains mainly the Arithmetic Logic Unit (ALU) to handle the real calculations, Control Unit to fetch and execute the instructions and Input\Output ports for the communication with external chips and the user.

The first single core design operates on single instruction at once, where the memory feed the CPU with the instructions one by one and each instruction operates on single data. This design is called Single Instruction Single Data (SISD).

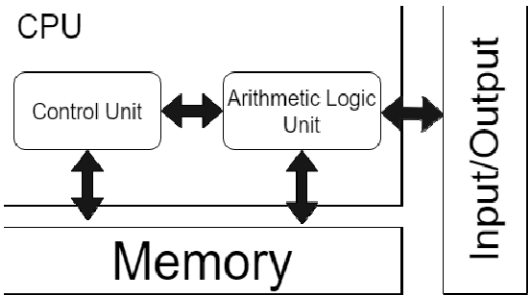

Fig. 5. CPU Architecture [17]

For long time this single core simple design was developed and enhanced in a lot of ways by making use of the clock freq increase. As the transistors capacity of the computer chips doubled nearly each two years. The main improvement methodologies are mainly represented in Pipelining, Super Scalar Architecture, Vector Processors (Single Instruction Multiple Data) and Caches. After many decades, those improvements begin to face some issues and limitations because the added complexity in the HW creates obstacles to the SW to use the added features, the clock frequency increase raise critical problem with power consumption, Moore's Law became very near to hit its end [17].

As a result, the Multi-Core solution is introduced based on the idea of decreasing the power consumption of the clock. In 1980, Shared memory architecture between separate processors was introduced. Later on, a similar concept is used to introduce a solution for the single core improvement limitation by duplicating logic parts of the processor on the same chip to create cores with shared memory. In Multi-Core design, cores can execute the code concurrently as if they were different processors on one chip sharing the same memory. In modern CPUs the pipeline design became very complex because of a lot of factors like slow memory, code flow dependencies and branching miss expectation. As a result, some of the CPU resources are idle. Multi-threaded solution is introduced for this issue. Interleaving the instructions of different programs threads can fill the pipeline gap. Intel Hyper-Threading is an example of this solution. [18].

OpenMp is compiler directives library extending C, C++ and FORTRAN to make use of shared memory structure. Using OpenMP APIs, the application can be divided into threads to be executed on multi-core multi-threaded modern structures in parallel. A lot of vendors' products and tools are compliant with OpenMp as its architecture is reviewed by the industry leading company's like Hewlett-Packard, Intel, IBM, Kuck and Associates, and Silicon Graphics [18-19][5].

We apply the facilities of OpenMP on our tool to make use of the Multi-Core parallel platform. The following flow chart in Fig. 6 describes the skeleton of our tool flow with OpenMP. In "Multi Thread Inputs Prepare", the main thread prepares the input extracted parameters to the memory for the threads. In "Begin Threads Parallel Execution", the keys cracking calculation parts is distributed over the Multi-Core available threads and each thread will fetch its own password from the Dictionary directly.

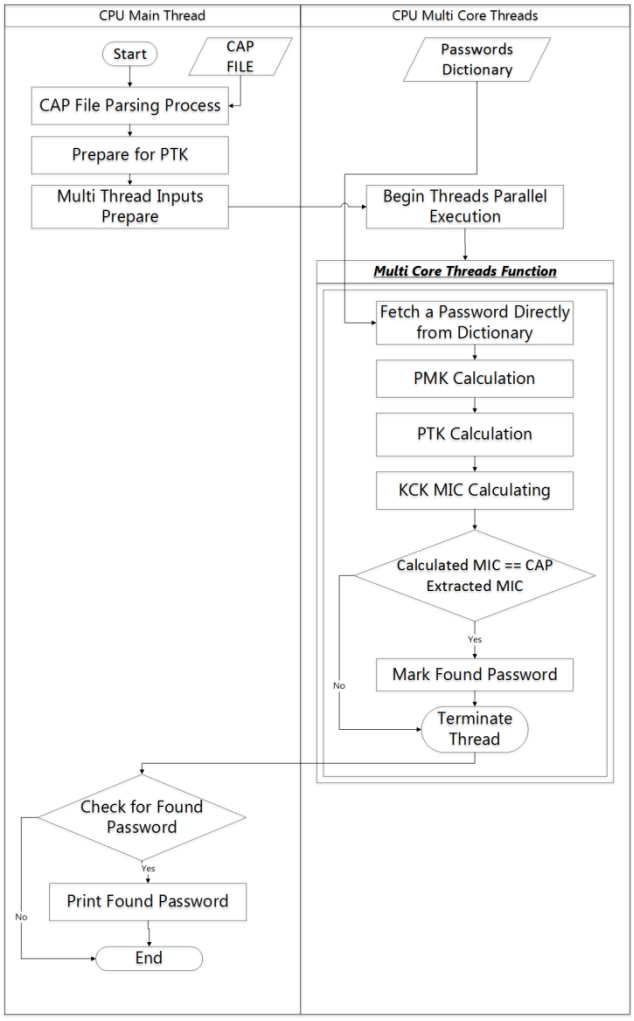

Fig. 6. Cracking Implementation on Multi-Core

VI. Implementation Results

In this part we will illustrate the environments setup and the results of single core, Multi-Core and GPU implementations.

The implementation of single thread CPU is benchmarked twice. The first time is on Windows 8 and the second time is on Ubuntu Virtual machine running over Windows 8. The installed processor is Intel i7-4710HQ CPU running on 2.50GHz with 4 cores and 8 logical cores (Threads). The implementation of Multi-Core version is benchmarked on Ubuntu 12.04 with installed processor Intel i7-2630QM CPU running on 2GHz with 4 cores and 8 logical cores (Threads) using Eclipse IDE. The implementation of GPU version is benchmarked on Windows using the Microsoft Visual Studio C compiler and CUDA development toolkit to build the Cuda C program part. The used GPU is GeForce GTX 860M with the declared specs in Table I.

For all the test bench results discussed in Table II, testing batches are used to calculate the execution time of the keys calculation cracking executable files of the different platforms implementations. The testing batches perform 4 main steps;

TABLE I. GTX 860M SPECS

Spec	Unit
CUDA Cores	640
Core clock	1019 MHz
Memory data rate	5010 MHz

TABLE II. OUR CRACKING IMPLEMENTION RESULTS

Platform	OS	Number of Password	Time	PSK\Sec
Single Thread CPU	Windows 8.1	50000	391 seconds	127.8
Single Thread CPU	Ubuntu (VM)	120000	690 seconds	173.9
Multi-Core	Ubuntu	1000000	949 seconds	1053
GPU	Windows	1000000	384 seconds	2604

record the current start time in ms, call the executable file and pass the needed inputs, wait for the cracking result, record the stop time and calculate the cracking consumed time.

It's very clear the contribution of the parallel platforms in speeding up the cracking calculations as shown in Table II achieves the fastest results; the Multi-Core platform is slower than the GPU but faster than the single thread CPU implementation.

In order to make a comparison of our work and instances of the field related work, we try the same cracking inputs on AC1.2 RC4 to be compared with our Single Thread CPU and Multi-Core implementations, as there are no published academic implementations for single thread implementation and it's relative Multi-Core implementation. Regarding the GPU implementations it's compared to the research implementation presented in [14].

For the Single CPU comparison, we import the source code of Aircrack (AC1.2 RC4) in Eclipse IDE on Ubuntu Virtual machine. We make use of the virtual machine which is running on one processor to force Aircrack to run over single thread then we disable the SSE2 assembly implementation from the project make file. By providing Aircrack with 120000 password dictionary the obtained time is 273 seconds, which means that our implementation takes 1.57 Aircrack time.

For the Multi-Core comparison, we move the Aircrack execution to Ubuntu machine. Using Eclipse IDE, AC1.2 RC4 is compiled without SSE2 assembly implementation. The main difference here between our implementation and AC1.2 RC4 is the use of OpenMp instead of Pthead for multi threading. By providing Aircrack with 1000000 password dictionary the obtained time is 851 seconds, which means that our implementation takes 1.1 Aircrack time. So our Multi-Core giving better results than single thread CPU implementation.

In [14], an experiment is performed to crack WPA\WPA2 PSK on GPU using a proposed random search method. When applying 600000 passwords the execution time reaches 2000 sec about 300 PSK on NVIDA Geforce GT420. By taking into consideration the difference in the used GPUs, we can expect that we are the same level of speed.

After the deep understanding of WPA/WPA2 PSK cracking flow and studying the related 802.11 standards in, the next step in our work is to optimize our own implementation on the different platforms. Beside the optimization we will make use of our work to study the new WPA3 and its authentication vulnerabilities when it became available in the market.

REFERENCES

[1] T. B. Johnson, "An FPGA architecture for the recovery of WPA_WPA2 keys, Master of Science Thesis, Iowa State University, 2014.

[2] K. T. Viet, "GPU - accelerated WPA PSK cracking solutions", Minnesota State University, 2010.

[3] M. Daniel, "WPA password cracking Parallel Processing on the Cell BE", Master thesis, Aalborg University, 2009.

[4] S. A. VISAN, "WPAWPA2 Password Security Testing using Graphics Processing Units", Journal of Mobile, Embedded and Distributed Systems, vol. V, no. 4, pp. 167-174, 2013.

[5] H. Kasim, V. March, R. Zhang, and S. See, "Survey on parallel programming model", Conference Network and Parallel Computing (NPC '08), vol. 5245. , Springer; 2008. p. 266–75.

[6] D. Kirk , W. W. Hwu, "Programming Massively Parallel Processors: A Hands-on Approach", Elsevier Science, 2010, 496 pp.

[7] Wikipedia , General-purpose computing on graphics processing units , [&%https://en.wikipedia.org/wiki/General-purpose_computing_on_graphics_processing_units], August 26, 2018.

[8] M. Kammerstetter, M. Muellner, D. Burian, C. Kudera, and W. Kastner, "Efficient HighSpeed WPA2 Brute Force Attacks Using Scalable LowCost FPGA Clustering", Cryptographic Hardware and Embedded Systems – CHES 2016, Vol. 9813, Santa Barbara, Springer, pp. 559–577,2016.

[9] M. Daniel, "WPA password cracking Parallel Processing on the Cell BE", Master thesis, Aalborg University, 2009.

[10] K. Tran, "GPU - accelerated WPA PSK cracking solutions," Minnesota State University, 2010.

[11] S. Visan, WPA/WPA2 Password Security Testing using Graphics Processing Units. Journal of Mobile, Embedded and Distributed Systems, 2013.

[12] M. Wolfe, "Vulnerability of wireless network security due to parallelized brute force attacks," in Proceedings of the 12th Winona Computer Science Undergraduate Research Symposium,Winona, USA,pp. 559–577, 2012.

[13] T. Roth, "Breaking encryptions using GPU accelerated cloud instances", Black Hat Technical Security Conference, USA, 2011.

[14] L. Ge, L. Wang, L. Xu, and S. Yang, "Password recovery for WPA/WPA2-PSK based on parallel random search with GPU", Wireless Communications, Networking and Applications. Lecture Notes in Electrical Engineering, vol 348. Springer, New Delhi, pp. 1149–1159, 2016.

[15] IETF, PRF, https://tools.ietf.org/rfc/rfc4868.txt, Accessed August 26, 2018.

[16] NVIDIA. "CUDA C PROGRAMMING GUIDE : Design Guide ", 2017.

[17] G. Hager, G. Wellein, "Introduction to High Performance Computing for Scientists and Engineers", Chapman & Hall/CRC Press, 2010.

[18] B. Chapman, G. Jost and R. van der Pas, "Using OpenMP: Portable Shared Memory Parallel Programming", MIT Press, Cambridge, 2007.

[19] L. Dagum and R. Menon, ''OpenMP: An industry standard API for shared-memory programming,'' IEEE Comput. Sci. & Eng., vol. 5, no. 1, pp. 46–55, 1998..