Distributed WPA Password Cracking System Exploiting Amazon GPU Clusters

Chaodong ZHENG
National Univ. of Singapore

Chengwen LUO
National Univ. of Singapore

Kartik Sankaran
National Univ. of Singapore

April 14 2012

Abstract

WPA is a widely used security protocol to protect the wireless communications. Attackers have been working for years to try to crack the WPA password, however there's no known algorithm vulnerability can be exploited in this process. Brute force/Dictionary attack is considered to be the only available attack today to crack the WPA password.

In this work, we design and implement the distributed WPA cracking system that can be easily scale and exploit the GPU computation power to greatly accelerate the WPA password cracking process. The system exploits the Amazon EC2 GPU cluster. We believe that due to the efficiency of this attacking system, WPA users are strongly advised to use strong passwords. Countermeasures to this attack remain an open topic.

Keywords

WPA **Password Cracking** **Distributed System** **GPU Computing**

1. Introduction

Introduced as an interim security enhancement over WEP while the 802.11i wireless security standard was being developed, WPA is a widely used secure protocol to protect the wireless networks. Most current WPA implementations use a PSK (preshared key), commonly referred to as *WPA Personal*, and the MIC (message integrity code) for authentication. The WPA protocol affords users with vastly stronger security than the WEP protocol. Today most of the attacks trying to break WPA are inefficient brute force/dictionary attacks. Success of the attack depends on the computation power of the attacker's machine and the strength of the password. If the password is strong enough, it can take even years to crack the password.

The authentication process leaves two considerations: the access point (AP) still needs to authenticate itself to the client station (STA), and session keys to encrypt the traffic need to be derived. Therefore the four-way handshake is used to establish another key called the PTK (Pairwise Transient Key). The PTK is in turn derived from a Pairwise Master Key (PMK). The ESSID and WPA Password are used to calculate the PMK. As shown in Figure 1, the PTK is generated by concatenating the following attributes: PMK, AP nonce (ANonce), STA nonce (SNonce), AP MAC address, and STA MAC address. The product is then put through a cryptographic hash function. All successful brute-force attacks need to capture the four-way handshake packets, which includes the ANonce, SNounce, and MIC [1]. Attackers keep calculating PMK and MIC using randomly chosen passwords. Once the calculated MIC matches the MIC in the packet, the password chosen is the real WPA password and the attack succeeds.

In the process, the most time-consuming step is the calculation of the PMK from the ESSID and Password, since it involves 8192 rounds of SHA1 calculation.

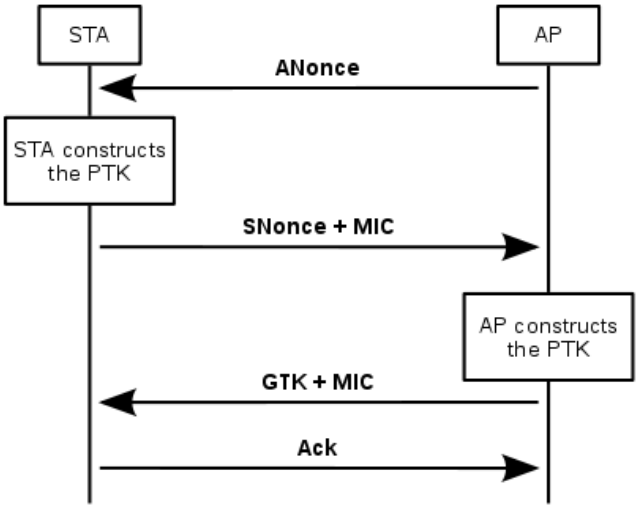

Figure 1. WPA Four-way Handshake

To accelerate the WPA cracking, one possible way is to exploit the great computation power of GPU computation. GPGPU is the means of doing General Purpose computation (traditionally handled by the CPU) on a Graphics Processing Unit (GPU). GPUs are designed as computational accelerators or companion processors optimized for scientific and technical computing applications. Architecture of GPU and CPU are highly different. GPU is specialized for compute-intensive, highly parallel computation and therefore designed such that more ALUs are devoted to data processing rather than data caching and flow control, as schematically illustrated by Figure 2.

More specifically, the GPU is especially well-suited to address problems that can be expressed as data-parallel computations – the same program is executed on many data elements in parallel – with high arithmetic intensity – the ratio of arithmetic operations to memory operations. There's a lower requirement for sophisticated flow control due to the fact that same program is executed for each data element, and because it is executed on many data elements and has high arithmetic intensity, the memory access latency can be hidden with calculations instead of big data caches [2].

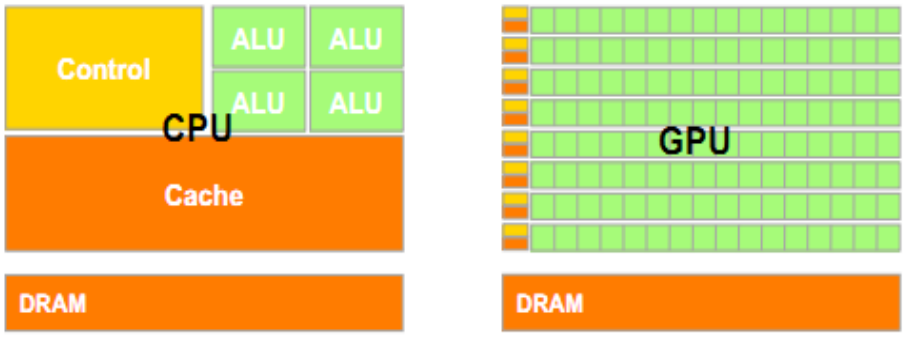

Figure 2. Architecture Difference Between GPU and CPU

Efficiency of WPA brute-force/dictionary-brute greatly depends on the computation power of the system. GPU computation increases the efficiency by exploiting compute-intensive architecture. However it still takes long time for single machine to crack the WPA protocol. As end systems are becoming cheaper, attackers start trying to build the distributed cracking systems which exploit large scale of parallel-working machines. Development of cloud computing in recent years provides an ideal way to scale the computation by dynamically allocating computing instances. One most successful cloud computing platform is known as Amazon EC2 [7]. Amazon Elastic Compute Cloud (Amazon EC2) is a web service that provides resizable computation capacity in the cloud. Amazon EC2 GPU clusters allow crackers to scale the cracking program and exploit the GPU computation power. Figure 3 below illustrates the cloud architecture of Amazon EC2.

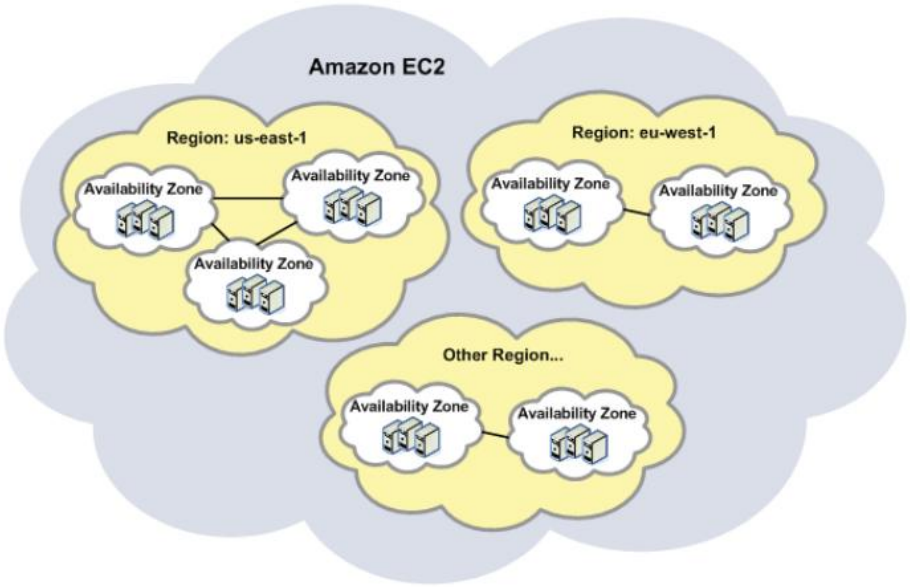

Figure 3. Amazon Elastic Compute Cloud

Amazon EC2's provides three different kinds of instances: On demand, Reserved and Spot instance. These three instances differ in the availability and cost. Spot instance can be created during runtime and through Amazon API and the cost is based on the current market price. Based on the property of our cracking system, spot instance provides the most flexible and cost efficient way to use the Amazon EC2 GPU cluster.

2. Motivation

WPA password cracking has long been a hot topic in the wireless security research. Goal of analyzing such vulnerabilities and cracking approach is to understand the problems better and come up with countermeasures to protect the wireless communications. Due to the reason that WPA password cracking process is highly parallelizable, and automatically scales with number of cracking processes, it is possible to build an efficient and automatically scale cracking systems with powerful computation powers easily. Our work exploits this property and explores the possibility of building such systems.

To automatically scale the cracking process, we exploit cloud computing to simplify system architecture and system implementation. Amazon EC2 cloud platform and provides GPU clusters that allows us to launch multiple GPU instances on demand at low cost. AWS API also provides easy way to manage them. Using EC2 GPU clusters we can scale our cracking process to hundreds or even thousand parallel compute tasks, thereby accelerating the cracking process.

Our work exploits Amazon GPU cluster instead of traditional CPU cluster for the reason that their properties differs a lot and GPU architecture is much more suitable for our system. For compute-intensive computations, GPU has far greater computational power than CPU. Figure 4 illustrates their different performances doing compute-intensive computations. GPU architecture can easily exploit parallel computing for large data computation.

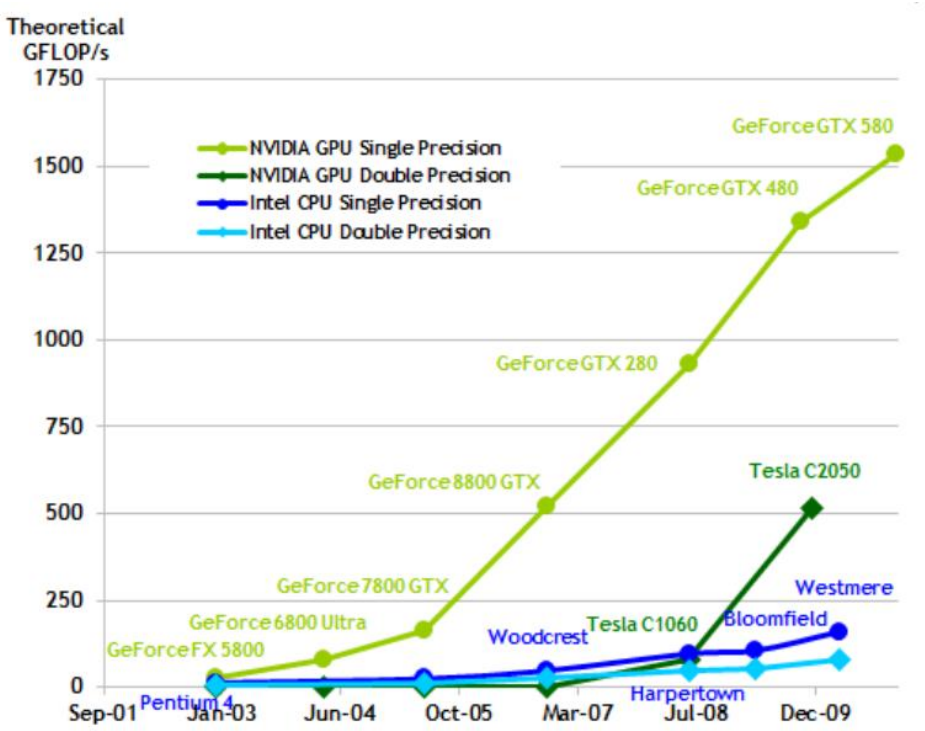

Figure 4. Computation Power Comparison Between CPU and GPU

3. Related Work

It has been known for quite some time that GPU can outperform CPU when it comes to works that are highly parallelizable. However, there are few tools which utilize this fact and apply it to the task of IEEE 802.11 WPA/WPA2-PSK passphrase cracking. Among the existing proposals, pyrit is the most popular and powerful one. Pyrit [3] is a free tool that allows creating massive databases, pre-computing part of the WPA/WPA2-PSK authentication passphrase in a space-time-tradeoff. It exploits the computational power of multi-core CPU platforms and other GPU platforms such as ATI Stream, Nvidia CUDA and OpenCL. According to its authors, pyrit also allows multiple computational nodes to connect through a network to commence the cracking process together. (This is achieved by utilizing pyrits RPC interface). Pyrit is written in Python and C, along with some code for each specific platform (For example, CU code for CUDA).

In recent years, with the emergence of cloud computing, people can have easy access to a huge amount of computational power at very low cost. This has inspired the business idea of operating websites to provide best effort WPA/WPA2-PSK passphrase cracking service at around 20 USD per request, which behind the scene actually utilizes Amazons cloud computing facility. Please pay attention to the phase “best effort”, this means such websites do not provide any guarantee that the correct passphrase will be found, as cracking WPA/WPA2-PSK is itself by brute force while the potential key space is infinite. These websites do not reveal many details regarding how the cracking process actually proceeds; neither do they release the tools they use.

More recently, a German security researcher Thomas Roth has developed a framework for distributed password cracking with emphasis on WPA/WPA2-PSK and Amazon EC2. He achieved the speed of validating about 22,500 PMKs per second per dollar for cracking WPA/WPA2-PSK passphrase. He intended to release the source code. However, due to various reasons, the source code is still not available till now. More information can be found in his presentation slides at the 2011 Black Hat Technical Security Conference (Europe) [4].

The tools mentioned above have limitations - (1) Most existing WPA cracking tools (Eg: aircrack [5]) only exploit CPU cores, and take a long time even for simple passwords. (2) Other existing tools (Eg: pyrit) can exploit GPU, but the computation is only limited to one GPU, and cannot automatically scale to multiple GPUs. (3) Trying to build a distributed system by purchasing the hardware to work in parallel to scale the cracking process is too costly.

Our tool allows us to launch multiple GPU instances on the cloud and scale easily, accelerating the cracking process at low cost.

4. System Architecture

As shown in Figure 5, the entire system includes one master node and multiple slave nodes. Master node runs the master program and slave nodes run the slave program. The user interacts with the master node. More specifically, the user will specify the WPA/WPA2-PSK 4-way handshake packet, the ESSID of the access point, the password search space, and number of Amazon EC2 instances they want to launch. The master node, which runs the master program, will then launch the instances (and hence also the slave program running on the slave nodes), splitting the validating tasks accordingly.

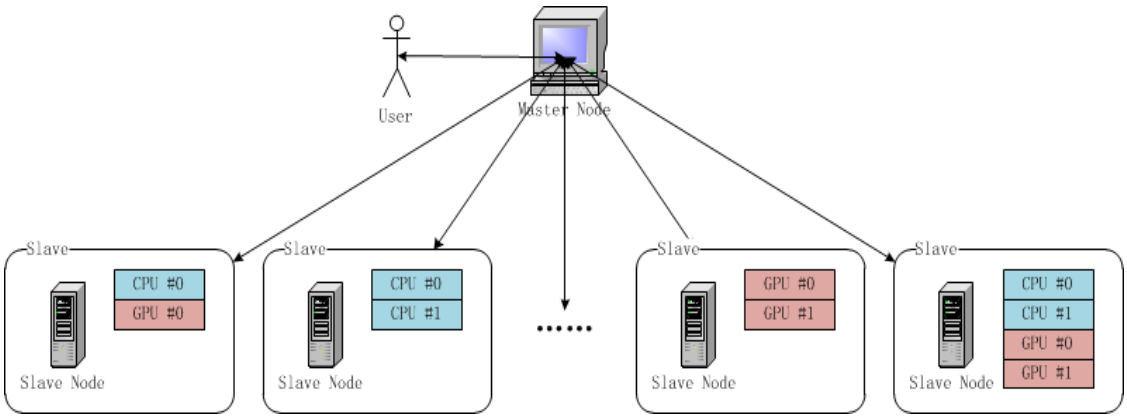

Figure 5. System Architecture

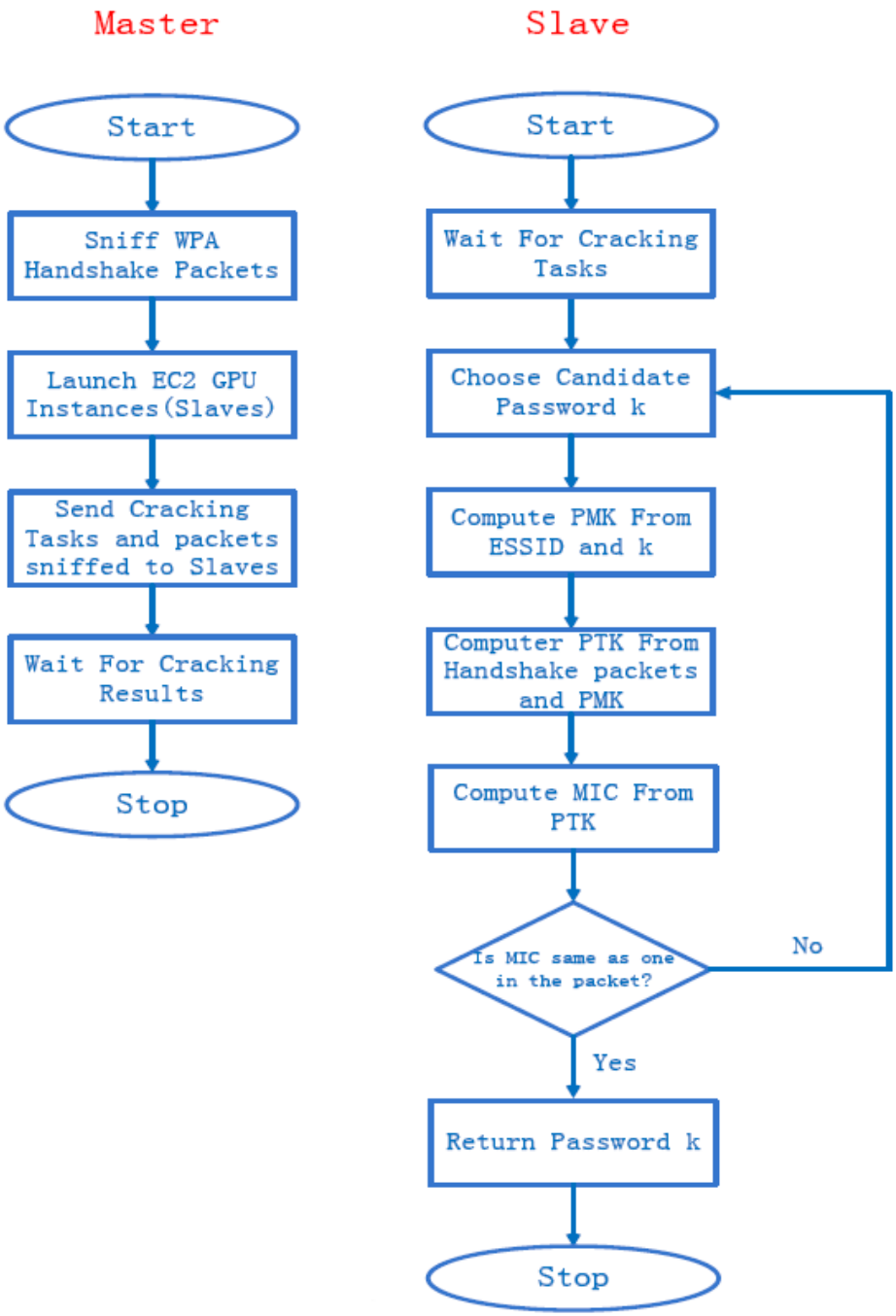

Figure 6. Program Flow Chart

The master program also passes the WPA/WPA2-PSK 4-way handshake packet, the ESSID of the access point and the (split) key space to each slave program. The slave program, on the other hand, runs on various types of Amazon EC2 instances. The slave program receives information and cracking task from the master program via socket communication. It then utilizes all present resources (i.e. CPU and GPU) to validate each key in the assigned key space. Once the correct key is found, it sends the correct key back to the master program, which records the key, and terminates all Amazon instances.

We now describe the master program and slave program in more detail (see Figure 6). Upon initialization, the master program reads capture files from disk which contain the WPA/WPA2-PSK 4-way handshake packet and ESSID of the access point. It also reads command line arguments to determine the key space the user want to validate, and the type and number of Amazon EC2 instances the user wants to start. The master program then uses Amazon's AWS SDK to boot the corresponding EC2 instances.

Upon successful booting, the master program will once again utilize AWS SDK to bind static public IPs to the allocated instances so that they can be accessed easily. The next step is to connect to the slave programs running on these instances using socket. Finally, the master program sends the WPA/WPA2-PSK 4-way handshake packet, the ESSID of the access point and the (split) key space to each slave program. The master program maintains the socket unless the slave program closes it, as the slave program utilizes it to send back the correct key.

Upon initialization, the slave will bind to a local port and wait for the incoming socket connection from the server program. Once this connection is established, the slave program will receive the above described information from the master program. The slave program will then detect the number of CPU and GPU present and start one cracking thread for each CPU core and GPU device. Within each thread, there is a loop. In one iteration of the loop, one (if the thread is run on a CPU core) or multiple (if the thread is run on a GPU device) keys will be tested. More specifically, the test process works in the following way: (a) for each key *k* that is going to be validated, the pairwise master key (PMK) will be calculated according to *k* and the ESSID of the access point; (b) the pairwise transient key (PTK) is derived from the PMK and the 4-way handshake packets; (c) the message integrity code (MIC) is calculated according to the PTK and the 4-way handshake packets; and (d) if the computed MIC is identical to the one in the 4-way handshake packets, then *k* is the correct key. If the correct key is found, the slave program will send back it to the master program via the previous established socket. Otherwise, if the correct key is not found in the assigned key space, the slave program will close the socket connection with the master program and exit.

5. Implementation on GPU

WPA Password Cracking is much faster on GPU than on CPU. However, although password cracking is easily parallelizable, implementing it on GPU is not trivial. This section gives a simple overview of GPU architecture, explaining why password cracking is faster compared to CPU. Next, the steps for executing the cracking process on GPU are described. This is followed by a description of the implementation challenges.

GPU Architecture

Figure 7 shows the architecture of a GPU core in contrast to a CPU Core. Unlike a CPU core, which typically has only 1 or 2 ALUs, a GPU core has many more (8 ALUs in Figure 7(b)). This allows the GPU core to perform multiple computations simultaneously. The CPU contains control logic circuits such as “Out-of-order control” (which allows different instruction streams to execute in parallel), “Branch prediction” (which make if-else condition statements execute fast), and “Memory pre-fetcher” (which loads the data cache with frequently used data from memory). The GPU, on the other hand, does not have these.

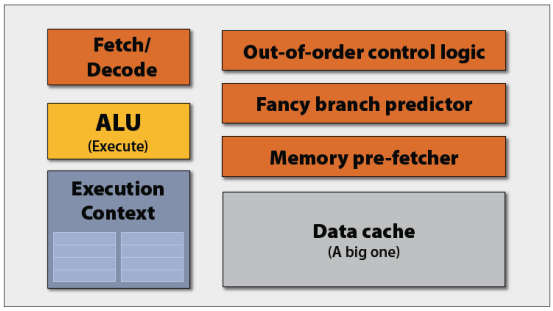

Figure 7(a) CPU Core

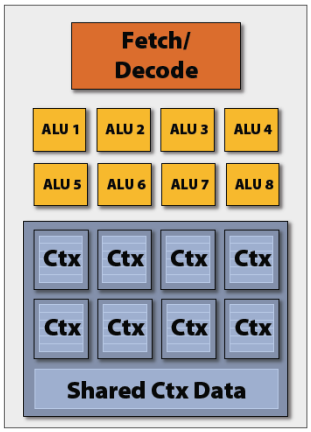

Figure 7(b) GPU Core

Since the GPU Core is smaller, simplified, and less power-hungry, it is possible to place a huge number of cores on one graphics card (448 cores in the Amazon GPU instance). The number of parallel threads that can run on a GPU is equal to the number of GPU cores times the number of ALUs per core. In the Amazon GPU instance, there are 2 GPUs, each with 448 cores, each core with 14 ALUs. So, ideally, the number of parallel computations is 2 * 448 * 14 = 12544.

GPU is suitable for programs which have a lot of computations, and lesser number of conditional statements. Each ALU on a particular GPU core executes the exact same instruction at the same time (though on different data), unlike the CPU core where different instructions can execute on different ALUs out-of-order. Data is not automatically cached in a GPU Core, the programmer must explicitly copy it.

Password cracking is an inherently compute-intensive process with few conditional statements, and hence is ideal for running on a GPU. Due to the larger number of parallel computations possible on GPU, it runs much faster compared to CPU.

Execution Steps

Password cracking on the GPU consists of the following major steps –

1. Pre-compute the first-round SHA1 for 50,000 passwords on the CPU
2. Copy the 50,000 hashes to the GPU memory from the Host Memory
3. Execute the next 4095 * 2 SHA1 rounds on each hash on the GPU to get 50,000 PMKs (i.e. 4095 * 2 * 50000 SHA1 hash computations are done on GPU in parallel)
4. Copy the results from GPU Memory back into Host memory
5. Calculate the MICs using each of the 50,000 PMKs, and check if any of them match the MIC in the sniffed packet. If so, the password has been found. Otherwise, repeat the same procedure for another 50,000 passwords.

The most time consuming is Step 3. Pre-computation and MIC verification is done on the CPU so that the code on the GPU is simplified, and does not have any branching statements.

Implementation challenges

Parallelization on GPU is more difficult compared to CPU. Since the code is running on the graphics card, and the graphics card is not allowed to access host memory, the GPU code cannot access *libc* library. In particular, the GPU cannot use the *openssl* library implementation of HMAC and SHA1. In order to compute hashes on the GPU, the SHA1 procedure has to be re-implemented.

It is inefficient to use existing C implementations of SHA1 directly on the GPU. To make the SHA1 computation efficient, 80 loop iterations have to be unrolled manually, allowing the Nvidia compiler to use the GPU registers optimally. The SHA1 rotations are also hardcoded manually instead of computing them. This makes the procedure "GPU-friendly".

APIs used

Nvidia graphics cards can be programmed in two ways – using OpenCL[6] or using CUDA[2]. OpenCL is an open specification supported by most graphics cards. CUDA is a proprietary API developed by Nvidia, which runs only on their graphics cards. Since CUDA is optimized for Nvidia, we chose the CUDA Runtime API to write our parallel implementation.

The code is written in C with Nvidia's syntax extensions. Functions executed on the GPU are tagged with `__global__` and `__device__` prefixes. The compiler generates code separately for the GPU and CPU.

6. Evaluation and Results

We tested our implementation on the Amazon EC2 platform, using the GPU cluster instances. The GPU programming was done using CUDA Tools Version 4.1. The cluster specifications are given below:

Cluster GPU Quadruple Extra Large Instance
22 GB of memory
33.5 EC2 Compute Units (2 x Intel Xeon X5570, quad-core "Nehalem" architecture)
2 x NVIDIA Tesla "Fermi" M2050 GPUs
1690 GB of instance storage
64-bit platform
I/O Performance: Very High (10 Gigabit Ethernet)

Table 1. Amazon GPU Cluster Instance

Since WPA Passwords are minimum 8 characters, people tend to use their phone number as the password, i.e. the password is all digits. In our implementation we have demonstrated password cracking for all-digit 8 character passwords.

We created a WPA2 PSK hotspot using an Android phone with an 8-digit password, and connected a laptop to it. Using Wireshark we could capture the WPA2 4-way handshake, containing the information needed for cracking. We used this capture file as our test data.

The master program automatically creates two instances of the Amazon GPU clusters. The number of instances created can be changed as required to a larger number, making the cracking process much faster. The master then allocates public IP Addresses to these GPU instances, allowing it to communicate via sockets. The slave program automatically runs on startup of the GPU instance. Depending on the number of instances created, the master program divides the

password search space equally among them (in our test, there are two instances, and hence the search space is divided into 2 parts – 00000000 to 49999999 and 50000000 to 99999999). After connecting to the slave programs, the master sends the cracking information to them.

The slave in turn subdivides the work among the CPUs and GPUs. In our test, there were 16 CPU cores, and 2 GPUs (each with 448 cores). The password search space is allotted to the CPU cores in chunks of 1000, and to the GPUs in chunks of 50,000 (since the GPUs are capable of running large number of threads). This continues till the password is found. The result is then sent back to the master program, which records the password. The master then terminates the instances and deallocates the public IP addresses.

Using our test data, we can measure the speed of password cracking in terms of PMKs computed per second. We obtained a speed of approximately 2,000 PMK/sec for CPU, and 15,000 PMK/sec for GPU, giving a total of 17,000 PMK/sec per Amazon instance. Since the cost of the spot instances is $0.6/hour, the cracking efficiency in terms of time and cost on GPU is 15000/0.6 = 25,000 PMK/sec/dollar, which is better than the efficiency achieved by Thomas Roth [4] (22,500 PMK/sec/dollar).

Using the speed, we can also calculate the approximate time and cost to crack different password search spaces. The Cost is calculated at the rate of 0.6 US Dollars/hour/instance, the current Amazon pricing.

Password Type	**Search Space Size (S)**	**Time taken using 1 GPU instance (T)**	**Cost (C)**	**Time taken using 100 GPU instances (T_{100})**
8 digits	100,000,000	1.6 hrs	$1.2	23 min
Dictionary word	250,000	14 sec	$0.6	< 1 sec
8 lower-case alphabets	26^8	142 days	$2045	1.42 days

Table 2. Time and Cost of Cracking different password types

Table 2 shows the time and cost to break different password search space sizes. In the last entry, note that even if we use 100 GPU instances instead of 1, the cost still remains roughly the same, but the password gets cracked much faster - within 2 days.

This shows how the Amazon cloud has made it possible to crack a previously difficult problem. Earlier, crackers would have to buy a large number of computers and 100 GPUs to crack passwords reasonably fast. However, this is too expensive. Using the cloud, it has now become possible to crack weak passwords at a relatively low cost of $2000. Due to the Elastic nature of the cloud, the cracker can easily create multiple instances, making the process faster, yet at the same cost.

This also demonstrates the importance of choosing strong passwords (long passwords with a mixture of digits, characters, and symbols) since the search space becomes too large even for the Amazon cloud. Weak passwords, which were previously thought as safe, and now crackable using the cloud. Choosing strong passwords is the solution to overcoming this security threat.

7. Conclusion

We developed a tool that exploits the Amazon cloud GPU cluster instances to crack WPA passwords in a scalable manner. The tool exploits the combined power of CPU and GPU. Unlike other existing tools, our work can scale to multiple GPU instances at low cost. Using our tool, we were able to crack 8 digit passwords in less than 2 hours, at a cost of less than $2. Compared to the work of Thomas Roth [4], who achieved 22,500 PMK/sec/dollar, we were able to obtain a cracking efficiency of 25,000 PMK/sec/dollar.

Our tool highlights the potential security threat by using weak passwords. Users of WPA PSK are therefore advised to use strong passwords in order to prevent this type of attack using the cloud.

8. Future Work

Our tool works only for 8 digit passwords. It can be easily extended to use a dictionary of commonly used passwords, instead of brute force searching over the entire password space.

It is also possible to pre-compute the PMKs for commonly used ESSIDs to enable fast lookup, instead of calculating them at runtime.

References

[1] IEEE 802.11i-2004
http://en.wikipedia.org/wiki/IEEE_802.11i-2004
[2] NVIDIA CUDA C Programming Guide
[3] Pyrit
http://code.google.com/p/pyrit
[4] Thomas Roth's Presentation
https://stacksmashing.net/stuff/bh_eu_2011.pdf
[5] Aircrack
http://www.aircrack-ng.org/
[6] OpenCL
http://en.wikipedia.org/wiki/OpenCL
[7] Amazon EC2
http://aws.amazon.com/ec2/