Parallelization of brute-force attack on MD5 hash algorithm on FPGA

Maruthi Gillela
Research Centre Imarat
DRDO
Hyderabad, India
g.maruthi@rcilab.in

Vaclav Prenosil
Faculty of Informatics
Masaryk University
Brno, Czech Republic
prenosil@fi.muni.cz

G. Venkat Reddy
Research Centre Imarat
DRDO
Hyderabad, India
venkatreddy.g@rcilab.in

***Abstract*—FPGA implementation of MD5 hash algorithm is faster than its software counterpart, but a pre-image brute-force attack on MD5 hash still needs 2^{128} iterations theoretically. This work attempts to improve the speed of the brute-force attack on the MD5 algorithm using hardware implementation. A full 64-stage pipelining is done for MD5 hash generation and three architectures are presented for guess password generation. A 32/34/26-instance parallelization of MD5 hash generator and password generator pair is done to search for a password that was hashed using the MD5 algorithm. Total performance of about 6G trials/second has been achieved using a single Virtex-7 FPGA device.**

***Index Terms*—LUT, HDL, GPU, IP core**

I. Introduction

MD5 message digest algorithm takes a message of an arbitrary length and computes a fixed 128-bit hash output [6]. One of the applications of this hash algorithm is to store passwords in a hashed form. The brute-force attack of pre-image type on MD5 hash needs a maximum of 2^{128} iterations which is unmanageable because of impractical runtime or cost involved. Attempts have been made to speed up the brute-force attack, for example, using GPUs (Graphics Processing Units). Another useful method is to make use of the advantages offered by hardware. Hardware offers higher performance in terms of number of trials per second.

This work attempts to increase the speed of the brute-force attack on MD5 hash using an FPGA. Pipelining of the logic improves the frequency of the operation and parallelization gives higher speed in terms of number of hash trials per second. Both of these techniques and other design & Hardware Description Language (HDL) coding techniques need to be employed for improving the speed.

Rest of the paper is organized as follows: section II briefly describes MD5 hash generation in general and using the hardware in particular. Then, section II further dwells upon various implementations and techniques for speeding up the hash generation. It also describes an earlier implementation of the brute-force attack in hardware. Section III will first present the overall architecture of our brute-force attack on FPGA. It then presents the implementation of MD5 hash generation as well as three architectures for guess password generation. It also describes factors that are worked upon to optimize the speed of password cracking. Section IV presents the results of the brute-force attack using three architectures presented in section III. This section will also compare the performance of the implementation of this work against other implementations in the literature and describe the challenges faced in parallelizing the brute-force attack. The last section will summarize the results of this work.

II. MD5 hash algorithm and brute-force attack on it

MD5 hash generation has a block size of 512-bits. Therefore, the input message bits are padded to have multiples of 512-bits in length (including a message length field of 64-bits). Padding in MD5 hash generation consists of bit 1 followed by the required number of bit 0's. In the MD5 algorithm, IV (Initial Value) and CV (Chaining Variable) which are used to compute MD5 hash are organized as A, B, C and D buffers of 32-bit each [6]. The MD5 algorithm has 4 rounds, with each round consists of 16 steps [1]. The computation in each round is given by [2]:

$$\begin{aligned} &A = B + ((A + AuxFn(B, C, D) + X_j[k] + T[i]) \lll S) \\ &A \leftarrow D, B \leftarrow A, C \leftarrow B, D \leftarrow C \end{aligned} \tag{1}$$

whereas the auxiliary function (denoted as AuxFn in (1)), one for each round, is defined as follows:

$$\begin{aligned} F(X, Y, Z) &= (X \wedge Y) \vee (\neg X \wedge Z) \\ G(X, Y, Z) &= (X \wedge Z) \vee (Y \wedge \neg Z) \\ H(X, Y, Z) &= X \oplus Y \oplus Z \\ I(X, Y, Z) &= Y \oplus (X \vee \neg Z) \end{aligned} \tag{2}$$

T[1...64] is a table of 32-bit constants constructed from sine function. S specifies the number of circular shifts (4 different values/round). X_j[1...16] holds current message block (512-bit) and it is basically an array of 32-bit length words. Each message word (32-bit) of the message block enters second, third and fourth rounds as per the following equations respectively (they enter in their original order in the first round) [6]:

$$\begin{aligned}\rho_2(i) &= (1+5i)\ mod\ 16\\ \rho_3(i) &= (5+3i)\ mod\ 16\\ \rho_4(i) &= 7i\ mod\ 16\end{aligned} \tag{3}$$

At the end, the modified A,B,C and D are added to their original values to get the final hash output.

Block RAM can be used for storing constants T and S. The auxiliary functions F, G, H and I are obtained using simple bitwise logical operators. The implementation also requires 32-bit adders and shifters. The shifters consume more logic resources and introduce significant delays [3].

There are some techniques in the literature to increase the speed of MD5 hash generation using an FPGA. Full-loop unrolled design of MD5 consumes more area but it is faster when compared to an iterative design [6]. Kimmo Jarvinen et al. [3] presented an architecture for the MD5 algorithm and observed that while pipelining the design increases throughput better than the parallelization, the latter is simple and easy to implement. Multi-parallel architecture for implementation of the MD5 algorithm is presented by Dongjing He and Zhi Xue [4]. 12 concurrent arithmetic MD5 modules are run in [4] and uClinux embedded system receives instructions from Host and sets four 32-bit PIO (programmed input/output). SRAM and internal memory are used by uClinux to communicate with MD5 modules. Multiple FPGAs are connected through Ethernet LAN (Local Area Network) to the host PC for increasing the throughput.

AL-Marakeby [5] has implemented two different architectures for MD5 hash on Altera DE2 FPGA. While the first architecture is a straightforward one with each MD5 module having its own tables, registers, etc., the second architecture shares tables, some counters, registers and control circuits with other MD5 modules. The second architecture requires lesser hardware, thus more number of MD5 modules can be fitted into a single FPGA yielding higher speeds. A cracking speed of 11.7M trials/second was achieved with the second architecture on Altera FPGA.

The cracking speed in case of software implementation is 57K trials/second using CPU, and 326K Trials/second using CPU and GPU together [7].

III. Parallelization of the brute-force attack on FPGA

A. *Top-level architecture*

The architecture for the brute-force attack on MD5 hash consists of 2 main components: one, the MD5 core which does the hashing of input messages, and two, the generator of the input messages (guess passwords in this case) to the MD5 core. Since the purpose of this work lies in the utilization of the hardware capability, it is planned to use FPGA to the maximum extent. Therefore, the generation of guess passwords (pw) is also implemented in the HDL.

Block level details of our pre-image brute-force attack is shown in Fig. 1. Multiple instances of guess password generator & MD5 hash generator (hereinafter referred to as PWMD5) pair will be there and the number of such pairs is dictated by the type of the architecture described in III-C. It is planned to use a processor just for providing the target hash (that is, the hash to be cracked) to the FPGA, then giving a start signal and keep on waiting for a flag from the FPGA to signal the success of the attack.

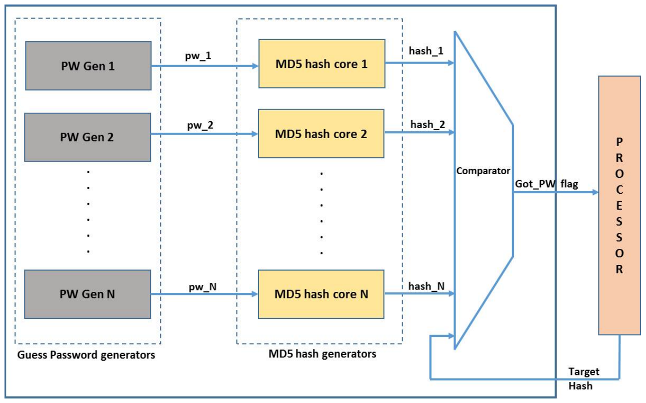

Fig. 1. Top-level architecture of our brute-force attack on MD5 Hash

Each MD5 core will have its own password generator for maximum speed. Processor comes as either hard or soft IP core on a Xilinx FPGA; rest of all the blocks in Fig. 1 are coded, synthesized and implemented on the FPGA.

B. *MD5 hash core*

MD5 hash core is implemented in VHDL based on (1) and (2). This involves 4 rounds of 16 steps each with a different auxiliary function given in (2) in each round. 64-stage pipelining is done in the MD5 hash core for achieving high frequency of operation on the FPGA. That is, each of the 64-steps output is registered before the next step operation. The implementation details for one MD5 step is shown in Fig. 2.

A full-loop unrolled architecture is implemented which require 64 such blocks connected in series to compute one MD5 hash. In Fig. 2, CV_i and CV_{i+1} are registers; remaining all are combinational elements. Constant vector matrix T and shift values S in each round in (1) are constants and are common to all 64 steps. Hence they are kept in a package which is shared among all 64 steps of the MD5 hash computation (shared among all MD5 core instances too). Functions F, G, H and I are computed based on (2). X is a message block (which is a password in this case with required padding), consisting of sixteen 32-bit words. Arithmetic and logical operations in one MD5 step depicted in Fig. 2 are ordered to get the minimum combinational delay.

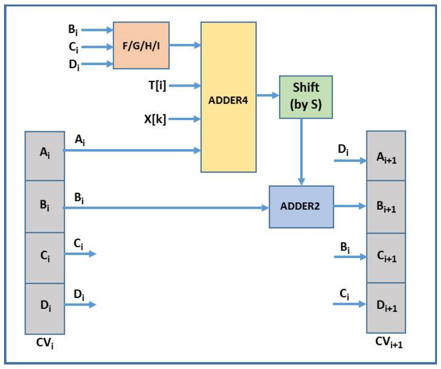

Fig. 2. Main process of the MD5 core implementation

C. Guess passwords

Generation of guess passwords requires multiple counters concatenated together. The start and stop values of counters depend on a full character set. If one looks at the ASCII table, the typeable characters are those from 20(Hex) to 7E(Hex), totaling 95 characters. This full character set includes capital & small alphabets, numbers and all special characters including space.

Guess passwords are generated as explained below (here, it is assumed that only one instance of PWMD5 is present): Let us assume $C_0C_1C_2....C_n$ is a guess password register (C_0 being the first character and C_n the last). Initially, the guess password will just be C_0 character which starts at 20(Hex) and increments by one for every clock. Once C_0 reaches 7E(Hex) then the guess password will have two characters concatenated as C_0C_1. C_0 is to be fixed at 20(Hex) and C_1 has to count from 20(Hex) to 7E(Hex). When C_1 reaches 7E(Hex), C_0 is to be incremented to 21(Hex) and C_1 has to go all over again. When both C_0 and C_1 reach 7E (this needs total 95^2 iterations), both of them have to be reset to 20(Hex). The guess password will now have three characters $C_0C_1C_2$, and so on.

Now, when multiple instances of MD5 hash cores are running in parallel, guess passwords also need to be generated in parallel. Three architectures are designed and tested for the generation of guess passwords. The difference amongst them is the character set, out of which passwords are generated, and the number of characters that each instance of the password generator should generate in the first position of the guess password. Every password generator is connected to exactly one hash generator in all the three cases. They are described in the following subsections:

1) ***Architecture-1 (Arch-1): Passwords with equal probability to all characters****:* In the first architecture, capital & small alphabets, numbers and special characters are all treated with the equal probability. So, there will be all of 95 characters in the character set in this case. There are total 32 instances of password generator and hence there will be one such instance for every 3 characters (with exception to the last instance, which has to generate only 2 characters) in the first position of the password. For example, instance-1 generates passwords that start with A, B or C. Instance-2 generates passwords starting with D, E or F and so on. It holds true for special characters and numbers too. Counters have to go through the full character set (that is, from ASCII value 20(Hex) to 7E(Hex)) in rest of all the positions in the password. This scheme is shown graphically in Fig. 3 where the characters are given as ASCII Hex numbers.

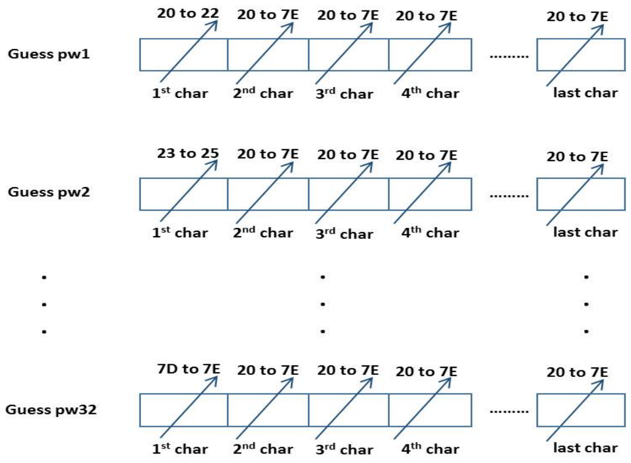

Fig. 3. Architecture-1 of guess password generation

2) ***Architecture-2 (Arch-2): Passwords with more importance to alphabets****:* This architecture is designed to take the advantage of the fact that a special character or number in the first position of the password is less probable and not allowed in some cases. Accordingly, this Arch-2 differs from Arch-1 in a way that less number of instances are allotted to such passwords which have a special character or number in the first position compared to passwords having first character as an alphabet. To be specific, there will be one instance for every two alphabets (capital or small letters) in the first position of the password, whereas there will be one instance for every 5 or 6 special characters or numbers in the first position of the password. As usual, counters have to go through all of 95 characters in rest of the positions in the password.

Total number of instances of password generator in this architecture is 34: 13 instances for capital letters, 13 instances for small letters and remaining 8 instances for numbers and special characters (again, this is with respect to the first position of the guess password). This scheme is shown graphically in Fig. 4.

3) ***Architecture-3 (Arch-3): Passwords with only alphabets****:* The third architecture generates passwords that con-

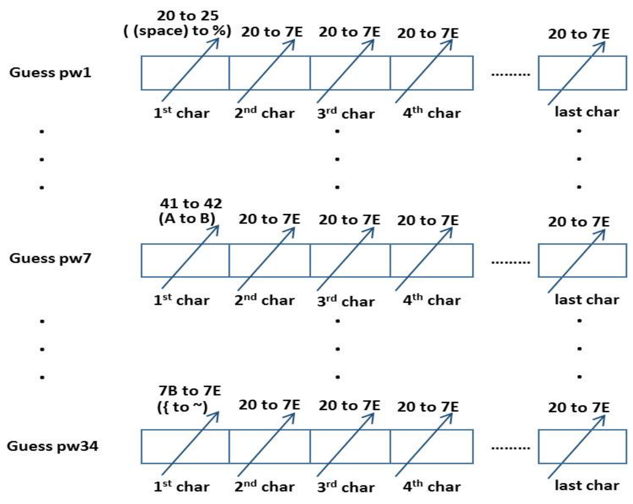

Fig. 4. Architecture-2 of guess password generation

tain only alphabets. Therefore, this architecture has smaller character set and hence is faster in cracking alphabets-only-passwords. So, the character set consists of 52 characters (small and capital letters). There will be total 26 instances and there will be one instance for every two alphabets in the first position of the password. For example, instance-1 generates passwords that start with A or B, instance-2 generates passwords that start with C or D and so on. Counters will go through the full character set of 52 letters in rest of the positions. This architecture is implemented only for comparing the cracking speed of passwords with only alphabets against that of passwords with all type of characters. This scheme is shown graphically in Fig. 5.

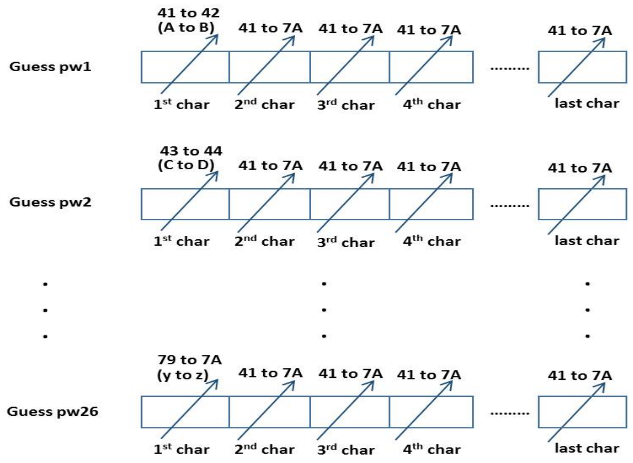

Fig. 5. Architecture-3 of guess password generation

D. *Optimization of performance of the brute-force attack*

Following factors or techniques are worked upon to maximize cracking speeds:

1) Number of parallel instances: In order to take full advantage of the parallelization, each MD5 hash core should have its own password generator to generate one guess password at every clock edge. The number of PWMD5 pairs that can be implemented on an FPGA depends on logic resources available in it. One such pair is taking up roughly 7500 LUTs (Look-up Tables) on a Xilinx FPGA. The FPGA hardware with the most number of logic resources that is available to us is Virtex-7 evaluation board VC707 from Xilinx. This board has Virtex-7 FPGA with full part number XC7VX485TFFG1761-2. This FPGA has got 485,760 logic cells containing 75,900 slices. Each slice of Xilinx 7 series FPGA contains 4 LUTs and 8 flip-flops. Since this work is about an algorithm implementation, LUTs will be used up mostly. 32 instances of PWMD5 (Arch-1) could be fitted into this Virtex-7 FPGA which resulted in about 81% utilization of LUTs. Further utilization resulted into either routing or timing failures.

2) Maximum frequency of operation and power dissipation: Another important consideration is the frequency of operation. As we place more and more instances, achieving higher frequency of operation becomes difficult. So, initially there was a trade-off between the number of PWMD5 instances versus frequency of operation of the logic. But later the power dissipation within the FPGA device has also become an issue once both the device utilization and frequency of the logic went up. For 32-instances parallelization, the frequency of operation of 100MHz could be achieved comfortably. But when the clock frequency is increased to about 150MHz, setup time margins have come down as well as the FPGA dynamic power is going up to 14W. It has then become a 3-way trade-off.

Following strategy is arrived at to optimize the speed of the brute-force attack in this scenario: first, fix the number of PWMD5 pairs to a decent high number based on the character set and device utilization. Then, keep on increasing the frequency of the implementation till the power dissipation becomes a bottleneck. The brute-force attack could be implemented with Arch-1 (which has 32 instances of PWMD5) for a maximum of 190MHz clock frequency with about 18.5W total power. Arch-2 (which has 34 instances) could be implemented at a maximum frequency of 178MHz with about 18.5W power. Arch-3 is implemented with 26 instances; hence its frequency of operation is higher. A frequency of 235MHz has been achieved with the third architecture with about 18.5W of power. All the power numbers mentioned in this paper are estimated figures and are taken from power analysis reports of Xilinx Vivado tool.

3) RTL design/coding techniques to improve the performance: The application demands not only functionality, but also high speed. Efficient design/RTL coding will help achieving this. For example, the following two cases (during the RTL development phase of this work) describe how enhancements to the design/VHDL code improved the speed of the design:

MD5 hash generation is implemented using 64-stage pipelining and the input message is required to be delayed for later stages down in the pipeline. Initially, the total delay logic (64 sets of flip-flops with one set for each pipeline stage) was present at the top-level of the MD5 core module. While this has met the functional requirement, the brute-force attack could be run for relatively higher frequencies when the individual delay logic is part of its corresponding pipeline stage of the MD5 hash generation due to reduced routing delays.

In another case, initially, a 'for loop' statement was used for computation of equations given in (1) within the main process of the MD5 core module. The process had 4 states, with each state computing 16-steps of the MD5 hash generation. There was one 'for loop' in each of the 4 states with loop count equal to 16. Although this gave a correct functional result, the frequency of operation of the logic was less. Because, whatever computation that is present inside the 'for loop' will be computed within one clock cycle. That means, the 16 steps will be completed in one clock cycle. This resulted in a huge combinational logic between the pipelined stages. The finer pipelining (16-stage pipelining in each of this four states) is done by removing the 'for loop' statement and by using 'if else' statements. In this way, each step (of the total 64 steps) of the MD5 hash generation is computed in one clock cycle. This partitioning of the combinational logic improved the speed of the design.

Attempts were made to pipeline each MD5 step shown in Fig. 2 further into 2 or 3 stages (keeping the number of PWMD5 instances constant as required by the guess password architectures) but they resulted in either device utilization or power requirement issues. This is due to the fact that the device utilization, power dissipation and frequency of operation are already high in the brute-force attack on Virtex-7 XC7VX485T FPGA using any of the three architectures of this work. The thermal dissipation issue due to high power dissipation is described in detail in IV-A2.

IV. Performance of the implemented brute-force attack

The total performance of architecture-1 and 2 reported in [5] is 527K trials/second and 11.7M trials/second, respectively. All the three architectures described in this work will run one trial per clock per one PWMD5 pair. Therefore, the performance of Arch-3, for example, is 235MHz X 26 instances. That is equal to 6,110M trials/second. The comparison of the performance of this work with that of AL-Marakaby [5] as well as software implementation is shown in table I. The cracking speed of this work is much higher than the software counterpart and about 522 times higher than that of [5].

TABLE I
Comparison of the performance of this work with software as well as other FPGA implementation

Parameter	In Software CPU+GPU [7]	Arch-2 of AL-Marakeby [5]	Arch-3 in this work
Performance (No. of Trials/sec)	326K	11.7M	6110M

Performance of all the three architectures presented in section III has been tabulated in table II. These are the actual numbers measured by running the brute-force attack on the FPGA hardware (VC707). A hardware timer (Xilinx AXI Timer v2.0 IP) is used to calculate the time taken for cracking the hash. This timer has got two 32-bit counters, but it is used in cascade mode to have a 64-bit counter for this work. Dividing the counter value with the frequency of operation gives out the cracking time in seconds.

TABLE II
Comparison of cracking time (practically tested on hardware) of all 3 architectures of this work

Password	MD5 hash cracking time in seconds		
	Arch-1	Arch-2	Arch-3
md5@∗	1.18	0.34	Not Applicable
sEwert	99	18	0.34
QwEsRTE	7,551	3,887	156

From table II it can be observed that, Arch-2 fares better when compared to Arch-1. This is expected, since Arch-2 runs more MD5 instances for alphabets when compared to Arch-1. Arch-3 is much faster since it looks for passwords that contain only alphabets. Arch-1 is a more general solution because it does not have any preferential treatment to a particular type of characters.

Table III shows the performance comparison of this work with AL-Marakeby's implementation [5] in terms of the cracking time for various length passwords. One important thing to note here is that all the values in the last column of this table are obtained by running the brute-force attack on VC707 board practically, whereas some of the numbers mentioned in column 2 are estimated (as mentioned in [5]).

TABLE III
Comparison of cracking times of this work with that of Al-marakeby [5]

Password	MD5 hash cracking time in seconds	
	Arch-2 in [5]	Arch-3 of this work
SFHA	0.5	2.85m
sEwert	1,041	0.336
QwEsRTE	8,012	156
DeSSaWeP	1,001,600	14,960

A. Challenges in parallelization of the brute-force attack

1) Requirement of more LUTs/logic cells on FPGA: Zedboard (an evaluation board of Xilinx Zynq FPGA) was used for this work initially. The FPGA on this board is XC7Z020-1CLG484C which has 85,000 logic cells with 53,200 LUTs. Since it could not accommodate more than 6 pairs of PWMD5, a custom board based on a bigger Zynq FPGA XC7Z100FFG900-2 was used later. This FPGA has got 444k logic cells with 277,400 LUTs. The brute-force attack could be parallelized up to 32 instances on this board. But this board was hanging up for more than 50MHz clock frequencies. An FPGA which can work at higher frequencies and has more LUTs was needed.

2) FPGA hang-up issue: Although synthesis and implementation was done at 100MHz successfully, Zynq XC7Z100 FPGA stopped responding when it was programmed with the bitstream. The evaluation board VC707, which is based on Virtex-7 XC7VX485T FPGA, was also hanging up for higher than 100MHz. After spending couple of days debugging the issue, it was found that there is a thermal dissipation problem on the board. When both the operating frequency and utilization of the FPGA are very high, it is drawing about 19A current (for 200MHz) on its core 1V supply from 1V regulator (U25 component on VC707 board). Though it is catered to supply up to 20A, the heat removal scheme from the regulator is not done properly. There is a temperature sensor inside the regulator which is detecting the high temperature due to high power dissipation and cutting-off the 1V supply within few minutes for 150MHz (14A current) frequency of operation, and after few seconds for 200MHz. Later, it was concluded that same was the issue with the Zynq custom board too. A portable fan (see Fig. 6) is setup which helps to remove the heat from the regulator of the VC707 board. With this setup, the brute-force attack could be run with Arch-1 at 190MHz continuously. The power dissipation at 190MHz is about 18.5W with about 81% utilization of LUTs.

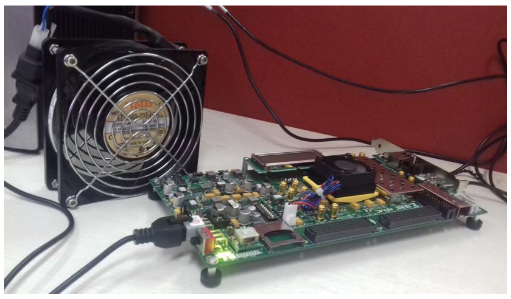

Fig. 6. VC707 Board with a portable fan for removing the heat from FPGA core voltage regulator

If this framework is scaled over multiple FPGAs, the number of trials/second in all the three architectures will also increase almost linearly. The number of PWMD5 instances per FPGA can be optimized in each of the three architectures based on the number of FPGAs that are available. The overall speed will be slightly less owing to the communication overheads among FPGAs.

V. Conclusion

The purpose of this work is to increase the speed of the brute-force attack on MD5 hash using an FPGA. Brute-force attack consists of two main components: guess password generation and hash generation. This work analyzed the ASCII character set and came up with 3 architectures of guess pw generation: Arch-1, Arch-2 and Arch-3. Arch-1 has equal number of instances of PWMD5 for passwords starting with alphabets, special characters or numbers. Arch-2 has one instance for every two alphabets in the first position of the password, whereas it has one instance for every five or six special characters or numbers in the first position of the password. This essentially means that Arch-2 has more computation power allotted to passwords that start with alphabets. Arch-3 targets passwords with only alphabets. Arch-1 is a more general and acceptable architecture in the sense that it does not impose any restriction on the type of characters that can be present in passwords and does not have any preferential treatment to a particular type of characters.

Cracking speed of about 6G trials/second has been achieved in the brute-force attack using all the three architectures in this work on Virtex-7 XC7VX485T FPGA. This performance is significantly higher (522X faster) than what is reported in the literature. A 7-digit password hashed using MD5 algorithm has been cracked on this Virtex-7 FPGA using Arch-3 in 156 seconds. This is made possible due to high levels of parallelism (26 instances in Arch-3), 64-stage pipelining within the MD5 hash core, better HDL coding techniques for high frequency implementation and superior performance of the latest FPGA device technology. These performance numbers are much higher than its software counterpart as well as other hardware implementation.

References

[1] rfc1810, "Report on MD5 Performance,"

[2] rfc1321, "The MD5 Message-Digest Algorithm,"

[3] K.Jarvinen et al., "Hardware Implementation Analysis of the MD5 Hash Algorithm," Proc 38th IEEE International Conference on System Sciences-2005.

[4] Dongjing He and Zhi Xue, "Multi-parallel Architecture for MD5 Implementations on FPGA with Gigabit-level Throughput," International Symposium on Intelligence Information Processing and Trusted Computing, 2010.

[5] AL-Marakeby, "Analysis of MD5 Algorithm Safety against Hardware Implementation of Brute Force Attack," International Journal of Advanced Research in Computer and Communication Engineering, vol. 2, issue 9, pp. 3332–3335, September 2013.

[6] J. Deepakumara et al., "FPGA Implementation of MD5 Hash Algorithm," Proc of the Canadian Conference on Electrical and Computer Engineering, CCECE 2001, Vol.2, pp.919-924, 2001.

[7] Feng Wang et al., "Constant Memory Optimizations in MD5 Crypt Cracking Algorithm on GPU-Accelerated Supercomputer Using CUDA," The 7th International Conference on Computer Science and Education, July 2012.