International Journal of Electronics and Communication Engineering (IJECE)
ISSN(P): 2278-9901; ISSN(E): 2278-991X
Vol. 9, Issue 5, Jul–Dec 2020; 25–32

International Academy of Science, Engineering and Technology
Connecting Researchers; Nurturing Innovations

ANALYSIS OF MD5 IMPLEMENTATION AND BRUTE FORCE ATTACK ON IT ON FPGA

Ambika. N

Research Scholar, Adhiyamaan College of Engineering (Autonomous), Hosur, India

ABSTRACT

The implementation of FPGA in MD5 algorithm is quicker when contrasted with programming execution, however, the brute force attack on MD5 actually needs 2-128 cycles hypothetically. This work will investigate the conceivable outcomes of improving the speed of brute force on FPGA of MD5 calculation. The proposed plan/technique in FPGA to parallelize the quest for a secret key that was hashed with the MD5 calculation. As a proof of idea the understudy will then, at that point exhibit the outcome for different secret key lengths appropriate to run inside sensible measure of time, and compare performance with the sequential implementation of brute-force attack in a Field Programmable gate array.

KEYWORDS: *FPGA, Brute Force Attack, MD5, Hash Function*

Article History

Received: *04 Aug 2020* | ***Revised:*** *10 Aug 2020* | ***Accepted:*** *14 Aug 2020*

INTRODUCTION

MD5 message digest algorithm takes a message of an arbitrary length and computes a fixed 128-bit hash output [3]. 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 algorithm on an FPGA.

Pipelining of the logic improves the frequency of operation of the logic and parallelization gives higher speed in terms of number of hash trials per second in case of the brute force attack on a hash algorithm. Both of these techniques and other design & Hardware Description Language (HDL) coding techniques need to be employed for improving the speed of the brute-force attack. In a pre-image type of brute-force attack on a hash algorithm, the hash is given and the corresponding input message is to be found. The method of generating guess input messages (hereinafter referred to as guess passwords or simply passwords) plays critical role in such a case. The emphasis will be on improving the cracking speed by efficient implementation of MD5 hash generation and guess passwords generation in an FPGA. The important factors to maximize the performance of brute-force attack are the number of parallel instances (of hash and password generators) that are running in a given FPGA, frequency of operation and power dissipation. An FPGA with high number of logic resources is required for this work. The brute-force attack is to be practically tested on an FPGA board and performance of the brute-force attack needs to be compared with the non-parallelized version as well as other works in the literature (both software and hardware implementations).

PROPOSED MD5 ALGORITHM:

The core of MD5 is a calculation which is utilized for the preparing of the message. The message M is isolated into 512-digit blocks which are handled independently we have several varieties of hash features that may be used to improve throughput. From the varieties of hash futures we utilize MD5 hash function. Pipelining is demonstrated a best answer for diminishing time in any VLSI Architecture. A Direct Advanced Synthesizer (DDS) have been proposed by [5] which utilizes half and half wave pipelining and Organize Revolution Computerized PC (CORDIC) calculation for programming characterized radios. The algorithm consists of four rounds, each of which comprises 16 steps.

The algorithm is performed as follows: first, values of A, B, C and D are stored into temporary variables. Then, every step operations are performed for i = 0 to 63 as follows:

$$B_{i+1} \leftarrow B_i + ((R(B_i, C_i, D_i) + A_i + T_{i+1} + M_j[i+1]) <<< S_{i+1})$$
$$A_{i+1} \leftarrow D_i, C_{i+1} \leftarrow B_i, D_{i+1} \leftarrow C_i$$

Finally, the values of the temporary variables are added to the values obtained from the algorithm, and the results are stored in the registers A, B, C and D.

When all the message blocks have been processed, the message digest of M is in A, B, C, and D. Analysis and Optimization for MD5.

Analysis and Optimization for MD5

The heart of MD5 is the algorithm which is used for calculating the values of A, B, C, and D in every step operations. From Eq.1, we can see that the values of A, C, D can be got directly, while the calculation of B is quite complicated, which consists of four mod 232 additions, a logical function and a circular shift left operation, forming the critical path of the MD5 algorithm[7].

And the delay of the critical path is: T=4×Delay(+) +Delay(R)+Delay(<<S)which is much larger than the iteration bound T∞=2×Delay (+)+Delay(R)+Delay(<<S) proposed it. Therefore, to achieve a throughput optimal design, we must optimize the iterative architecture to shorten the critical path.

The Pipelined Architecture Based on the Loop Unrolling Technique:

The Pipelined Design Dependent on the Circle Unrolling Strategy: The 4-stage pipelined engineering is introduced in Fig, which presents the unrolling method dependent on MD5. It unrolls every one of the 16 stages in each round, and acted in alone, so in this engineering, each round computation will be prepared in one cycle in any event. In the 4-stage pipe-lining, barring the initial 512-bit of message square will be acted in 4 clock cycles, the later ones will be acted in just a one clock cycle.

As can be appeared in equationBi+1=Bi+((R(Bi,Ci,Di)+Ai+Ti+1+Mj[i+1]))<< Si+1, Ti+1 and Si+1 are constants in each clock cycle, Mj[i+1] is a 32-digit message in a 512-bit of square, which likewise exhibited as a fixed steady after the 512-bit is inputted, and the helpfulness of artificial intelligence equivalents to Di-1, so on the off chance that we present an impermanent variable Tempi in the ith clock cycle, and let Tempi be Di-1+Mj[i+1]+Ti+1, at that point in the i+1th clock cycle, Bi+1 can be improved on as:Bi+1=Bi+((R(Bi,Ci,Di)+Tempi))< Si+1 some pre-calculation is performed.

After the streamlining, each progression activities are:/Since register An isn't convenience in the accompanying computations, the value of Tempi can be put away in register simulated intelligence, for example Ai+1=Di+Ti+2+ Mj[i+2]. Besides, a Convey Save Snake can likewise be applied in Di+Ti+2+Mj[i+2], which can save some region and decrease some deferral. After the streamlining, the calculation is abbreviated, there are just two increases rather than four, which improves the speed altogether.

$$B_{i+1} \leftarrow B_i + ((R(B_i, C_i, D_i) + Temp_i) <<< S_{i+1})$$
$$A_{i+1} \leftarrow D_i, C_{i+1} \leftarrow B_i, D_{i+1} \leftarrow C_i$$
$$Temp_{i+1} \leftarrow D_i + T_{i+2} + M_j[i+2]$$

And the delay of the path is T=2×Delay (+)+Delay(R)+Delay(<<S) which is equal to the iteration round of MD5 proposed in [4], according to [4], the optimized architecture achieves the maximum outcome of throughput by iterative architecture. Moreover, the logic area of the considered structure is not increased almost distinguish to the earliest structure.

The Pipelined Architecture Based on Iterative Technique

This construction is found on 4-stage pipelining, it can ascertain four distinctive 512-digit of message hinders at the same time, prohibit the initial 512-pieces of message squares will be acted in 64 clock cycles, the later ones will be acted in just 16 clock cycles. Start signal is used when a determination of another message digest 5 is begun, for instance, when M0 is prepared, and Continue signal is almost new for the later ones, like Mj, where j=1. The engineering checks from 0-64 and it is restart to zero when Start or Continue signal is high.. Other blocks, comparatively the input block, and memory block and the encrypting block are all controlled by the counter.

Then the 4 BUF blocks are used to keep four different groups of 512-bit of message blocks, which are make use of 4 encrypting blocks respectively. The ABCD_REG blocks are used to keep four different groups of middle outcomes of the A, B, C, and D. T_REG is recycled to keep the constant T. The four encrypting blocks are the main functional units of the algorithm, which are utilized for the four rounds of calculations respectively. Each encrypting block is fetched to the equivalent BUF block, and which can just only process the given statistics in the connected BUF block.

After processing the corresponding data, the result will be transferred to another encrypting block, then the features of the connected BUF block are also transferred to the upcoming BUF block. This pattern has the advantages as follows: first, it avoids the bus competition, and each encrypting block can own the whole bandwidth of the connected BUF block, which reduce the delay of the design. Second, this pattern simplifies the logic control, and it also reduces the logic requirements demanding.

In addition, this structure has well expansibility, when it needs to increase or decrease the pipelined stages, we can add or delete the corresponding encrypting structure and the connected BUF blocks simply. Take the one stage pipelining in this proposed work for example, which is only needed to delete three encrypting development and the three BUF blocks connected from the constructed Fig. And for the 32-stage pipelining, add encrypting blocks and the corresponding BUF blocks to 32[8].

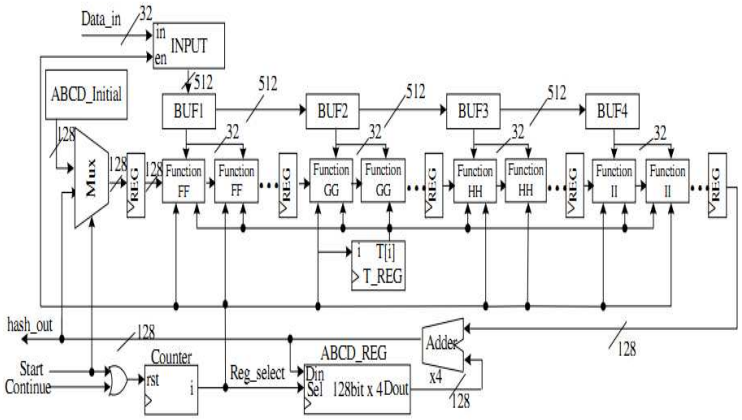

Figure 1: Pipelined Architecture of MD5 Iterations.

ANALYSIS OF MD5 IMPLEMENTATION AND BRUTEFORCE ATTACK ON IT ON FPGA

MD5 hash generation has a block size of 512-bits. Therefore, the 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. Minimum and maximum number of bits to be appended are 1 and 512 respectively [1]. 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 [3]. The MD5 algorithm has 4 rounds, with each round consists of 16 steps [2]. The computation in each round is given by [1]:

$$A = B + \left((A + \text{AuxFn}(B, C, D) + Xj\,[k] + T[i]) \ll S \right)$$

$$A \leftarrow D, B \leftarrow A, C \leftarrow B, D \leftarrow C \qquad (1)$$

Whereas the auxiliary function (denoted as AuxFn in equation (1.1)), one for each round, is defined as follows:

$$F(X, Y, Z) = (X \wedge Y) \vee (\neg X \wedge Z)$$

$$G(X, Y, Z) = (X \wedge Z) \vee (Y \wedge \neg Z)$$

$$H(X, Y, Z) = X \oplus Y \oplus Z$$

$$I(X, Y, Z) = Y \oplus (X \vee \neg Z) \qquad (2)$$

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). Xj [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 the process of MD5 hash generation four times. Each of 16 words of X enters each round as per the following equations (except in the first round where they enter in their original order).

$$\rho 2(i) = (1 + 5i) \bmod 16$$

$$\rho 3(i) = (5 + 3i) \bmod 16$$

$$\rho 4(i) = 7i \bmod 1 \qquad (3)$$

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 bit-wise logical operators on input data. The implementation also requires 32-bit adders and shifters. The shifters consume more logic resources and introduce significant delays in the implementation of MD5 hash algorithm on FPGA.

PARALLELIZATION OF BRUTE-FORCE ATTACK ON MD5 ALGORITHM ON FPGA

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 guess passwords to the MD5 core. Since the purpose of this project lies in the utilization of the hardware capability for cracking the MD5 hash, it is planned to use FPGA to the maximum extent. Therefore, guess password (pw) generation is also implemented in verilog HDL.

Block level details of my brute-force attack are shown in Figure 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 3.3. 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 FPGA to signal the success of the attack. Based on this, both MD5 hash generation as well as guess passwords generation are planned to be implemented in the hardware. Each MD5 core will have its own password generator for maximum performance. Processor comes as either hard or soft IP core on FPGA; rest of all blocks in Figure 2 are coded, synthesized and implemented on the FPGA.

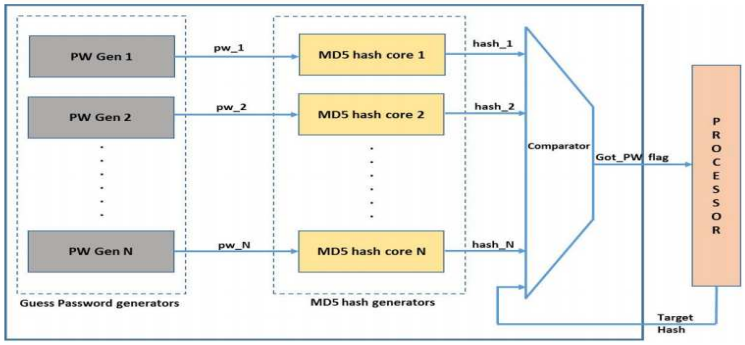

Figure 2: Top-Level Details of Brute-Force Attack on MD5 Hash.

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 typical 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; that is, there is no parallelization): Let us assume C0C1C2....Cn is a guess password register (C0 being the first character and Cn the last). Initially, the guess password will just be C0 character which starts at 20(Hex) and increments by one for every clock.

Once C0 reaches 7E(Hex) then the guess password will have two characters concatenated as C0C1. C0 is to be fixed at 20(Hex) and C1 has to count from 20(Hex) to 7E(Hex). When C1 reaches 7E(Hex), C0 is to be incremented to 21(Hex) and C1 has to go all over again. When both C0 and C1 reaches 7E (this needs total 952 iterations), both of them have to be reset to 20(Hex). The guess password will now have three characters C0C1C2 and C2 has to count from

20(Hex) to 7E(Hex). Once C2 reaches 7E, then C1 is to be incremented to 21(Hex) and C2 has to go all over again. When both C1 and C2 reaches 7E, then C0 is to be incremented to 21(Hex) and C1 and C2 have to go all over again. Once C0, C1 and C2 all reach 7E (this needs 953 iterations) then C3 will come into picture and guess password will now be C0C1C2C3, and so on.

Now, when multiple instances of MD5 hash cores are running in parallel, guess passwords also need to be generated in parallel. This architecture was designed and tested for the generation of guess passwords for the brute-force attack on MD5 hash. The difference amongst them is the character set, out of which passwords are generated, and the number of characters that each instance of password generator should generate in the first position of the guess password.

Architecture: Passwords with equal Probability to All the 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, the 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 figure 2 where the characters are given as ASCII Hex numbers.

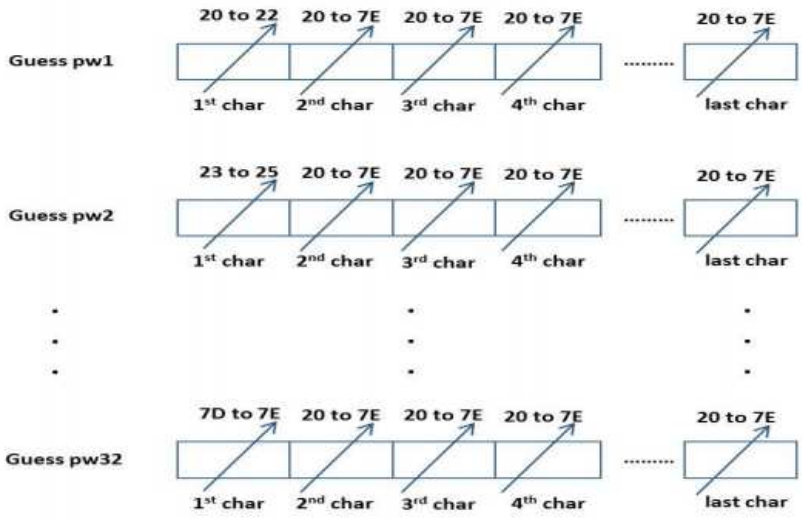

Figure 3: Architecture of Guess Password Generation.

RESULTS AND DISCUSSIONS

In the brute force attack of Guess Password generation the propose work is analyzed by generation 4 PW fig 4. For this four PW generation outcomes the find result is fetched to the improved design of 16 pipeline stages MD5 core. Output shown in Figure 5.

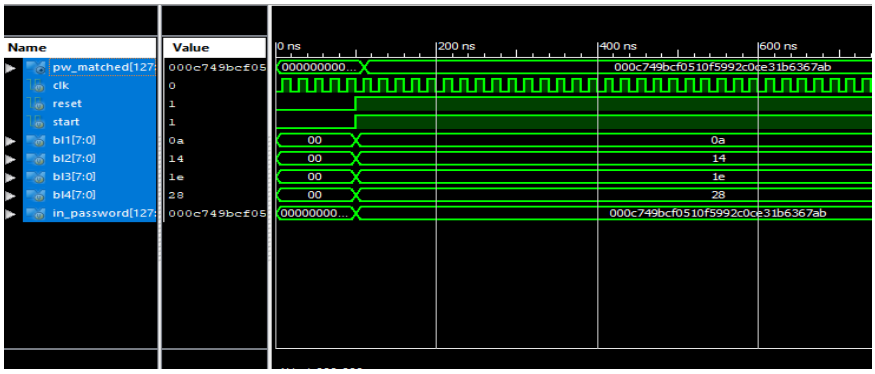

Figure 4: Output of 4 Guess Password.

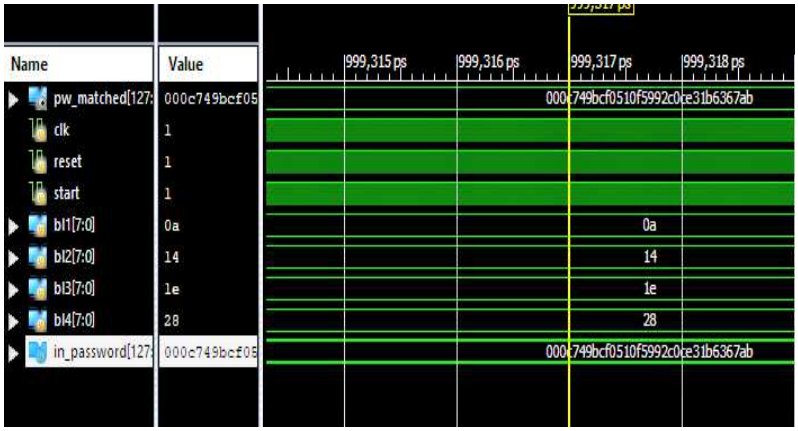

Figure 5: Output of Pipeline Stages of Brute Force Attack.

FUTURE SCOPE

Although, SHA is safer than MD5 because of an assortment of reasons. To begin with, it delivers a bigger review, 160-piece contrasted with 128-bit, so a savage power assault would be significantly more hard to do. Likewise, no known impacts have been found for SHA. SHA-1 is quickest hashing capacity with ~587.9 mms per 1M tasks for short strings and 881.7 ms per 1M for longer strings. MD5 is 7.6 % more slow than SHA-1 for short strings and 1.3 % for longer strings. SHA-256 is 15.5 % more slow than SHA-1 for short strings and 23.4 % for longer strings. A portion of the contrasts among MD5 and SHA are depicted in underneath Table.

Table 1: Differences between MD5 and SHA

S. No	MD5	SHA
1	MD5 can have 128 bits length of message digest.	Whereas SHA1 can have 160 bits length of message digests.
2	The speed of MD5 is quick in examination of SHA1's speed.	While the speed of SHA1 is delayed in examination of MD5's speed.
3	To make out the underlying message the attacker would want2^128operations while misuse the MD5 algorithmic program.	On the contrary hand, in SHA1 it'll be 2^160 that makes it very problematic to search out.
4	MD5 is less complex than SHA While SHA1 is more perplexing than MD5.	While SHA1 is more complex than MD5.

CONCLUSIONS

Four sorts of MD5 configuration dependent on Xilinx 14.2 and higher, Hardware Kit: Xilinx Spartan 6 were effectively incorporated and actualized. The outcomes are shown that the modified MD5 unfolding with 16 phases of pipelining gives a superior MD5 plan as greatly as speed, area and throughput. The gain of unfolding change a span of throughput of MD5 to diminishing its latency. Additionally, the pipelined MD5 configuration has rapid in the increasing the frequency of pipeline. Subsequently, a MD5 configuration consolidating the unfolding change and pipelining can improve the output fundamentally.

REFERENCES

1. *Diez, J.M., Bojanić, S., Stanimirovicć, L., Carreras, C., Nieto-Taladriz, O.: Hash Algorithms for Cryptographic Protocols: FPGA Implementations. In: Proceedings of the 10th Telecommunications Forum, TELFOR 2002, Belgrade, Yugoslavia, November 26-28 (2002)*
2. *Yiakoumis, I., Papadonikolakis, M., Michail, H.: Efficient Small-Sized Implementation of the Keyed-Hash Message Authentication Code. In: EUROCON 2005, Serbia & Montenegro, Belgrade, November 22-24 (2005)*
3. *T. Madheswaran, M., Menakadevi, T. An Improved Direct Digital Synthesizer Using Hybrid Wave Pipelining and CORDIC algorithm for Software Defined Radio. Circuits Syst Signal Process32, 1219–1238 (2013)*
4. *Rivest, R.L.: The MD5 Message-Digest Algorithm. RFC 1321, MIT Laboratory for Computer Science and RSA Data Security, Inc. (April 1992)*
5. *Amphion. CS5315, High Performance Message Digest 5 Algorithm (MD5) Core. Datasheet(2004), http://www.amphion.com/acrobat/DS5315.pdf*
6. *K.Jarvinen et al., "Hardware Implementation Analysis of the MD5 Hash Algorithm," Proc 38th IEEE International Conference on System Sciences-2005.*
7. *AL-Marakeby, "Analysis of MD5 Algorithm Safety against Hardware Implementation of Brute Force Attack," International Journal of Ad-vanced Research in Computer and Communication Engineering, vol. 2, issue 9, pp. 3332–3335, September 2013.*
8. *T Menakadevi, M Madheswaran, "FPGA implementation of direct digital synthesizer using pipelined cordic algorithm", European Journal of Scientific Research, Vol. 79, Issue. 2, pp. 269-278.*
9. *N. Ambika, Dr. Menaka Devi. T. "Power estimation and High throughput MD5 design by Unfolding transformation" International journal of interdisciplinary innovative research and development (IJIIRD), having an impact factor 5.222 & ISSN: 2456-236x in 05 Volume, Issue 01.*