Pecking Order Hash – A New Hashing Algorithm

Siva Srinivasa Rao Mothukuri
Department of Information Technology
RVR & JC, Acharya Nagarjuna University Guntur, Andhra Pradesh, India

Abstract

Since the data is present in digital form, security is a major assert and aspect that need to be considered in present activities. However, several incidents show us how security has compromised leading to theft/loss of data like credit card numbers and passwords. Having profound knowledge about passwords alone doesn't make strong security. Saving the pass-phrase on the server matters a lot. If this fails, leads to failure of entire system. This paper introduces a new hashing algorithm which uses levels of hierarchy and turns strong when it is used along with other hashing algorithms.

Keywords: Cryptography, MD5, SHA Family

I. INTRODUCTION

When "Security (Digital)" term strikes our mind, immediately passwords come into picture. As every privacy data is only accessed with help of one token, we can't compromise on it. However, if this token is stored as plain text on the server, then there is no use of having a unique private token for every individual. A working employee can know all the user passwords. If a simple login password is known to others, then it might not be a big problem but the same cannot be acceptable in the case of bank/credit card passwords. Now, there comes the challenge where a token is converted to a unique string and can't be reverted back. This is simply coined as "Hash" and the technique is "Hashing".

Security is a huge base which contains other elements along with cryptography like encoding, encryption and so on. Under cryptography, hashing is major area where a message can't be reverted back to original. There are different hashing techniques followed like MD5, SHA. In this paper a new hashing technique is introduced.

II. PECKING ORDER HASH

Pecking Order Hash is a new hashing technique that we are going to deal with in this paper.

It has series of phases while generating the hash. As there are level of hierarchy, the algorithm is named as pecking order hash. The technique is based on the words in the message along with some of initial variables.

A. Initial Stage:

1) *Initial Variables:*

To start off, this algorithm uses a series of initial inputs for hashing. Processing of message is done along with these initial variables. The reason to use these variables is to avoid collision when the input contains only 0 or empty string. Initial variables are 16 in count with 32bit length each. Below is the list of Initial Variables in Hex String Format.

I_0 = 01234567
I_1 = 89ABCDEF
I_2 = 10325476
I_3 = 98BADCFE
I_4 = 0A1B2C3D
I_5 = 4E5F6978
I_6 = 02467531
I_7 = 8ACEFDB9
I_8 = 0123ABCD
I_9 = 4567EF89
I_10 = ACE02468
I_11 = BDF13579
I_12 = FEBA7632
I_13 = DC985410
I_14 = E1D2C3B2
I_15 = C9B8A776

All rights reserved by *www.ijirst.org*

2) *Padding:*

In this stage, every input message is considered as bits of 1024 which in turn converted to 512 bits of output. If a message is less than 1024 bits then bit stuffing is done with respective length. In this process, we will have different bit stuffing since the length of the message varies every time. So, at the time of padding the bits will change unless they have same length.

If a message length is greater than 1024 bits, then bit stuffing is done until we get a message of length which is a multiple of 1024 bits.

3) *Chunking:*

Each message is split into chunks of 1024 bits. This will be easy for every operation when we process the message. Furthermore, hashing technique is performed on one chunk at a time, by dividing it into words of 32 bits.

B. Processing Stage:

1) *Pre-Phasing:*

The pre-phasing stage checks for number of chunks. If there is only one chunk, then it is processed through all the stages as it is. The same processing is not followed when more number of chunks are encountered. After completion processing of each chunk, the output of it is taken as initial variable and further processing is performed on next chunk.

2) *Phase I:*

In this phase, the 32 words are processed with a series of iterations (I) of 512. In every iteration, 1^{st} block of input word is assigned to 2^{nd}, 2^{nd} block input to 3^{rd} and so on except for following.

- 4^{th} block of output is XOR of Mod[Ith,16] Initial Variable and Mod[Ith,32] word.
- 5^{th} block of output is Shift Right 12 for 4^{th} word.
- 7^{th} block of output is ANDNOT of Mod[Ith,6] Initial Variable and 6^{th} word.
- 10^{th} block of output is AND of Mod[Ith, 13] Initial Variable and 19^{th} word.
- 11^{th} block of output is AND of 6^{th} word and 11^{th} word.
- 12^{th} block of output is XOR of Mod[Ith,5] Initial Variable and 9^{th} word.
- 13^{th} block of output is ANDNOT of 0^{th} word and 27^{th} word.
- 16^{th} block of output is AND of Mod[Ith,2] Initial Variable and 14^{th} word.
- 21^{th} block of output is Shift Right 6 of 16^{th} word.
- 23^{th} block of output is XOR of Mod[Ith,4] Initial Variable and 19^{th} word.
- 27^{th} block of output is Shift Right 12 of Mod[Ith,9] Initial Variable and 30^{th} word.

Mod is function that leaves remainder for input divided with other. The output is passed to phase 2 for further processing.

3) *Phase II:*

In phase II, 32 words are converted to 16 words by performing bitwise XOR operation between 1^{st} word and 16^{th} word, 2^{nd} word and 15^{th} word and so on, in chronological order. The output is passed to next phase.

4) *Phase III:*

Now, 16 words are processed with a series of iterations (I) of 512. Similar iterations are followed in this phase. In every iteration, 1^{st} block of input word is assigned to 2^{nd}, 2^{nd} block input to 3^{rd} and so on except for 4^{th} block.

- 4^{th} block of output is Function output of Mod[Ith,16] Initial variable with Mod[Ith, 16] word, Mod[Ith,3].

The Function output is a processing method that contains following.

case 0
result = AND Operation(word1, word2)
result = Shift Right 12 of (result, word2)
result = ANDNOT(word1, result)
case 1
result = Shift Right 12 of (word1, word2)
result = ANDNOT(word1, result)
case 2
result = ANDNOT(word1, word2)
result = Shift Right 12 of (result, word2)

The further output is passed to final stage.

All rights reserved by *www.ijirst.org*

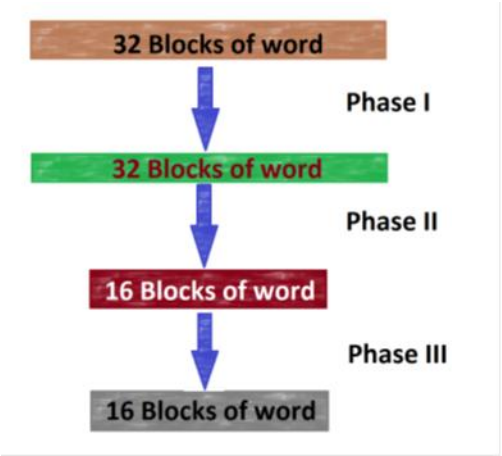

Fig. 1: shows phases hierarchy

C. Final Stage:

At this stage, XOR operation is performed for initial variables and words. If, there is more than one chunk then output is taken as initial variables for next iteration.

III. IMPLEMENTATION

A. Algorithm:

Inputs:

- 16 Variables – of Initial inputs
- Message that need to be hashed

Output:

- Hash for input message

Processing:

1) *Padding:*

Convert Input String to Hex
For Hex > 255
Make input to pad with the length in hex format with nearest possible multiple of 255
Else
Pad with the length in hex format

2) *Chunking:*

For Padded input > 256
Split into multiples of 256
For Chunks
Split into 32 words of length 32 bit

3) *Phase I:*

For u in 0 to 512
For i in 0 to 32
outputword[i]=inputword[(i+1)%32]
Except for following

- 4th block of output is XOR of Mod[Ith,16] Initial Variable and Mod[Ith,32] word.
- 5th block of output is Shift Right 12 for 4th word.
- 7th block of output is ANDNOT of Mod[Ith,6] Initial Variable and 6th word.
- 10th block of output is AND of Mod[Ith, 13] Initial Variable and 19th word.
- 11th block of output is AND of 6th word and 11th word.
- 12th block of output is XOR of Mod[Ith,5] Initial Variable and 9th word.
- 13th block of output is ANDNOT of 0th word and 27th word.
- 16th block of output is AND of Mod[Ith,2] Initial Variable and 14th word.
- 21th block of output is Shift Right 6 of 16th word.
- 23th block of output is XOR of Mod[Ith,4] Initial Variable and 19th word.
- 27th block of output is Shift Right 12 of Mod[Ith,9] Initial Variable and 30th word.

END

All rights reserved by *www.ijirst.org*

inputword=outputword
END

4) *Phase II:*

For i in 0 to 16
outputword[i]=XOR of inputword[i] and inputword[31-i];
END

5) *Phase III:*

For u in 0 to 512
For i in 0 to 32
outputword[i]=inputword[(i+1)%32]
Except for following
4th block of output is Function (initial[i % 16], inputwords[i % 16],i % 3);
END
Inputword = outputword
END

6) *Final Stage:*

For i in 0 to 16
outputword[i] = XOR of inputword[i], initial[i];
END
END

B. Sample Snapshots:

Following are the sample snapshots for the implementation of the algorithm.

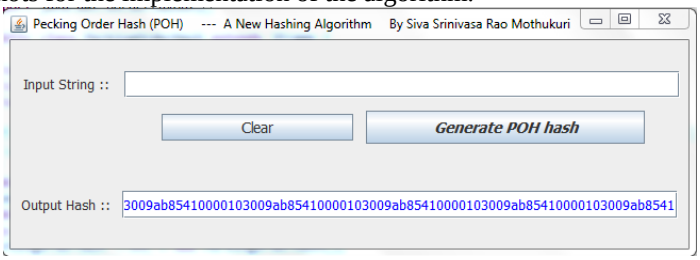

Fig. 2: (a) sample snapshots

INPUT: POH("");
OUTPUT:
123012389a24025123012309ab85410000103009ab85410000103009ab85410000103009ab85410000103009ab85410000103009ab85410000103009ab8541
Above snapshots, shows the hash for an empty string.

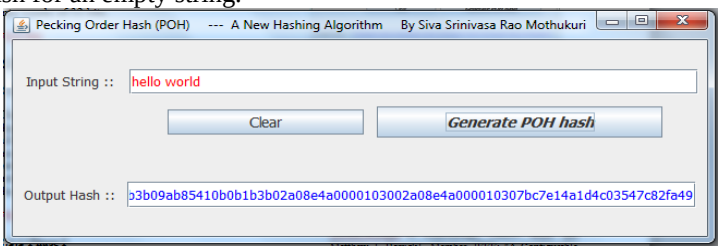

Fig. 2: (b) sample snapshots

INPUT: POH("hello world")
OUTPUT:
68642828e6a97b2e636f23237789f1421647085f70ccea410b0b1b3b09ab85410b0b1b3b02a08e4a0000103002a08e4a000010307bc7e14a1d4c03547c82fa49

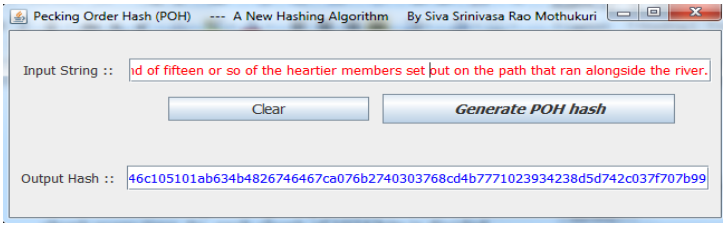

Fig. 2: (c) sample snapshots

All rights reserved by *www.ijirst.org*

Input: POH("
As they rounded a bend in the path that ran beside the river, Lara recognized the silhouette of a fig tree atop a nearby hill. The weather was hot and the days were long. The fig tree was in full leaf, but not yet bearing fruit. Soon Lara spotted other landmarks—an outcropping of limestone beside the path that had a silhouette like a man's face, a marshy spot beside the river where the waterfowl were easily startled, a tall tree that looked like a man with his arms upraised. They were drawing near to the place where there was an island in the river. The island was a good spot to make camp. They would sleep on the island tonight.

Lara had been back and forth along the river path many times in her short life. Her people had not created the path—it had always been there, like the river—but their deerskin-shod feet and the wooden wheels of their handcarts kept the path well worn. Lara's people were salt traders, and their livelihood took them on a continual journey. At the mouth of the river, the little group of half a dozen intermingled families gathered salt from the great salt beds beside the sea. They groomed and sifted the salt and loaded it into handcarts. When the carts were full, most of the group would stay behind, taking shelter amid rocks and simple lean-tos, while a band of fifteen or so of the heartier members set out on the path that ran alongside the river.
")
OUTPUT:
2b6360ec85626c6559430d19500ff12e1b0c6c195b51ad3f12146c105101ab634b4826746467ca076b2740303768cd4b7771023934238d5d742c037f707b99
Above snapshot shows hash for a lengthy text.

IV. FUTURE WORK

The POH can't be strong all alone; this can be added up with private salt which gives more security. Furthermore, one can have multiple hashing with multiple salts at every level of hashing, results in high level of security. Moreover, security can be increased by having multiple levels of hashing techniques like SHA-1, POH, MD5. These results in very apex security, even the output can withstand rainbow table attacks up-to some extent.

V. CONCLUSION

Security is most important aspect in today's environment. Every essential work is controlled through a single pass-phrase, which cannot be compromised. POH provides a new way of hash along with other hashing techniques.

REFERENCES

[1] Chih-Chung Lu ,Shau-Yin Tseng "Integrated design of AES (Advanced Encryption Standard) encrypter and decrypter ","Application-Specific Systems, Architectures and Processors, 2002. Proceedings. The IEEE International Conference"
[2] A flexible hardware implementation of SHA-1 and SHA-2 Hash Functions Docherty, J. ; Koelmans, A. Circuits and Systems (ISCAS), 2011 IEEE International Symposium on DOI: 10.1109/ISCAS.2011.5937967 Publication Year: 2011 ,IEEE CONFERENCE PUBLICATIONS
[3] Novel high throughput implementation of SHA-256 hash function through pre-computation techniqueMichail, H. Milidonis, A. Kakarountas, A. Goutis, C. Electronics, Circuits and Systems, 2005. ICECS 2005. 12th IEEE International Conference on DOI: 10.1109/ICECS.2005.4633433 Publication Year: 2005 IEEE CONFERENCE PUBLICATIONS
[4] Design and Performance Analysis of a Unified, Reconfigurable HMAC-Hash Unit Khan, E. ; El-Kharashi, M.W. ; Gebali, F. ; Abd-El-Barr, M. Circuits and Systems I: Regular Papers, IEEE Transactions on Volume:54, Issue: 12 DOI: TCSI.2007.910539 Publication Year: 2007
[5] Security Analysis of salt||password Hashes Gauravaram, P. Advanced Computer Science Applications and Technologies (ACSAT), 2012 International Conference on DOI: 10.1109/ACSAT.2012.49 Publication Year: 2012 ,IEEE CONFERENCE PUBLICATIONS
[6] Implementation and Comparison of Two Hash Algorithms Zhenqi Wang ; Lisha Cao Computational and Information Sciences (ICCIS), 2013 Fifth International Conference on DOI: 10.1109/ICCIS.2013.195 Publication Year: 2013 ,IEEE CONFERENCE PUBLICATIONS

All rights reserved by *www.ijirst.org*