Implementing Rainbow Tables in High-end FPGAs for Super-fast Password Cracking

Kostas Theoharoulis, Ioannis Papaefstathiou
ECE Department,
Technical University of Crete
Chania, Crete, Greece
ygp@mhl.tuc.gr

Charalampos Manifavas
Applied Informatics & Multimedia Dept.
Technological Educational Institute of Crete
Heraklion, Crete, Greece
harryman@epp.teicrete.gr

***Abstract*— One of the most efficient methods for cracking passwords, which are hashed based on different cryptographic algorithms, is the one based on "Rainbow Tables". Those lookup tables offer an almost optimal time-memory tradeoff in the process of recovering the plaintext password from a password hash, generated by a cryptographic hash function. In this paper, the first known such generic system is demonstrated. It is implemented in a state-of-the-art reconfigurable device that cracks passwords, which are encrypted with a number of different cryptographic algorithms. The proposed FPGA-based system is up to 1000 times faster than the corresponding software approach. This is achieved by using a highly parallel architecture employing a fine-grained pipeline.**

I. INTRODUCTION

A basic problem in cryptology is the computation of the inversion of the one-way functions. For example, an attack on a block cipher analyzes the mapping of the key to the cipher text, which should be an one-way function. If such a cryptographic function leads to an "n-bit" result, there are two straightforward methods: a) an exhaustive search can be performed over an average of 2^{n-1} values until the target is reached, b) 2^n input and output pairs can be pre-computed and stored in a table. In order to invert a particular value, we just look up the pre-image in the table; in this case the inverting function requires only a single lookup.

The Hellman algorithm [5] lies in-between those two solutions. The pre-computation time of this approach is still on the order of 2^n, but the memory complexity is $2^{2n/3}$ and the inversion of a single value requires only $2^{2n/3}$ function evaluations. The initial algorithm is further generalized by Fiat and Naor [2], who propose a more demanding variant at the cost of extra processing power and/or memory. Moreover, Kusuda and Matsumoto [7] further analyzed the Hellman algorithm, while they demonstrated the relationships between the memory complexity, the processing complexity and the success probability. The memory accesses of the initial Hellman's algorithm were further reduced by employing distinguished points instead of continuous ones; this approach was analyzed by Borst [3].

In 2003 and more recently in 2008, Oechslin [1], [6] further expanded Hellman's approach and suggested the "Rainbow Tables", which are utilized in the pre-computation task of the algorithm. As mentioned in Oechslin's papers "this method combines the advantage of the distinguished point approach (reduced number of memory accesses), with the higher success probability and easier analysis of Hellman's original method".

Moving to a different area, a Field Programmable Gate Array (FPGA) is a semiconductor device containing programmable logic components and programmable interconnects. The programmable logic components can be programmed to duplicate the functionality of basic logic gates such as AND, OR, XOR, NOT, or even more complex combinational functions, such as decoders, or math functions. These programmable logic components also include memory elements, which may be simple flip-flops or complete blocks of highly-dense memories. A hierarchy of programmable interconnects allows the logic blocks of an FPGA to be interconnected according to the system designer's needs. These logic blocks and interconnects can be programmed by the customer/designer after the manufacturing process (hence the term "field programmable", i.e. programmable in the field), so that the FPGA can each time perform the necessary logical function.

State of the art FPGAs include thousands of programmable logic blocks and provide embedded processors, memories, transceivers, multipliers and adders interconnected in a modular way. In FPGAs, the growth of transistors' density is faster than the one in general processors. Nowadays, the largest FPGAs in the market belong to the Xilinx Virtex4 and Virtex5 family of devices and provide more than eight million "equivalent gates" (the relative density of logic). These advanced devices also offer features, such as built-in hardwired processors (e.g. the IBM PowerPC), substantial amounts of memory in the range of 10Mbits, clock management systems. Additionally, they support very fast device-to-device signaling technologies. Furthermore, they feature dedicated application-specific hardware blocks, related to Digital Signal and Math Processing (DSP) and specialised Input/Output ports for interfacing the FPGA with high-speed off-chip links, giving bandwidth in the range of 40 Gbps. These benefits indicate FPGAs as desirable platforms for encryption/decryption applications, such as the ones addressed by our system. Additionally, FPGAs are generally less expensive, for a given performance, than their processor counterparts; e.g. a high-end FPGA that offers the equivalent processing power of more than 35 Intel High-end CPUs, in certain tasks, is less than 5 times more expensive than a single such CPU.

This paper, proposes a system, which is implemented in a high-end reconfigurable device that can create Rainbow Tables very efficiently. The actual creation process of the

1946-1488/10 $26.00 © 2010 IEEE
DOI 10.1109/FPL.2010.120

Rainbow Tables depends on the cryptographic algorithm attacked; therefore, three different sub-systems for attacking the most widely used cryptographic algorithms, have been implemented. As the implementation results clearly demonstrate, this approach is more than 1000 times faster than the corresponding software one, while they have the same probability of success in cracking the passwords. Furthermore, our highly parallel architecture scales very well and can be efficiently utilized in the future generations of the even larger reconfigurable devices.

II. Related Work

Few hardware systems have been proposed so far for the cryptanalysis of the encrypted passwords. Two such systems are briefly described below:

1. An FPGA system implementing Hellman's method was proposed by Quisquater [8]. This system could attack passwords encrypted with the 40-bit DES algorithm. Quisquater also presented cost estimations for the cryptanalysis of a full DES (with 56 bits). A more generic full cost analysis of the time-memory trade-off with and without distinguished points has been provided by Wiener in [10].
2. More recently, N. Mentens et al. implemented in an FPGA platform an efficient system for cryptanalysis of the UNIX password hashing, using the Rainbow Table approach [4], [9].

Both systems can attack passwords that are encrypted only by a certain algorithm (DES), while their approach is up to 50 times faster than the software one.

The proposed innovative device differs from the existing systems since:

1. We have implemented a framework that can attack passwords encrypted with a number of different algorithms, which are not covered by any of the existing systems.
2. Our system is more than 1000 times faster than the existing software approaches, mainly due to the high-level of parallelism employed.

The architecture of an initial prototype of our system attacking only a single algorithm (i.e. LM Hash) has been presented in [13]. However, this paper describes the architecture of a more generic system which is able to attack multiple algorithms, whereas significantly more experiments have been conducted in this latest full version of the device.

III. Rainbow Tables and Cryptographic Algorithms

In this section, the Rainbow Tables as well as the three cryptographic algorithms, which are attacked by our system, are briefly described.

A. Rainbow Tables

A Rainbow Table is defined as a compact representation of related plaintext password sequences (or chains).

Each chain starts with an initial password, which is processed by a hash function. A reduction function is then applied to the resulting hash and the outcome is a different plaintext password. This process is repeated for a fixed number of times. The initial and final passwords of the chain comprise a Rainbow Table entry and they are called Starting and End Points respectively.

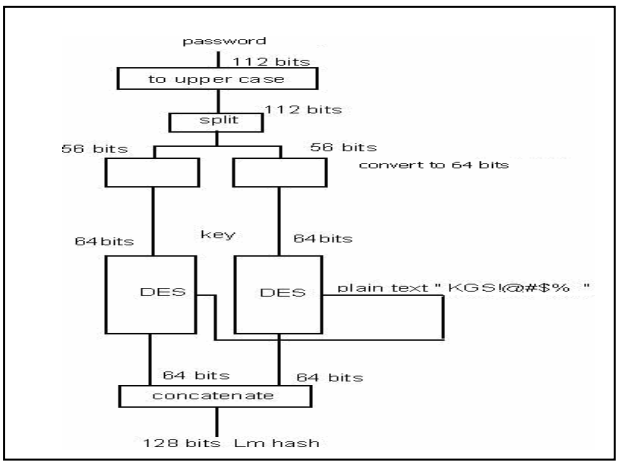
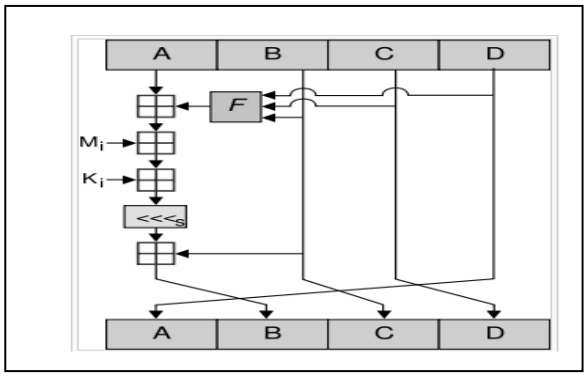


Figure 1. Diagramm of LM Hash.

Figure 2. Operation of a single MD5 round.



The recovering of a password using a Rainbow Table consists of two steps. First, the password hash is processed by the reduce-hash sequence, described in the last paragraph. The structure of the table and its reduction function guarantees that the running password will match a final password on one of the chains (not necessarily the correct one). Second, the iteration is repeated, starting with this initial password, until the original hash is found. The password used at the last iteration is the one being recovered.

B. LM Hash (LAN Manager Hash)

LM Hash (Fig. 1) is an encryption/hash algorithm used by Windows in order to store the passwords that are fewer than 15 characters long. The LM hash is computed as follows:

1. The password is converted to uppercase.
2. This password is either null-padded or truncated to 14 bytes.
3. The "fixed-length" password is split into two 7-byte segments.

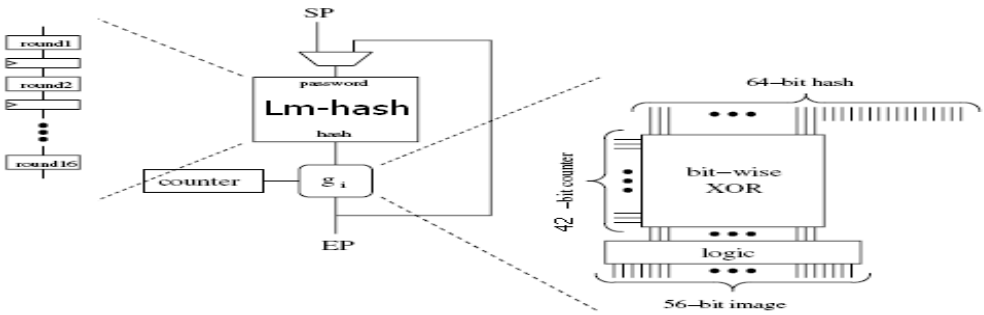


Figure 3. Pipeline architecture, for the creation of Rainbow chain.

4. These values are used to create two DES keys, one from each 7-byte segment, by converting the seven bytes into a bit stream and inserting a parity bit after every seven bits. This generates the 64 bits needed for the DES key.
5. Each of these keys is used to DES-encrypt the constant ASCII string "KGS!@#$%", resulting in two 8-byte cipher texts.
6. The two cipher texts are concatenated to form a 16-byte value, which is the resulting LM hash.

Although it is based on DES, which represents an efficient block cipher, the LM hash can easily be cracked due to two main weaknesses: First, passwords longer than seven characters are divided into two segments and each segment is hashed separately. Second, all lower case letters in the password are changed to upper case before the password is hashed.

C. MD5 (Message-Digest algorithm 5)

MD5 was designed by Ron Rivest in 1991 [11], in order to replace an earlier hash function (MD4). It is used for hashing passwords in certain UNIX distributions. MD5 processes a variable-length message into a fixed-length output of 128 bits.

The input message is divided into 512-bit blocks (sixteen 32-bit little-endian integers); the message is padded so that its length is divisible by 512. The padding works as follows: first, a single 1 is appended to the end of the message. This is followed by as many 0 as required in order to bring the length of the message up to 64 bits less than a multiple of 512. The remaining bits are filled up with a 64-bit integer, representing the length of the original message in bits.

The main MD5 algorithm operates on a 128-bit state, divided into four 32-bit words, denoted as "a", "b", "c" and "d". These are initialized to certain fixed constants. The main algorithm then operates on each 512-bit block. The processing of a message block consists of four similar stages that are called rounds. Each round is composed of sixteen similar operations, based on a non linear function F, modular addition, and left rotation. Fig. 2 illustrates one operation within a round. Four possible functions F exist; a different one is used in each round:

R1 (b,c,d) = bc+b'd, R2 (b,c,d) = bd+cd',
R3 (b,c,d) = bXORcXORd , R4 (b,c,d) = cXOR (b+d').

D. SHA-1 (Secure Hash Algorithm)

The original specification of the algorithm was published in 1993 as the Secure Hash Standard [12]. SHA-1 produces a 160-bit digest from a message with a maximum length of 2^{64-1} bits and is based on principles similar to those used by Rivest in the design of the previously described MD5 digest algorithm. SHA-1 is also used in hashing UNIX passwords in certain distributions such as Free-BSD and OSF in the past.

IV. HARDWARE IMPLEMENTATION

The core of proposed high-end system is represented by the module which implements a single rainbow chain, such as the one illustrated in Fig. 4. It should be noted that a Rainbow Table consists of numerous parallel independent such chains. When implementing this module it is crucial to select the most appropriate function in order to map the cipher text to a key (i.e. the mask function or "g_i" in the figures). For such a function several options exist, for example: permutations (i.e. P-boxes), XOR functions and bit swap functions.

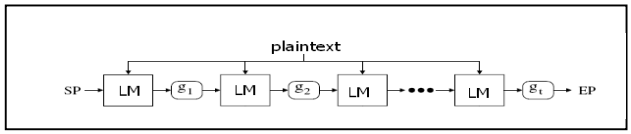
From the point of view of hardware complexity, all three suggested options are suitable, since they can easily be implemented in limited resources of a reconfigurable device. However, in the Rainbow Tables a chain contains many different mask functions, therefore any overhead in the control logic should also be minimized. To this aim, the XOR function seems to be the most suitable solution, since it does not need any specific control. Moreover, in the present case due to the limited alphabet, permutations may not even offer enough choices for the different needed masks. Since the complexity of our key space is 2^{42} we chose to throw away the last sixteen bits of the 56-bit hash result and to XOR with a 42-bit counter. In this way, just one generic mask function can be employed having different unique states (the states of the 42-bit counter), and resulting effectively in a total of 242 different mask functions. Finally, the 42-bit output of the XOR function passes through some logic gates to obtain a 56-bit result, which is a valid 56-bit key. In the LM Hash case a valid hash contains only capitals and numbers, while for MD5 and SHA-1 it contains capital and small letters, numbers and the characters " \" and "." Fig. 3 depicts the general architecture of our proposed design. In order to construct a chain, an alternating sequence

of block cipher computations and mask functions is applied, using a simple feedback loop.

Additionally, the initial keys (i.e. Starting Points or SPs), are constructed in the reconfigurable device, instead of loading them from an external source. In this way we save the PC-FPGA communication overhead imposed by the existing solutions that get the SPs from a PC. The SPs are created by simple 42-bit counters.

The pairs of Start-End text (i.e. SP, EP) are stored in a table where the End Points (EPs) operate as pointers, since according to the specifications of the Rainbow Tables, the results should be classified with respect to the EP.

Figure 4. Schematic representation of a Rainbow chain.



Since the EPs are 56-bits long and our on-chip memory has a maximum depth of 217 (the maximum that can be produced in our high-end reconfigurable device when the width is 114 bits, so as to fit an SP, EP pair and some flags), a hash function is utilized in order to associate the 56 bits with the 17-bit addresses of the on-chip memories. The hash function that was used, according to our measurements, creates randomly distributed addresses. This is illustrated in Fig. 5.

This function utilizes a number of random tables upon which the corresponding is performed (qi), while "x" are the input bits and "a" are the output bits. Since up to 128K entries can be inserted in our table, up to 128K distinct rainbow chains can be created at a given time. In order to produce larger Rainbow Tables without any additional overhead, we take advantage of the fact that the on-chip memories (BRAMs) of the high-end FPGAs are dual-ported. Therefore, a simple DRAM controller has been implemented as well. It reads out the ready chains/tables from one of the ports of the BRAM, while the processing units continuously store the new data by utilizing the second port. Thus, we interleave the creation of the chains with their off-loading to the external memories. In order to know when a chain has been moved to the external DRAM and thus this entry can be re-used, certain flags have been defined in each table entry. Obviously this scheme produces further collusions; for example a new chain may try to over-write one, which has not been uploaded to the external memory yet. However, based on our measurements and since the DRAM controller is very fast, aware of the chains that are ready and it reads out them directly (it keeps a state with the ready-chains), those collisions are minimal in all our experiments. After producing all the entries, the actual recovery process is performed. Within this process, the most appropriate chain is first figured out. Following this, staring from the EP, we trace back to the different output values produced by our modules until the key is found. The recovering of a single key in a rainbow chain takes about "t(t-2)/2" function evaluations. When the occasion of an SP that does not lead to the right key is captured, there is a so-called "false alarm". This is due to the fact that the key can be a part of a chain with the same endpoint, but it may not belong to the actual table.

A. Micro-Architecture

The actual micro-architecture of our system for cracking the passwords encrypted with the LM-Hash function is depicted in Fig. 6. Our system contains certain counters producing the initial SPs and some logic control functions in order to ensure that the outcome of the counter will be a string of valid characters for the specific algorithm.

Moreover, a sub-module producing the hash cipher text (a DES module in this case) has been implemented, as well as a counter producing 42-bit quantities, which will be XOR-ed with the outcome of the DES sub-module. Additional implementations are some logic control functions to ensure that the outcome of the XOR will be a string of valid characters for the specific algorithm, a MUX, determining whether the algorithm's input is the SP or the result of the algorithm and a block consisting of the memory and the address pointer of the memory.

Finally there is a controller that appropriately set the signals for writings of the memory and the output of the multiplexer of the chain production stage.

The corresponding modules for the attacking of the passwords that are hashed by MD5 and SHA-1 are very similar to the one presented in Fig. 6; the only difference is that instead of the DES block, an MD5 and an SHA-1 was employed accordingly.

V. Implementation Characteristics

The described design has been targeted to a Xilinx XCV5VLX330T FPGA chip on a HitechGlobal card. This design has been implemented manually in VHDL and we have synthesized, mapped, placed and routed it using Xilinx ISE 10.1 tool. The hardware complexity of our design is demonstrated in Table I. As clearly demonstrated in this table, in a high-end FPGA up to 64 parallel engines can be fitted, for creating multiple rainbow chains in parallel. Since the more chains we create the higher the probability to crack a certain password, numerous parallel engines are required.

Regarding the performance of the system, the maximum clock frequency supported by each system is presented in Table II. The reduction in the clock frequency, when 64 engines are fitted in a single device, is less than 5% than in the case that only one engine is implemented.

VI. Experimental Results

In order to measure the performance of our approach several real-world experiments have been carried out with different configurations of the system and various data-sets. We concentrated on the results of attacking to passwords encrypted by LM Hash, since we strongly believe that this is the most important case. Moreover, the results of SHA-1 and MD5 are very similar to those of LM Hash (in fact they are slightly better), mainly in terms of the acceleration achieved when compared with the corresponding software approach.

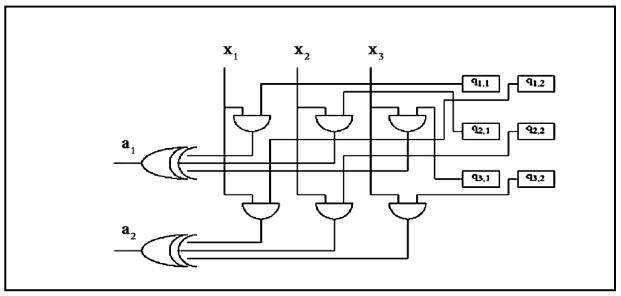


Figure 5. Hash Function

TABLE I. HARDWARE COMPLEXITY OF THE PROPOSED SYSTEM

	Number of Logic Cells for 1 engine/ Percentage of high-end FPGA	Number of Logic Cells for 64 engines/Percentage of high-end FPGA	Percentage of BRAMs for 64 engines
LM Hash	3,553 / 1%	234,532 / 70%	69%
MD5	3,245 / 1%	212,856 / 67%	69%
SHA-1	3,348 / 1%	223,648 / 69%	69%

TABLE II. PERFORMANCE CHARACTERISTICS

	Maximum Frequency 1 engine	Maximum Frequency 64 engines
LM Hash	192.308MHz	181.543MHz
MD5	82.680MHz	75.406MHz
SHA-1	126.744MHz	114.744MHz

TABLE III. PERFORMANCE OF A SINGLE HARDWARE ENGINE

Number of chains	Probability of Success	Time for Constructing a single chain (seconds)	Total Time Needed (days)
131072	0.86	5.83	9
16384	0.86	45.97	9
32768	0.86	23.42	9
65536	0.86	8.84	7
65536	0.36	2.65	2

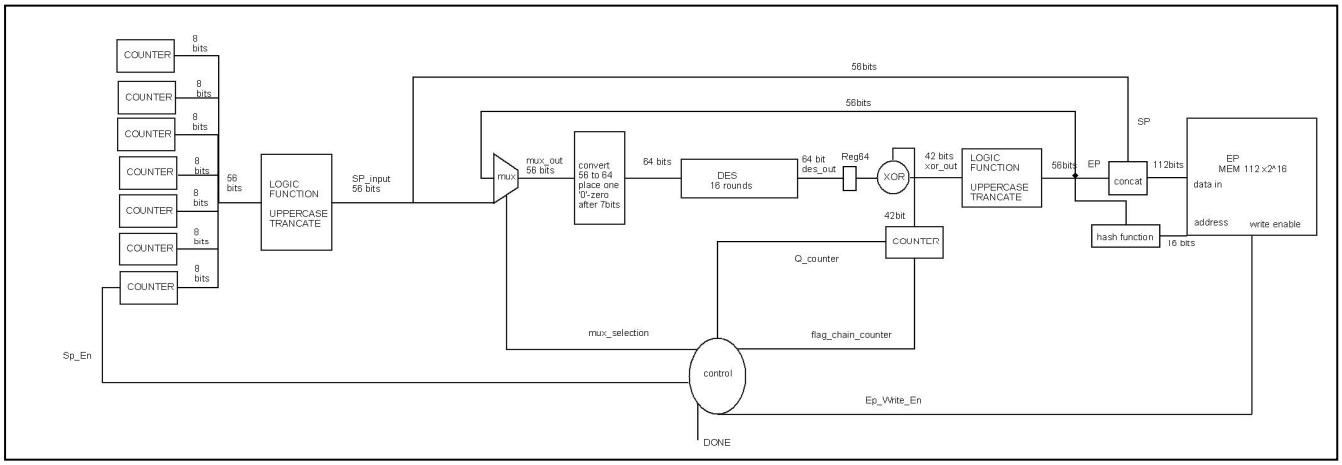


Figure 6. Microarchitecture of module creating the Rainbow Tables for attacking LMHash.

TABLE IV. PERFORMANCE COMPARISON OF SOFTWARE AND HARDWARE APPROACH WHEN A SINGLE HARDWARE ENGINE IS USED

Number of chains	Probability of Success	Time for Constructing a single chain Hardware (ns)	Time for Constructing a single chain Software (ns)	Total Time Needed Hardware (seconds)	Total Time Needed Software (seconds)	Speedup of Hardware over Software
5	0.000000056	442	28,966,912	0.03	1.02	35
1000	0.087	88,400	5,793,382,400	6	223	38
10000	0.14	884,000	57,933,824,000	58	1565	27
20000	0.53	1,768,000	115,867,648,000	116	5579	48
100000	0.59	8,840,000	579,338,240,000	579	11239	19

The real time needed for creating a Rainbow Table with certain characteristics and when only one hardware engine is employed, is first demonstrated. The important characteristics of a Rainbow Table are its size and its "probability of success". The latter is calculated by a certain formula (taking into account also the size of each chain), derived by Oechslin and stating the possibility of cracking a certain password using the specified Rainbow Table. Our measurements are presented in Table III. It is clear from those measurements that even our innovative hardware system needs several days in order to create an efficient Rainbow Table, when utilizing a single engine.

We have also measured the time needed for creating certain tables in a Dual-Core Pentium working at 2.66GHz and executing the optimal code provided by the inventor of the Rainbow Tables. We have removed any system-calls from the code and the times listed are those given by the "time" command of Linux for User CPU-time. In addition we have created the exact same tables in our innovative reconfigurable system. The times measured in each case are listed in Table IV together with the speedup that our system triggers. As the results of this table demonstrate, even by using a single processing unit of our FPGA-system, the proposed system is at least 19 times faster than a standard PC, when executing the exact same algorithm.

Since we can fit up to 64 processing engines in a single device, the increase in the performance achieved when more engines are utilized has also been measured. In this experiment, a table consisting of 131,072 chains with a probability of success of over 85% has been created (the heaviest case in terms of processing power). The results are delineating in Fig. 7. Based on those, we claim that our approach scales very well, since the time needed for creating the table is reduced almost linearly with the numbers of the processing units employed.

Finally, we have measured the speedup achieved when 1, 32 and 64 hardware engines are utilized in a large high-end FPGA, against the performance of a new dual-core PC which executes the most optimized version of this same algorithm. These worst-case results are demonstrated in Fig. 8. It is clear that our system can be more than 1000 times faster, when 64 units are utilized, than a PC when creating efficient Rainbow Tables for cracking passwords.

VII. CONCLUSIONS

The "Rainbow Table" is probably the most efficient scheme for password cracking, nowadays; it is a new variant of the well-know Hellman's original cracking algorithm with even better performance. The main drawback of this system is that in order to create an efficient Rainbow Table on a single PC, up to a couple of months may be needed.

In this paper a hardware system implemented on a state-of-the-art reconfigurable device is presented, This system can create such tables up to 1000 times faster than a new PC. The huge speedup is achieved by using a highly parallel architecture. As our results clearly demonstrate, this architecture scales very well, therefore we strongly believe that it can also be efficiently utilized in the next-generation, even larger reconfigurable devices, consisting of more than 100 Million system gates.

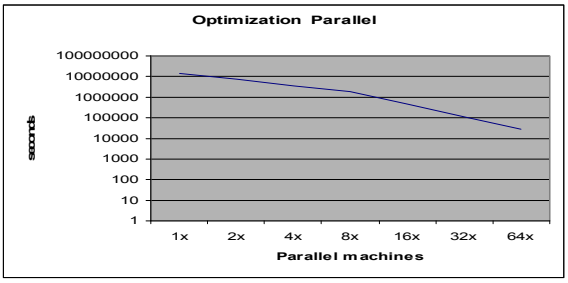


Figure 7. Time needed for creating a single table when the number of parallel engines is altered

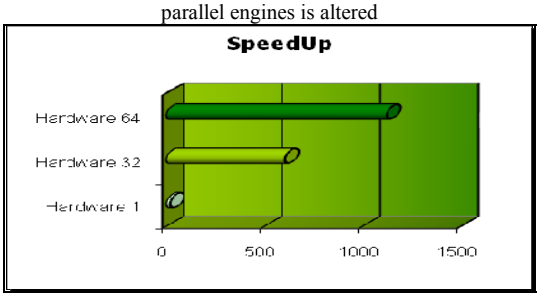


Figure 8. Speedup against software for different number of processing units.

REFERENCES

[1] G. Avoine, P. Junod, and P. Oechslin, "Characterization and Improvement of Time-Memory Trade-Off Based on Perfect Tables," ACM Transactions on Information and System Security vol. 11, no 4, pp. 1-22, July 2008.

[2] A. Fiat, and M. Naor, "Rigorous time/space tradeoffs for inverting functions," Proc. of the 23rd Annual ACM Symposium on Theory of Computing, 1991, pp. 534-541.

[3] J. Borst, B. Preneel, and J. Vandewalle, "On the memory trade-off between exhaustive key-search and table precomputation," Proc. of the 19th Symposium on Information Theory in the Benelux, The Netherlands 1998, pp. 111-118.

[4] N. Mentens, L. Batina, B. Preneel, and I. Verbauwhede, "Cracking Unix passwords using FPGA platforms, SHARCS'05, Paris, February 24.-25. 2005.

[5] M. Hellman, "A cryptanalytic time-memory trade off," IEEE Trans. Inform. Theory, vol. 26, no 4, pp. 401-406, 1980.

[6] P. Oechslin, "Making a faster cryptanalytic time-memory trade-off," Proc. of Advances in Cryptology, 23th Annual International Cryptology Conference, Santa Barbara, CA. USA, 2003.

[7] K. Kusuda, and T. Matsumoto, "Optimization of time-memory trade-off cryptanalysis and its application to DES," IEICE Trans. Fundamentals E79-A, vol. 1, Jan 1996, pp. 35-48.

[8] J. Quisquater, F.-X. Standaert, G. Rouvroy, and J.D. Legat, "A cryptanalytic time-memory trade-off: First FPGA implementation," FPL 1998, Tallinn, Estonia, Au31 - Sep 3, 1998.

[9] N. Mentens, L. Batina, B. Preneel, and I. Verbauwhede, "Time-Memory Trade-Off Attack on FPGA Platforms: UNIX Password Cracking," ARC 2006, Delft, The Netherlands, March 1-3, 2006

[10] M. J. Wiener. "The full cost of cryptanalytic attacks," Journal of Cryptology, vol. 17, no 2, pp. 105-124, 2004.

[11] R. Rivest. "The MD5 Message-Digest Algorithm," RFC 1321, 1992.

[12] Bruce Schneier, Applied Cryptography, 1995,ISBN:0-471-11709-9

[13] K. Theoharoulis, C. Manifavas, and I. Papaefstathiou, "High-End Reconfigurable System for fast Windows' Password Cracking," IEEE FCCM, Napa, California, U.S.A. April 5-7 2007