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

Konstantinos Theocharoulis, Charalampos Manifavas, Ioannis Papaefstathiou

Electronic and Computer Engineering Department (ECE),
Technical University of Crete,
Chania, Greece

IEEE FPL 2010

Presentation Overview

- Motivation
- Rainbow Tables
- Hashing algorithms
- System's Architecture
- FPGA Implementation
- Experimental Results
- Related Work
- Conclusions

Motivation & Purpose

- One of the most efficient methods for cracking passwords, encrypted by different cryptographic algorithms, is the one based on “rainbow tables”
- The heavy computational load of this algorithm results in extremely large processing times in the range of months!

Crypto-attack algorithms

- The most important attack on a block cipher analyzes the mapping of the key to the ciphertext; this mapping is always a one-way function.
  - If such a cryptographic function has 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) we can precompute and store 2^n input and output pairs in a table and in order to invert a particular value, we just look up the pre-image in the table; in this case the function inverting requires only a single table lookup.
- In-between those two solutions lies the Hellman algorithm; the precomputation time of this approach is still in 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.

Rainbow Tables

- In 2003 and more recently in 2008, Oechslin further expanded Hellman's approach and suggested the "rainbow tables"
- Those tables are utilized in the precomputation task of the algorithm.
  - ✓ As listed 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".

Rainbow Tables (2)

- A rainbow table is 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.

Rainbow Tables (3)

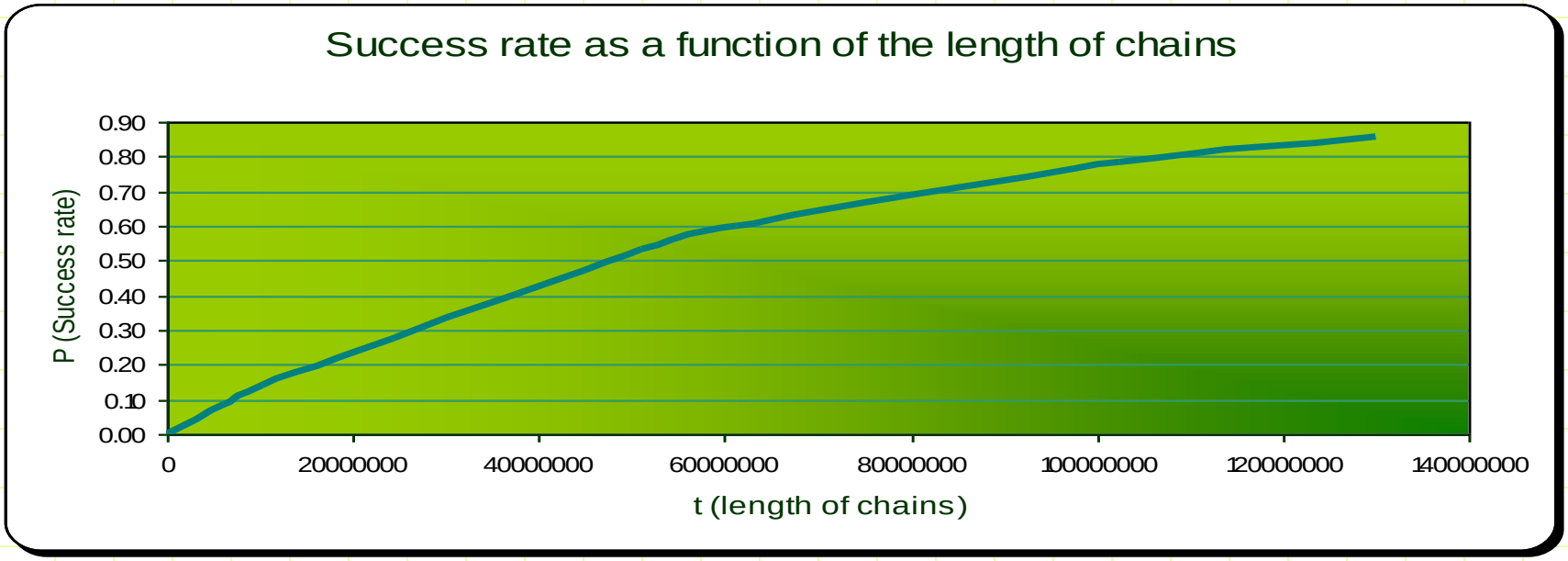
- Recovering a password using a rainbow table consists of two steps.
  1. First, the password hash is processed by a reduce-hash sequence.
  2. Second, the iteration is repeated starting with this initial password until the original hash is found. The password used at the last iteration is the password being recovered.

Success Rate of Rainbow Tables

$$P = 1 - \prod_{i=1}^{t} \left(1 - \frac{m_i}{N}\right)$$

$$m_{n+1} = N\left(1 - e^{-\frac{m_n}{N}}\right)$$

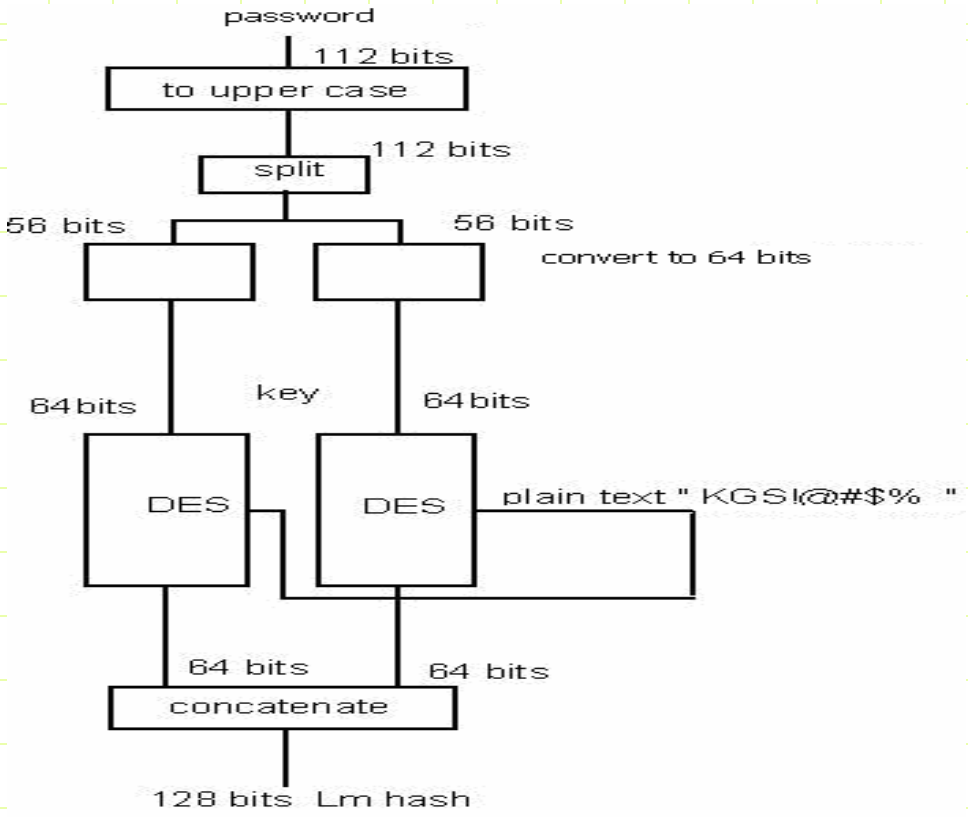
Where m_1 = m is the memory size in which the tables are stored and N the size of the search(i.e. password) space



LMHash

- LM Hash is an encryption/hash algorithm used by Windows. 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.
  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 zero 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 ciphertexts.
  6. The two ciphertexts are concatenated to form a 16-byte value, which is the resulting LM hash.

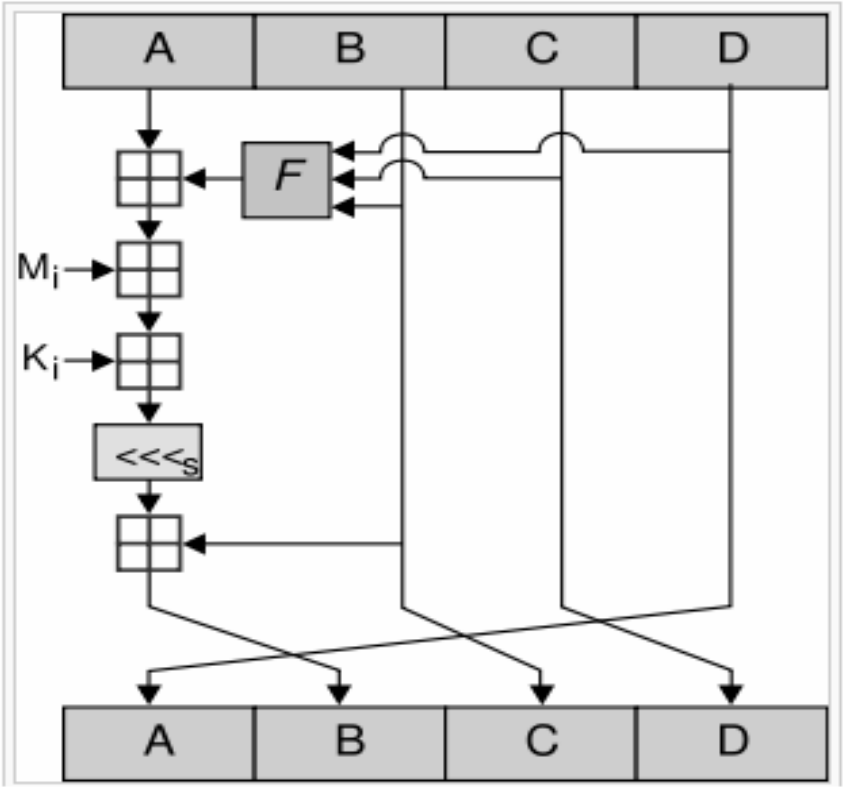
LMHash High-level overview



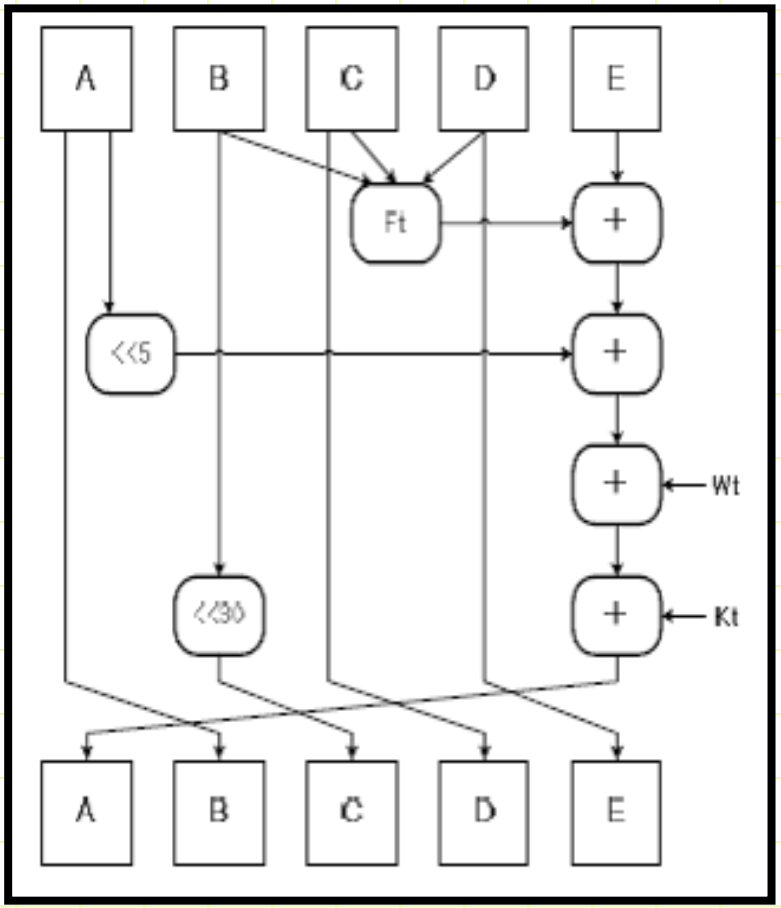
MD5

- MD5 was designed by Ron Rivest in 1991
- MD5 processes a variable-length message into a fixed-length output of 128 bits. The input message is broken into 512-bit blocks (sixteen 32-bit little-endian integers); the message is padded so that its length is divisible by 512.
- The main MD5 algorithm operates on a 128-bit state, divided into four 32-bit words, denoted *a*, *b* ,*c* and *d*. These are initialized to certain fixed constants.
- The processing of a message block consists of four similar stages that are called *rounds*; each round is composed of 16 similar operations based on a non-linear function *F*, modular addition, and left rotation:
  $\mathbf{R_1}\ (b,c,d) = bc + b'd \qquad \boldsymbol{R_2}\ (b,c,d) = bd + c\ d'$
- $\boldsymbol{R_3}\ (b,c,d) = b \text{ xor } c \text{ xor } d \qquad \boldsymbol{R_4}\ (b,c,d) = c \text{ xor } (b + d')$

MD5 High-Level overview

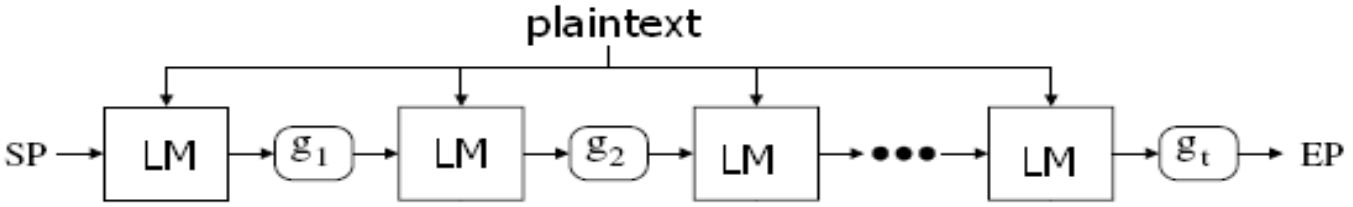


SHA-1



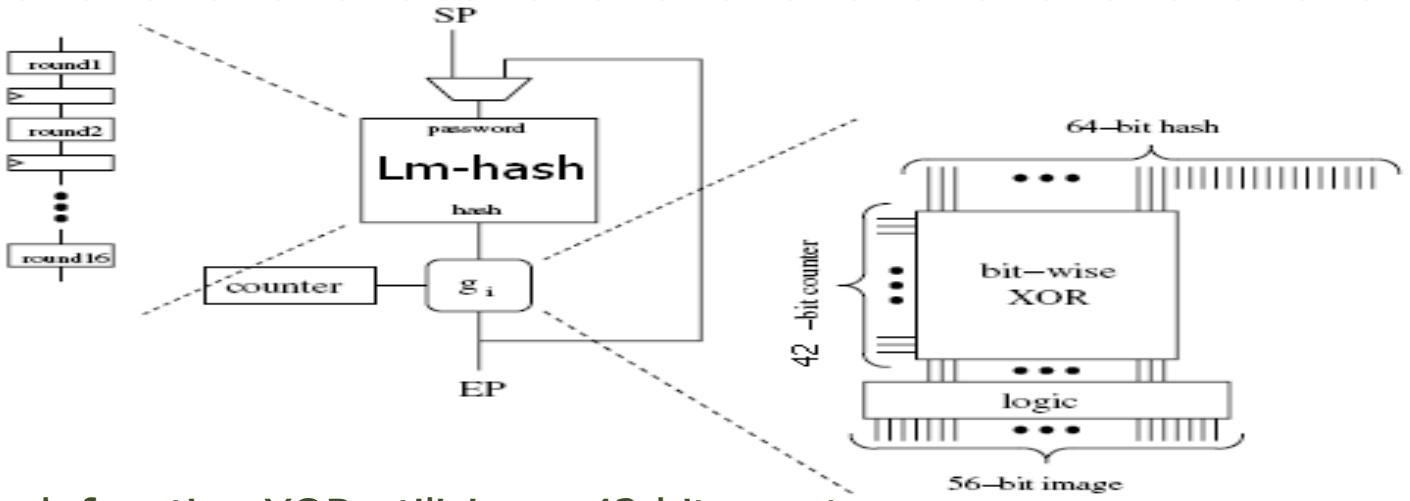
Our Design Choices

- The core of our high end system is the module implementing a single rainbow chain



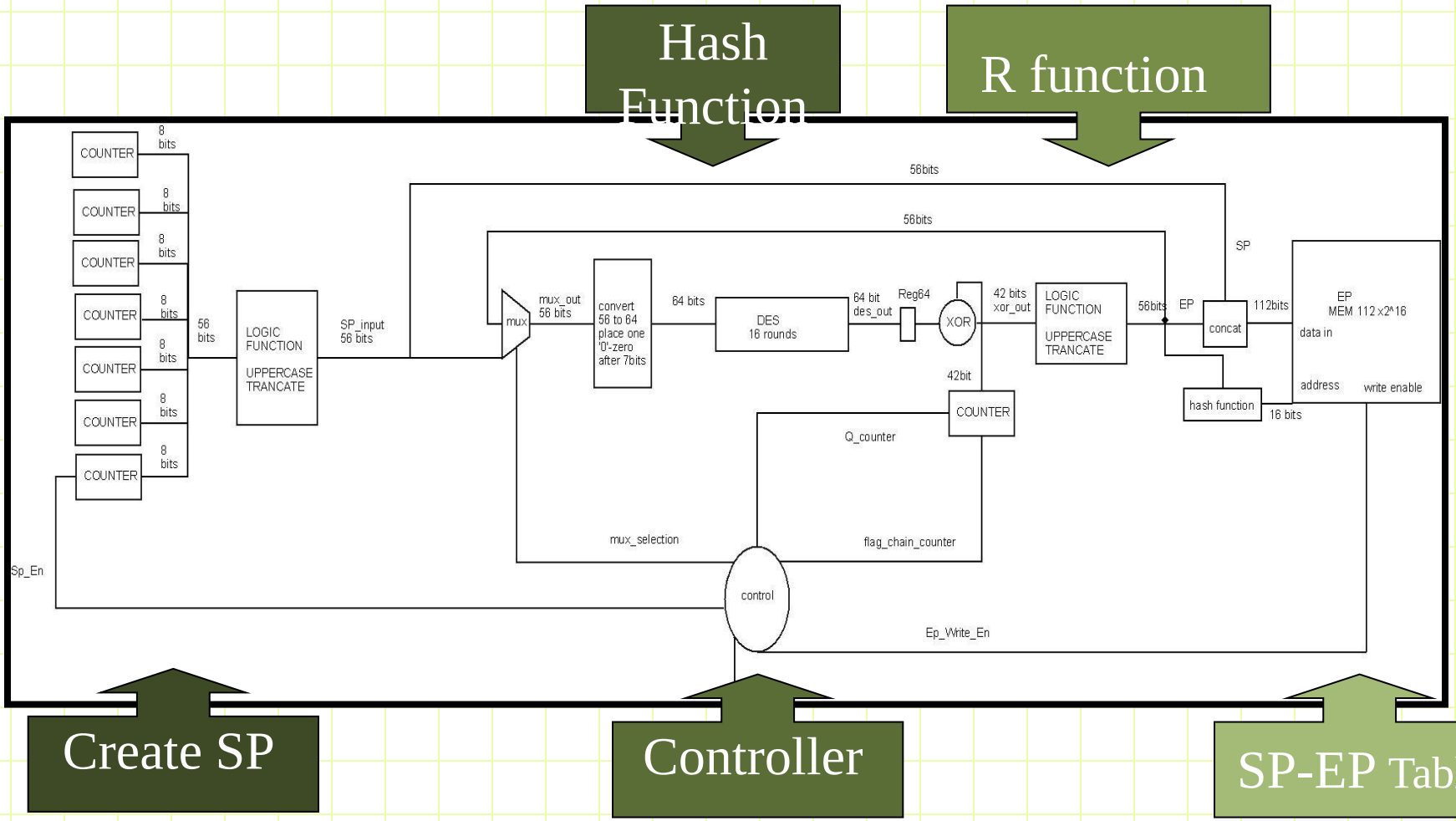
- For g_i the algorithm can utilize one of the following
  1. permutations (i.e. S-boxes)
  2. xor functions
  3. bit swap functions
     - ✓ We use xor functions so as to simplify the control unit

Design Choices(2)



- Mask function XOR utilizing a 42 bit counter
- SP is produced in Hardware by a 42-bit counter
- The SP-EP pairs are initially stored in a on-chip memory, for which EPs act as indexes and then a clever DRAM controller puts them off-chip

Micro-architecture



Hardware Performance

	Minimum period	Maximum Frequency
LM hash	5.508ns	181.543MHz
MD5	13.261ns	75.406MHz
SHA-1	8.715ns	114.744MHz

Hardware cost

Single Machine

	Number of Slice LUTs	Number of Block RAM/FIFO
LM hash	1%	69%
MD5	1%	69%
SHA-1	1%	69%

64 Parallel Machines

	Number of Slice LUTs	Number of Block RAM/FIFO
LM hash	70%	69%
MD5	65%	69%
SHA-1	67%	69%

Performance Results of HW and SW

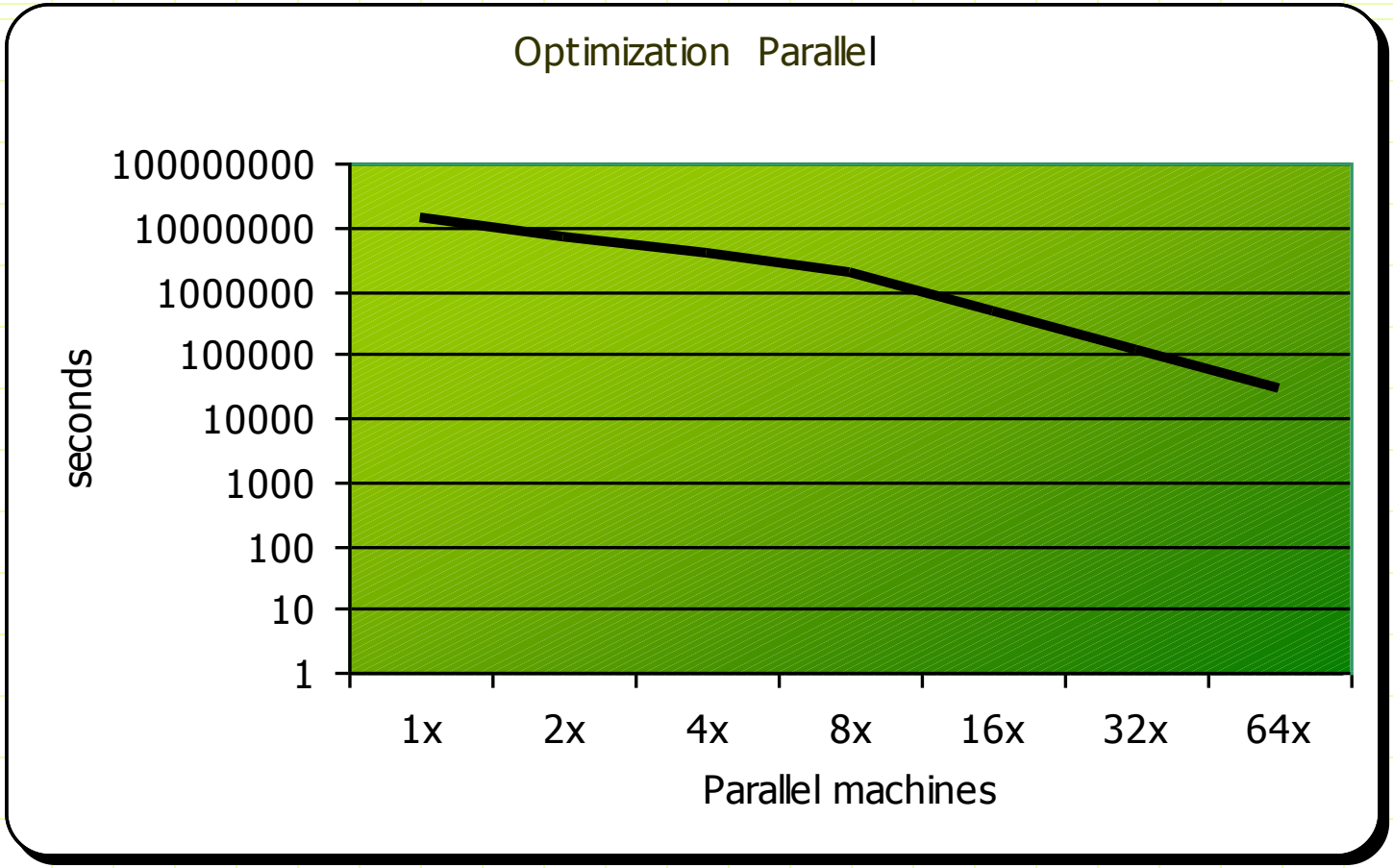
Number of chains	Probability of Success	Time for Constructing a single chain Hardware (ns)	Time for Constructing a single chain Software (ns)	Total Time Needed Hardware (seconds)	Total Time Needed Software (seconds)	Speedup of Hardware over Software
5	0.00000056	442	28,966,912	0.03	1.02	35
1000	0.087	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

SW : Core2Duo at 2.66GHz (optimized code provided by Rainbow Tables' inventor
HW : Virtex5-240LX

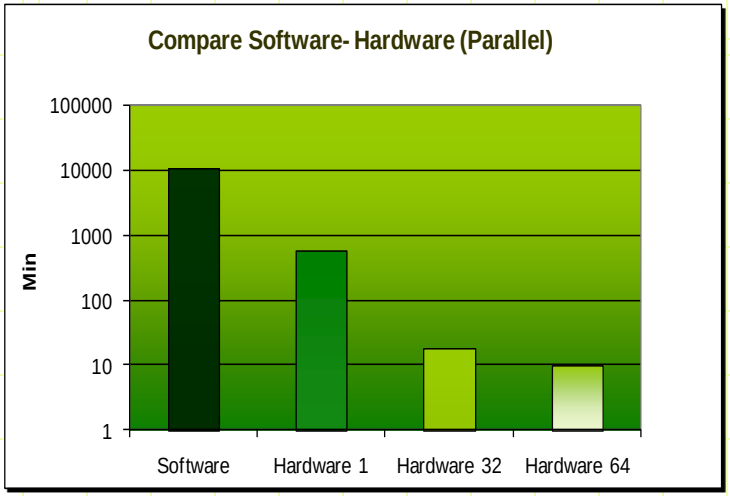
Performance of a single hardware engine

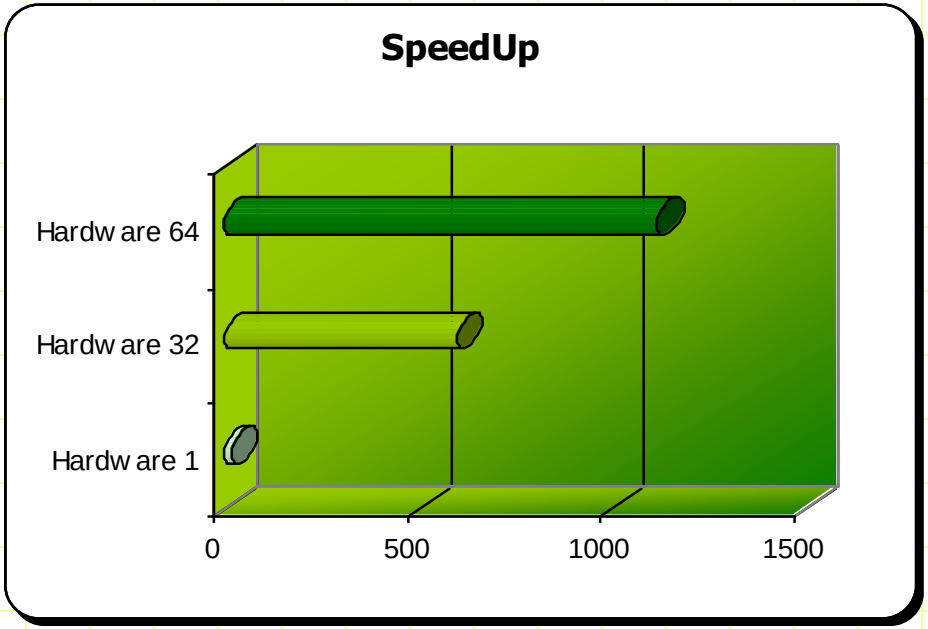
Number of chains	Length of Chains	Probability of Success	Time for Constructing a single chain (seconds)	Total Time Needed (days)
131072	66,000,000	0.86	5.83	9
16384	520,000,000	0.86	45.97	9
32768	265,000,000	0.86	23.42	9
65536	100,000,000	0.86	8.84	7
65536	30,000,000	0.36	2.65	2

Performance of Parallel machines



HW vs SW


Compare Software- Hardware (Parallel)
100000
10000
1000
100
10
1
Min
Software
Hardware 1
Hardware 32
Hardware 64


SpeedUp
Hardw are 64
Hardw are 32
Hardw are 1
0
500
1000
1500

Related Work

- [1] J. Quisquater, F.-X. Standaert, G. Rouvroy, and J.D. Legat. “A cryptanalytic time-memory trade-off: First FPGA implementation”. In Proceedings of the 8th International Workshop on Field-Programmable Logic and Applications (FPL), volume 2438 of Lecture Notes in Computer Science, pages 780–789, 2002.
- [2] Nele Mentens and Lejla Batina and Bart Preneel and Ingrid Verbauwhede, “Time-Memory Trade-Off Attack on FPGA Platforms: UNIX Password Cracking”,In Proc. of ARC 2006, volume 3985 pp 323-334, 2006

Comparison with FPGA-based systems

- Our system is at least 20 times faster than both proposed systems.
  - ✓50 times faster than [1]
  - ✓20 times faster than [2]
- Our framework can attack passwords hashed with a number of different algorithms, which are not covered by any of the existing systems.

Conclusions

- Our system is the fastest FPGA-based system for attacking passwords hashed with different algorithms
- Our speedup is triggered by employing high level of parallelism.
- Our system is more than 1000 times faster than the standard software approaches and more than 20 times faster than the similar systems implemented in reconfigurable devices

Note : It is a proprietary system that will not be used for non-research purposed and it will not be distributed anywhere ☺

Thank you!

Questions ?