Speeding up GPU-based password cracking

SHARCS 2012

Martijn Sprengers[1,2]   Lejla Batina[2,3]

Sprengers.Martijn@kpmg.nl
KPMG IT Advisory[1]
Radboud University Nijmegen[2]
K.U. Leuven[3]

March 17-18, 2012

Who am I?

Professional life

- Ethical hacker
  - KPMG IT Advisory
- Education
  - Master Computer Security at the Kerckhoffs Institute
- Expertise and experience
  - Computer and network security
  - Password cracking
  - Social Engineering

Spare time

Cracking password hashes with GPU's

Goals

- Show how password hashing schemes can be efficiently implemented on GPU's
- Impact on current authentication mechanisms
- Pose relevant questions immediately but save discussions for the end

Outline

- Background information on MD5-crypt and GPU
- Optimizations and speed-ups
- Results and improvements

Cracking password hashes with GPU's

Goals

- Show how password hashing schemes can be efficiently implemented on GPU's
- Impact on current authentication mechanisms
- Pose relevant questions immediately but save discussions for the end

Outline

- Background information on MD5-crypt and GPU
- Optimizations and speed-ups
- Results and improvements

Motivation

Why password hashing schemes?

- Database leakage
  - Disgruntled employee
  - SQL injections
- Accessible storage
  - 'SAM' file (Windows)
  - 'passwd' file (Unix)

Why exhaustive search?

- Humans and randomness → ☹
- Humans and memorability → ☹
- Limited keyspace → enables exhaustive search

Motivation

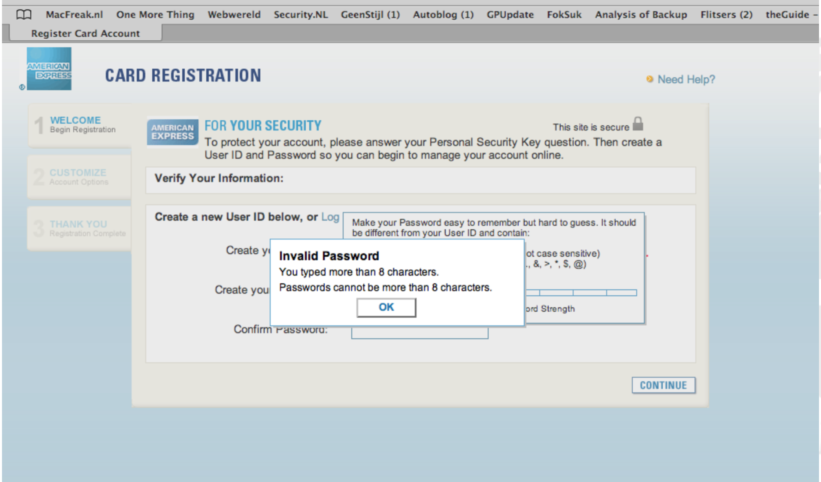
Why password hashing schemes?

- Database leakage
  - Disgruntled employee
  - SQL injections
- Accessible storage
  - 'SAM' file (Windows)
  - 'passwd' file (Unix)

Why exhaustive search?

- Humans and randomness → ☹
- Humans and memorability → ☹
- Limited keyspace → enables exhaustive search

Why exhaustive search?



Motivation

Why MD5-crypt?

- Commonly used
  - Default Unix scheme, Cisco routers, RIPE authentication
- Basis for other hashing schemes and frameworks
  - SHA-crypt, bcrypt, PBKDF2

Why GPU?

- New API's support native arithmetic operations
- Designed for highly parallelized algorithms

Motivation

Why MD5-crypt?

- Commonly used
  - Default Unix scheme, Cisco routers, RIPE authentication
- Basis for other hashing schemes and frameworks
  - SHA-crypt, bcrypt, PBKDF2

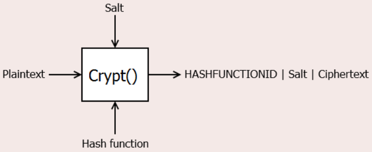
Why GPU?

- New API's support native arithmetic operations
- Designed for highly parallelized algorithms

Password hashing schemes

Definition

$$PHS : \mathbb{Z}_2^m \times \mathbb{Z}_2^s \to \mathbb{Z}_2^n$$



Properties

- *Correct use of salts*
  Prevents from time-memory trade-off attacks
- *Slow calculation*
  Key-stretching
- *Avoid pipelined implementations*
  Hashing k passwords with the same salt should cost k times more computation time than hashing a single password

Avoid pipelined implementations

```
var salt = "btediz(KD+$$40";
this.setKey = function(key) {
    encryptionKey = key ;
        if ( !key )
            encryptionKey = null ;
        else {
            for(var i = 0; i < 100; i++)// loop to improve encryption strength
                encryptionKey = MD5(salt + encryptionKey);
        }
```

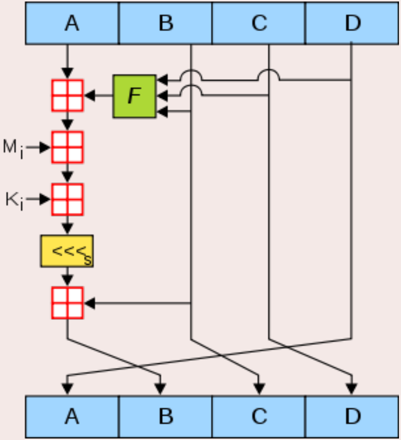
MD5-crypt

MD5-crypt

MD5-crypt("somesalt","password") =
1somesalt$W.KCTbPSiFDGffAGOjcBc.

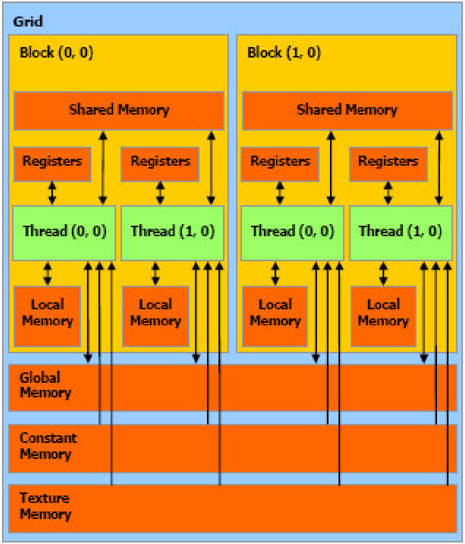
- Key-stretching
  - 1002 calls to *MD5-compression function*
  - Concatenates password, salt and intermediate result pseudo randomly

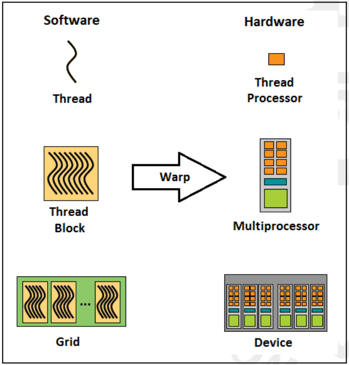
MD5-compression round



Source: Wikipedia

CUDA and memory model





Attacker model

Assumptions

- Attacker model
  - Plaintext password recovery
  - Exhaustive search (ciphertext only)
  - No time-memory trade-off
- Hardware
  - One CUDA enabled GPU: NVIDIA GTX 295
  - 480 thread processors
  - 60 streaming multiprocessors
- Password generation
  - Password length < 16
  - Performance measured in unique password checks per second

Our optimizations

Our optimizations

- Memory → Fast shared memory
- Algorithm wise → Precompute intermediate results
- Execution configuration → Block- and gridsizes
- Maximizing parallelization → Password hashing is embarrassingly parallel
- Instructions → Modulo arithmetic is expensive
- Control flow → Branching is expensive

Algorithm optimizations

- Password length < 16 → One call to MD5compress()
- Password length $<< 16$ → Precompute intermediate results

Our optimizations

Our optimizations

- Memory → Fast shared memory
- Algorithm wise → Precompute intermediate results
- Execution configuration → Block- and gridsizes
- Maximizing parallelization → Password hashing is embarrassingly parallel
- Instructions → Modulo arithmetic is expensive
- Control flow → Branching is expensive

Algorithm optimizations

- Password length < 16 → One call to MD5compress()
- Password length $<< 16$ → Precompute intermediate results

Our optimizations

Our optimizations

- Memory → Fast shared memory
- Algorithm wise → Precompute intermediate results
- Execution configuration → Block- and gridsizes
- Maximizing parallelization → Password hashing is embarrassingly parallel
- Instructions → Modulo arithmetic is expensive
- Control flow → Branching is expensive

Algorithm optimizations

- Password length < 16 → One call to MD5compress()
- Password length $<< 16$ → Precompute intermediate results

Memory optimizations

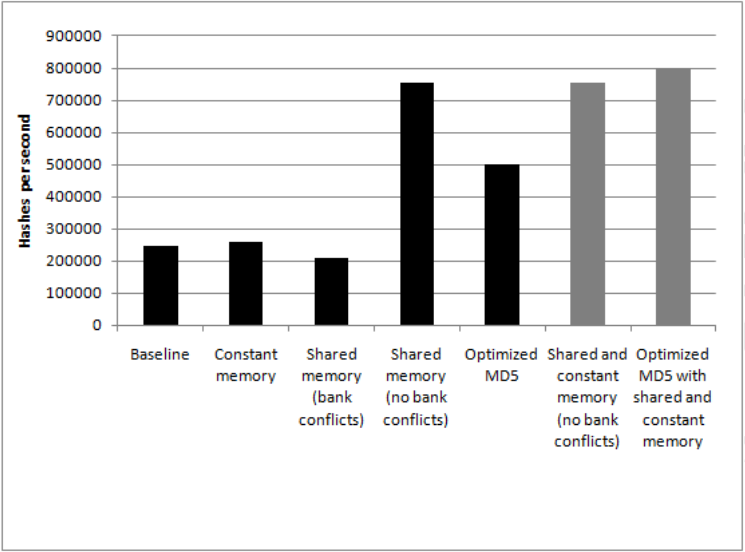
Constant memory

- Default: variables stored in *local memory*
  - Physically resides in global memory (500 clock cycles latency)
- Cached on chip
- As fast as register access (1 clock cycle latency per warp)

Shared memory

- User managed cache
  - On chip (2 clock cycles latency per warp)
  - Shared by all threads in a block
  - Small (16384 Bytes per multiprocessor)
  - Accessed via 16 *banks*

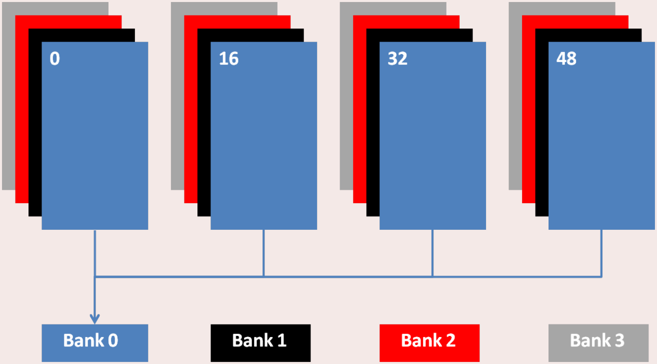
Memory and algorithm optimizations



Bank conflicts

Problem

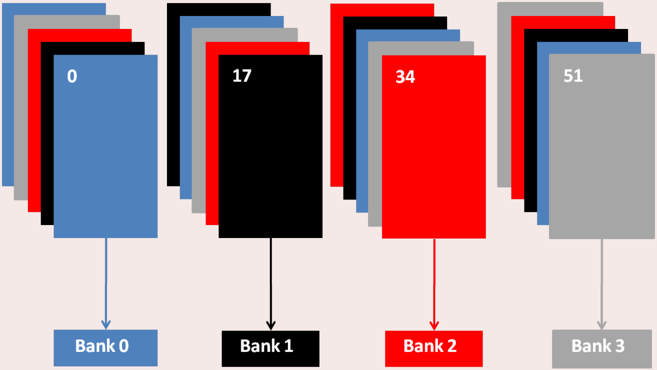
```
int shared[THREADS_PER_BLOCK][16];
int *buffer = shared[threadId];
```



Bank conflicts

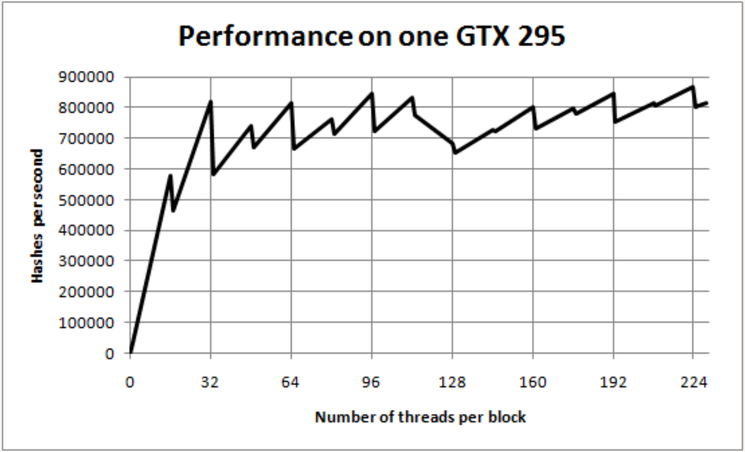
Solution

```
int shared[THREADS_PER_BLOCK][16+1];
int *buffer = shared[threadId]+1;
```

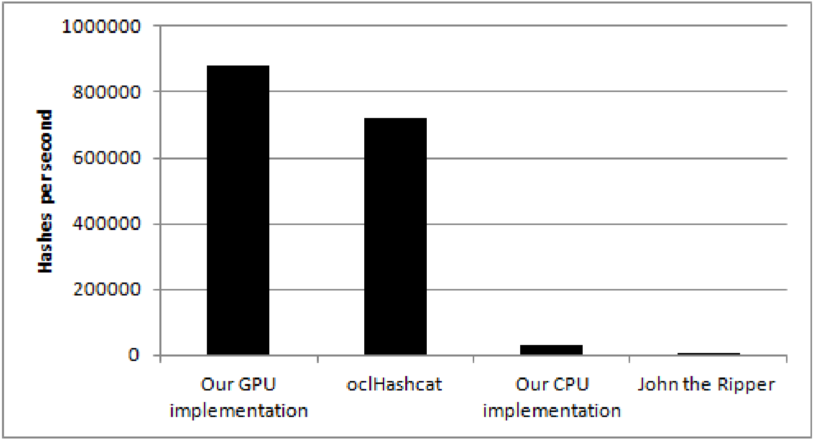


Execution configuration optimizations

Influence on our implementation



Comparison with CPU implementations



Comparison with other implementations

Other implementations

Work	Cryptographic type	Algorithm	Speed up GPU over CPU
Bernstein et al. [2, 1]	Asymmetric	ECC	4-5
Manavski et al. [5]	Symmetric	AES	5-20
Harrison et al. [3]	Symmetric	AES	4-10
Harrisonet al. [4]	Asymmetric	RSA	4
This work	Hashing	MD5-crypt	25-30

Consequences for password safety

Influence on password classes

Length	26 characters	36 characters	62 characters	94 characters
4	0,5 Seconds	2 Seconds	16 Seconds	2 Minutes
5	13 Seconds	1 Minute	17 Minutes	2 Hours
6	5 Minutes	41 Minutes	18 Hours	10 Days
7	2 Hours	1 Days	46 Days	3 Years
8	2 Days	37 Days	8 Years	264 Years
9	71 Days	4 Years	488 Years	20647 Years
10	5 Years	132 Years	30243 Years	2480775 Years

Conclusions

Should we worry?

- Yes, if your password length is < 9 characters
- Increase entropy in passwords $\rightarrow$ password policy
  - Advantage: old schemes still usable
  - Disadvantage: humans and randomness $\rightarrow$ ☹
  - Disadvantage: humans and memorability $\rightarrow$ ☹
  - What is a *good* policy?
- Increase complexity by at least 4 orders of magnitude
  - Advantage: MD5-crypt still usable
  - Disadvantage: passwords not backwards compatible
  - Disadvantage: Moore's law
  - Switch to SHA-crypt or PBKDF2

Conclusions

Should we worry?

- Yes, if your password length is $<$ 9 characters
- Increase entropy in passwords $\rightarrow$ password policy
  - Advantage: old schemes still usable
  - Disadvantage: humans and randomness $\rightarrow$ ☹
  - Disadvantage: humans and memorability $\rightarrow$ ☹
  - What is a *good* policy?
- Increase complexity by at least 4 orders of magnitude
  - Advantage: MD5-crypt still usable
  - Disadvantage: passwords not backwards compatible
  - Disadvantage: Moore's law
  - Switch to SHA-crypt or PBKDF2

Future work

Future work

- Optimizations
  - Additional algorithm optimizations
  - Newer hardware
  - Time-Memory Trade-Off
- Heterogenous crack clusters
  - Consisting of a mix of GPU's, CPU's, mobile devices, etc.
  - Large distributed environments → *Jungle computing* or *Amazon's EC2*
  - OpenCL
- Other schemes and applications
  - SHA-crypt, bcrypt, etc.
  - Frameworks as PBKDF2

Questions and discussion

Thank you for your attention!

- Any questions?
- Contact: Sprengers.Martijn@kpmg.nl

References

D. Bernstein, H. C. Chen, C. M. Cheng, T. Lange, R. Niederhagen, P. Schwabe, and B. Y. Yang.
ECC2K-130 on NVIDIA GPUs.
Progress in Cryptology-INDOCRYPT 2010, pages 328–346, 2010.

D. J. Bernstein, H. C. Chen, M. S. Chen, C. M. Cheng, C. H. Hsiao, T. Lange, Z. C. Lin, and B. Y. Yang.
The billion-mulmod-per-second PC.
SHARCS Workshop, 2009.

O. Harrison and J. Waldron.
Practical symmetric key cryptography on modern graphics hardware.
In *Proceedings of the 17th conference on Security symposium*, pages 195–209. USENIX Association, 2008.

Owen Harrison and John Waldron.
Efficient acceleration of asymmetric cryptography on graphics hardware.
In Bart Preneel, editor, *AFRICACRYPT*, volume 5580 of *Lecture Notes in Computer Science*, pages 350–367. Springer, 2009.

S. A. Manavski.
CUDA compatible GPU as an efficient hardware accelerator for AES cryptography.
In *Signal Processing and Communications, 2007. ICSPC 2007. IEEE International Conference on*, pages 65–68. IEEE, 2008.

Execution configuration optimizations

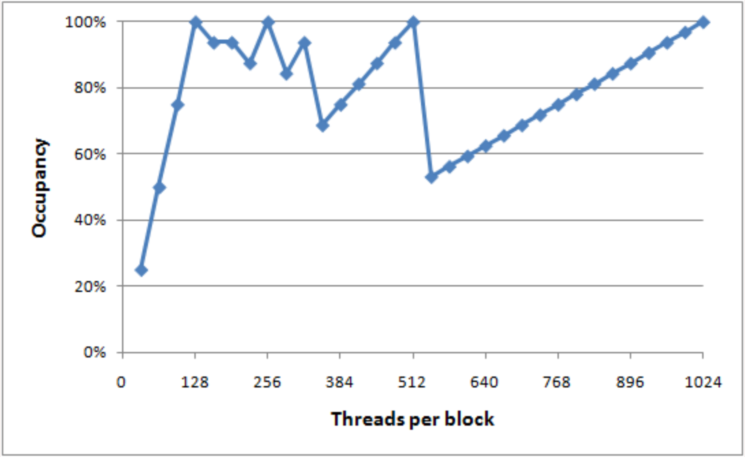
Occupancy

$$\text{Occupancy} = \frac{\text{Active warps per multiprocessor } W_\alpha}{\text{Maximum active warps per multiprocessor } W_{max}}$$

- W_α restricted by *register* and *shared memory* usage
- W_{max} restricted by hardware (32 in our case)
- Programmer can influence W_α by setting the number of threads per block T_b correctly

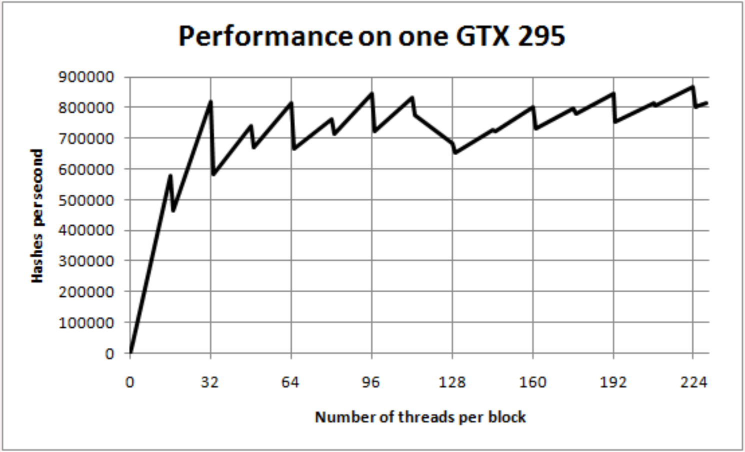
Execution configuration optimizations

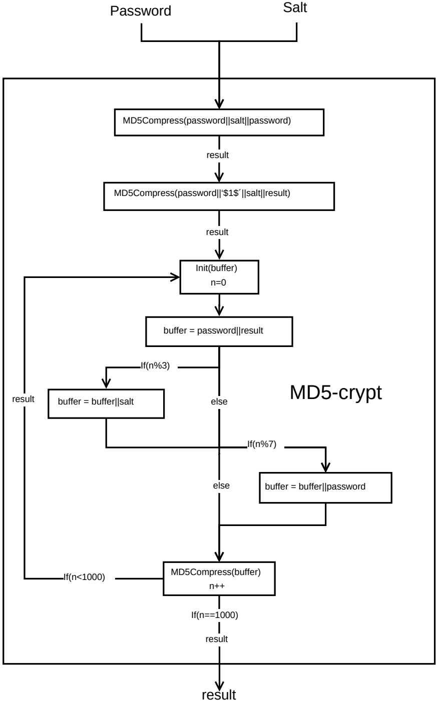
Theoretical calculation



Execution configuration optimizations

Influence on our implementation




Password
Salt
MD5Compress(password||salt||password)
result
MD5Compress(password||'1´||salt||result)
result
Init(buffer)
n=0
buffer = password||result
If(n%3)
buffer = buffer||salt
else
result
MD5-crypt
If(n%7)
buffer = buffer||password
else
MD5Compress(buffer)
n++
If(n<1000)
If(n==1000)
result
result