MD5 CHOSEN-PREFIX COLLISIONS ON GPUS BLACK HAT USA 2009

Marc Bevand
m.bevand@gmail.com
marc_bevand@rapid7.com

July 30, 2009

Abstract

In December 2008, an MD5 chosen-prefix collision attack was performed on a cluster of 215 PlayStation 3 consoles to create a rogue CA certificate. A new implementation of this attack has been researched and developped to run an order of magnitude faster and more efficiently on video card GPUs. This paper gives an overview of the GPGPU technology. It then describes the most computing intensive part of the MD5 chosen-prefix collision attack, known as birthdaying. Finally it demonstrates how a breakthrough performance gain can be achieved by carefully implementing the MD5 birthdaying stage on ATI R700 family GPUs (HD 4000 series).

1 GPGPU Technology

As video card GPUs[1] have evolved to the point of having fully programmable rendering pipelines, the technology has been exposed to the software development world at large under the name of GPGPU[2]. This section explains why GPUs are a particularly good fit to run ALU-bound cryptographic workloads such as MD5.

1.1 Comparison to CPUs

Because graphics workloads are *embarrassingly parallel* workloads, where each pixel on the screen can be rendered independently, a typical modern GPU is based on a massively parallel architecture. As of 2009, a high-end ATI/AMD or Nvidia GPU is capable of executing hundreds of 32-bit instructions per cycle, has a frequency clock in the range of 0.5-1.5 GHz which gives it a theoretical single-precision computing power of about 500-1500 GFLOPS, runs within a TDP[3] of 100-200 W, and has a street price of 150-300 USD. Dual-GPU video cards double the performance, are generally a bit

[1]Graphics Processing Units
[2]General-Purpose computing on Graphics Processing Units
[3]Thermal Design Power

Table 1: Technical specifications of high-end video cards (current as of June 2009)

Video Card / GPU	Number of ALUs	Clock (MHz)	TDP (W)	Perf. (GFLOPS)	Perf. (Ginstr/s)
ATI HD 4870 X2	1600	750	289	2400	1200
ATI HD 4850 X2	1600	625	230	2000	1000
ATI HD 4890	800	850	190	1360	680
ATI HD 4870	800	750	160	1200	600
Nvidia GTX 295	480	1242	289	1788	596
ATI HD 4850	800	625	114	1000	500
ATI HD 4770	640	750	80	960	480
Nvidia GTX 285	240	1476	204	1063	354
Nvidia GTX 275	240	1404	219	1011	337
Nvidia GTX 280	240	1296	236	933	311
Nvidia GTX 260 216SP	216	1242	171	805	268
Nvidia GTX 260 Core 216	216	1242	182	805	268
Nvidia GTX 260	192	1242	182	715	238

more efficient (less than 2x the power consumption), and cost about twice as much. See table 1 for the technical characteristics of a handful of recent high-end GPUs.

By comparison a typical AMD or Intel 4-core CPU is capable of executing 12 128-bit SSE instructions per cycle (3 per core), has a frequency clock in the range of 2-3 GHz which gives it a theoretical single-precision computing power of 100-150 GFLOPS, runs within a power envelope of 50-150 W, and has a street price of 200-1000 USD.

In other words, when looking at the sweet spot of GPUs and CPUs, GPUs offer a performance/USD or performance/Watt ratio that is roughly an order of magnitude better than CPUs if not more. It is no wonder that GPGPU technology is gaining grounds in the HPC[4] industry. In our case, cryptographic workloads such as MD5 are perfect candidates for GPUs.

1.2 GPU Architecture

This section focuses on the latest generation of ATI and Nvidia GPUs: the ATI R700 family (HD 4000 series) and Nvidia GT200 family (GTX 200 series).

A traditional GPU contains hundreds of independent *ALUs*[5]. Each ALU is capable of executing a single 32-bit integer, logical, or floating point instruction. The reader should be aware that ATI and Nvidia use a different terminology which can be confusing; hence the reason this paper uses a more standard nomenclature. ATI calls an ALU a Stream Core and used to call it a Stream Processing Unit; Nvidia calls it a Streaming Processor.

ATI GPUs are based on a 5-way VLIW[6] architecture, where 5 ALUs are put to-

[4]High-Performance Computing
[5]Arithmetic and Logic Units
[6]Very Long Instruction Word

gether to form one *VLIW unit*. ATI calls a VLIW unit a Thread Processor and used to call it a Stream Processor. A single VLIW instruction therefore can execute up to 5 instructions on 5 different ALUs. For the purpose of comparing ATI GPUs to Nvidia GPUs, this VLIW detail, as a logical unit, can be ignored (however it provides significant advantages at the micro-architectural level, see next subsection).

Multiple ALUs are organized into a logical *SIMD*[7] *unit*, which is only capable of dispatching 1 instruction, the same one, to every ALU per cycle. Because each ALU has its own independent register file and is, technically, executing a different thread, all the threads handled by a SIMD unit execute the same code path at any point in time. It is crucial to understand this technical point. It is the reason why GPU designers can pack so many ALUs per GPU ASIC[8], and it is the reason why "branchy" workloads perform poorly on GPUs. To take a simple example and without entering into too many details, a loop which is iterated 100 times by 1 thread and 1 times by all the other threads will in practice cause all the threads to step through 100 iterations of the loop, thereby wasting computing resources. An ATI R700 GPU organizes 80 ALUs (16 VLIW units) into a single SIMD unit (called a SIMD Core or SIMD Engine); an Nvidia GT200 organizes 8 ALUs into a single SIMD unit (called a Streaming or Shading Multiprocessor).

Finally, multiple SIMD units coexist on a single GPU ASIC. An ATI R700 GPU has 10 SIMD units, or 160 VLIW units, or 800 ALUs; an Nvidia GT200 GPU has 30 SIMD units, or 240 ALUs.

1.3 ATI vs. Nvidia

It is important to note that although GFLOPS ratings are often used to compare GPUs between each other, they are particular misleading numbers when comparing ATI GPUs to Nvidia GPUs. This is because the GFLOPS rating of an ATI GPU is computed assuming each ALU executes 1 multiply-add instruction per cycle (MAD = 2 floating point operation per cycle), whereas for an Nvidia GPU it is computed assuming each ALU executes 1 fused pair of multiple & multiply-add instructions per cycle (MUL+MAD = 3 floating point operations per cycle).

A better performance rating number to predict the relative performance of different GPUs on ALU-bound cryptographic workloads is the number of billion instructions per second as seen in table 1, column Performance (Ginstr/s).

By that number, it is obvious that ATI GPUs offer a significantly superior level of performance/Watt, performance/USD, and absolute performance, than Nvidia GPUs. These 3 metrics are the reason why the rest of the paper focuses on ATI GPUs, despite the fact that the ATI GPGPU SDK (Stream SDK) is generally perceived as less mature than the Nvidia GPGPU SDK (CUDA). The VLIW architecture chosen by ATI seems to have paid off as it allows them to provide more computing power per square inch of die area.

It remains to be seen how the next-generation GPU families will compare to each other. The author of that paper is keeping a close and interested eye on ATI R800, Nvidia GT300, and Intel Larrabee.

[7]Single Instruction, Multiple Data
[8]Application-Specific Integrated Circuit

2 Chosen-Prefix Collision Overview

This section gives a quick overview of the MD5 algorithm, how the chosen-prefix collision attack works, and how it was applied to produce a rogue CA.

2.1 MD5 Algorithm

The MD5 algorithm, described in RFC 1321, is presented in this section in very succinct (and incomplete) details, mostly to establish the notation used in the rest of the paper.

1. The message is padded by appending a '1' bit. Then a number of '0' bits is appended to make the bitlength equal to $448 mod 512$. Finally the bitlength of the original unpadded message is appended as a 64-bit little-endian integer.
2. The padded message is partitionned into N consecutive 512-bit blocks $M_1, M_2, ..., M_N$.
3. To hash a message consisting of N blocks, MD5 goes through $N + 1$ states IHV_i, for $0 \leq i \leq N$, called the *intermediate hash values*. Each intermediate hash value IHV_i consists of 4 32-bit words a_i, b_i, c_i, d_i. They are fixed to predefined values for $i = 0$, and for $i = 1, 2, ..., N$, IHV_i is computed using the MD5 compression function:
   $IHV_i = MD5Compress(IHV_{i-1}, M_i)$
4. The final hash value is the last IHV_N.

The MD5 compression function partitions the message block M_i in 16 32-bit words $m_0, m_1, ..., m_15$ and executes 64 steps. Each step t is of the following form:
$w = x + ((w + func(x, y, z) + m_k + AC_t) <<< RC_t)$
Where: x, y, z, w is a permutation of 4 working registers, $func()$ is a non-linear function involving logical operators (defined as one of the 4 $F(), G(), H(), I()$), AC_t is an addition constant, and RC_t is a rotation constant.

2.2 Rogue CA

In December 2008, an MD5 chosen-prefix collision attack was performed by Sotirov, Stevens, Applebaum, Lenstra, Molnar, Osvik, and Weger to create a rogue CA certificate using a cluster of 215 PlayStation 3 consoles. After having presented their results at the 25th Chaos Communication Congress (25C3), they released a paper containing more details in March 2009 [2], followed by another paper containing the complete details plus new improvements in June 2009 [3].

The attack performed by the 7 researchers involved 3 steps:

1. They pre-computed a legitimate certificate "A" and a rogue CA[9] certificate "B", in such a way that the to-be-signed parts of the 2 certificates collided under MD5, so that a signature for a certificate could also be applied to the other.

[9]Certification Authority

2. Then they submitted a specially crafted certificate signing request (CSR) to a CA that ended up with the CA generating a certificate that was identical to certificate ”A”. This step involved submitting the request at a very specific time in order to match the validity period and the serial number that had to be predicted in the first step.

3. The valid signature from certificate ”A” was then simply copied to the rogue CA ”B”, which is possible because the to-be-signed parts collide.

2.3 Chosen-Prefix Collision Details

A chosen-prefix collision attack is an attack where 2 message prefixes P and P' can be chosen freely, and collision blocks C and C' can be constructed such that the MD5 hash of the concatenated messages $P||C$ and $P'||C'$ collide. Because of the nature of MD5, any suffix S can be appended to these messages so that the MD5 hash of $P||C||S$ and $P'||C'||S$ also collide. The attack was used to generate the legitimate certificate ”A” and the rogue CA certificate ”B” in the following way:

- The legitimate certificate ”A” was partitionned into 3 parts so that the arbitrary prefix P was chosen to map to most of the certificate fields (version, serial number issuer, validity period, subject, etc). The collision blocks C were chosen to map to part of the public key field. The suffix S was chosen to map to the end of the public key field, followed by the X.509 extensions, etc.
- A rogue CA certificate ”B” was built as the other arbitrary prefix P' in a way to start a ”Netscape comment” field that would cover the collision blocks C' and S.

In order to compute the collision blocks C and C', pre-conditions on the prefixes P and P' have to be met. In order to meet these, the last bits of the prefixes are actually bruteforced in a stage known as a birthday search, or *birthdaying*. This is the most computing intensive part of the attack, which was originally estimated by the researchers at $2^{51.x}$ MD5 compression function calls, or 18 hours on 215 PlayStation 3 consoles, according to [1].

Once this birthdaying stage is completed, the collision blocks are computed using an automated way to find differential paths for MD5 (near collision stage). Then the final stage consists of choosing a suffix S such that the public key of the legitimate certificate ”A” is a valid RSA modulus. These final steps are significantly less computing intensive and are not covered in this paper.

3 Birthdaying on ATI R700

This section describes the goal of the birthdaying stage and how it was optimized for the ATI R700 GPU.

3.1 Algorithm

The last few bits (”birthday bits”) of the last message block of the prefixes P and P' have to be bruteforced in order for the difference between IHV_n and IHV'_n to satisfy some conditions, namely: $\delta IHV_n = (0, \delta b, \delta c, \delta c)$ for some δb and some δc. In other words we are looking for a collision $(a, c - d) = (a', c' - d')$ between $IHV_n = (a, b, c, d)$ and $IHV'_n = (a', b', c', d')$.

The algorithm used to find such a collision is a deterministic pseudo-random walk in the birthday bits search space using the Pollard-Rho method. The concept behind that algorithm is similar to the problem of colliding chains in rainbow tables, except that in this case we are looking for collisions. Many variation of the algorithms are valid but the one I chose to implement is the following.

The inputs of the algorithm are: the last partial message blocks B and B' of the prefixes P and P', and the current IHV_n and IHV'_n states. The steps are as follow:

1. Initially, random birthday bits x are chosen.
2. If the less significant birthday bit is 0: call $MD5Compress(IHV_n, B\|x)$, compute $(a, c - d)$, save some of these bits as the current birthday bits,
   else: call $MD5Compress(IHV'_n, B'\|x)$, compute $(a', c' - d')$, save some of these bits as the current birthday bits.
3. If the current birthday bits have reached a distinguished point (say the less significant 32 bits are zero), return. The initial birthday bits (”startpoint”) and current birthday bits (”endpoint”) will be saved in a hashtable whose keys are the endpoints.
4. Else, go to step 2.

Eventually, the hashtable will contain endpoint collisions. Because the startpoint are saved as well, the ”colliding paths” can be replayed to determine where they collide. 50% of the collisions will be useless (collision between the same blocks), and the remaining 50% will be useful for performing the following stage of the chosen-prefix collision attack.

3.2 CAL IL Implementation

CAL IL stands for Compute Abstract Layer Intermediary Language and is one of the lowest-level language available to program ATI GPUs. It conceptually sits right above the native R700 ISA, and below Brook+. It has been chosen for its maturity (compared to Brook+), portability (across R600, R700, and R800 in the future), and excellent ability to exploit the computing power of the GPU close to its theoretical maximum.

Obviously, the MD5 compression function calls in the birthdaying stage are the most time-intensive. Therefore an MD5 compression function has been hand-coded in CAL IL.

Here is a step involving the F() function for example:

Table 2: MD5Compress() benchmark

	PlayStation 3 (Cell BE 3.2 GHz)	ATI HD 4850 X2
Street price	400 USD	200 USD
TDP	130 W (45nm Cell)	230 W
MD5Compress/s	180 Mcompr/s	1634 Mcompr/s
Performance/USD	0.45 Mcompr/s/USD	8.17 Mcompr/s/USD
Performance/Watt	1.38 Mcompr/s/W	7.10 Mcompr/s/W

```
; F(x,y,z)
ixor r5, r0, r1
and  r5, r5, r3
ixor r5, r5, r1
; m_k + AC_t
iadd r6, cb0[0].zzzz, cb1[0].zzzz
; sum everything
iadd r5, r2, r5
iadd r5, r5, r6
; rotate <<< RC_T
ushr r6, r5, l1.xxxx
umad r5, r5, l1.yyyy, r6
; final add
iadd r2, r3, r5
```

One of the optimization technique used here is to perform the rotate operation by using a pair of unsigned shift right (ushr) + unsigned multiply-add (umad).

The addition constants AC_t are stored in constant buffers to reduce the number of literal registers that are used.

4 Performance

After table 1, the author selected the ATI HD 4850 X2 as the optimal GPU to run the birthdaying stage. Available for about 200 USD on NewEgg as of June 2009, this card is one with the most promising performance/USD ratio to run this MD5 workload.

For reference purposes, the PlayStation 3 Cell BE processor at 3.2 GHz is able to execute about 180 million MD5Compress() call per second, or about 30 Mcompr/s per SPU[10] according to number inferred from [1].

According to table 2, the ATI HD 4850 X2 video card **costs half the price of a PS3, is 9 times faster, provides a performance/USD ratio 18 times better, and a performance/W ratio 5 times better.**

[10]Synergistic Processing Unit

5 Conclusion

GPUs are massively parallel chips that have become a commodity thanks to the gaming industry. It is clear that many ALU-bound cryptographic workloads are perfect targets for GPGPU technology. Considering any sort of general purpose CPU almost doesn't make sense anymore.

An MD5 birthdaying tool has been written to exploit this new level of performance, it will be open-sourced and released at `http://www.epitech.net/~bevand_m` . This makes MD5 chosen-prefix collision attacks practical for anybody. Public CAs have stopped signing certificates with MD5, but what about private/corporate CAs?

It is clear by now that systems should be migrated to SHA-2 or SHA-1, however the latter is already starting to show its weaknesses[4]. The NIST hash function competition for a new SHA-3 function is more than welcome.

References

[1] MD5 considered harmful today, Creating a rogue CA certificate
`http://www.win.tue.nl/hashclash/rogue-ca/`

[2] Short Chosen-Prefix Collisions for MD5 and the Creation of a Rogue CA Certificate
`http://eprint.iacr.org/2009/111`

[3] Chosen-prefix Collisions for MD5 and Applications
`https://documents.epfl.ch/users/l/le/lenstra/public/papers/lat.pdf`

[4] SHA-1 collisions now 2^{52}
`http://eurocrypt2009rump.cr.yp.to/837a0a8086fa6ca714249409ddfae43d.pdf`