security-assessment.com Ltd
Level 1 / 5 Nelson Street
PO Box 106-402
Auckland 1001 / New Zealand
Tel: +64 9 302 5093
Fax: +64 9 302 5023
Web: www.security-assessment.com

White Paper

# Title: Overview Of SPU-Optimised Cryptography / "CrackStation"

Prepared by: Nick Breese
Senior Security Consultant
Security-Assessment.com

Date: 25th February 2008

## Abstract

An implementation of MD5 using vector computing, rather than typical scalar computing, has yielded incredible gains in calculation time. I have been working on a project currently dubbed "Crackstation" for the past 6 months to implement common ciphers and hash functions using vector computing. The current focus has been on the Cell Broadband Engine ("CBE" / “Cell” ) within the PlayStation 3. Quick tests have confirmed that vector implementations of MD5 using x86 SSE also increases calculation time. Other vector processing technologies may also be applicable, but are currently untested.

The focus of this whitepaper is to show how easy it is to start porting an existing C-based algorithm implementation to a vector equivalent. This is intended to supplement the BlackHat presentation slides and source code. All information will be available at http://www.security-assessment.com and mirrored at http://insecure.io.

## Technology Overview

The SPU processors within the PlayStation 3's CBE are effectively vector processors with enhanced features. Per-core local storage memory, a high number of general purpose registers and their self-contained nature make the SPU processors excellent choices for specific vector-based cryptography implementations.

Benchmarks have been conducted to assess the potential for vector calculation within password cracking. While the results are strictly only to be used as a guideline, they are encouraging. The PlayStation 3 manages to conduct over 1.4 billion MD5 calculations a second.

## Introduction To SIMD

Vector processing is known commonly as Single-Instruction, Multiple Data (SIMD). Unlike scalar operations which conduct single mathematical calculations to a single piece of data; SIMD allows a single mathematical operation to be applied to a data group. This method is not intended to conduct each operation faster. The concept is to apply the same operation to multiple pieces of data, which yields in a greater number of results.

**Single instruction, single data (scalar / "traditional")**

```
1 + 4 = 4
```

1 ← data, + ← instruction

**Single instruction, multiple data (vector)**

```
{1, 2, 3, 4} + 4 = {5, 6, 7, 8}
```

1, 2, 3, 4 ← data set, + ← single instruction

**Single instruction, multiple data (vector)**

```
{1, 2, 3, 4} + {5, 6, 7, 8} = {6, 8, 10, 12}
```

1, 2, 3, 4 ← data set, + ← single instruction, 5, 6, 7, 8 ← second data set

**Single instruction, single data (scalar / "traditional")**

```
  int a, b, c;

  a = 1;

  b = 2;

  c = a + b;
```

**Single instruction, multiple data (vector)**

```
  vec_int4 vec_a, vec_b, vec_c;

  vec_a = (vec_int4) {1,2,3,4};

  vec_b = (vec_int4) {5,6,7,8};

  vec_c = spu_add(vec_a,vec_b);

// vec_c is now {6, 8, 10, 12}
```

The above demonstrates the simple concept and implementation of using SIMD. In the latter example, the vector of individual values is merely replaced with the values of the multiple MD5 calculations we are performing at one time.

## Optimisation Overview

While there may be exceptions to this rule, it is likely that conducting multiple calculations in parallel would be the most practical. This is especially true in the case of MD5 as an avalanche effect occurs. This means that each intermediary value is dependant on the previous value. Running different parts of the same md5 calculation in parallel is not possible.

At its core, the MD5 hash function operates on 32bit unsigned integers. Both SSE2 and the Cell SPU use a 128bit general purpose register file for vector data groups. This in turn allows for four 32bit concurrent MD5 computation values to co-exist at the same time. With my implementation, each mathematical calculation of MD5 is applied against the vector data group, resulting in four separate MD5 calculation streams to be conducted concurrently. Using Linux under the PlayStation 3 hypervisor allows access to six SPUs in total, giving 24 concurrent MD5 calculations.

If your algorithm uses values greater than 32bits in places (i.e. DES), attempt to figure out a way of breaking them down to smaller chunks. Remember that people have been using 32bit processors for years, so there is likely an elegant solution in some freely-available implementation.

## Optimisation Details

Vector functions are available to a C coder via instrinsics. The vast amount of time spent writing an SPU accelerated algorithm is spent on simply mapping each operation (ie. add, subtract, xor, etc.) to an equivalent vector operation using the appropriate intrinsic. These can be found within IBM's C/C++ Language Extensions for Cell Broadband Engine Architecture[1] document.

Below are code snippets from equivalent functions found in L. Peter Deutsch's md5.c [2] and spu_md5.c.

The repeatedly used "rotate left" function. Negative value of n is required to correctly conduct the shift using spu_rlmask.

**md5.c**

```
#define ROTATE_LEFT(x, n) (((x) << (n)) | ((x) >> (32 - (n))))
```

**spu_md5.c**

```
vec_uint4 rotate_left (vec_uint4 * vec_x, unsigned int n) {
  int rshift = 32-n;
  return spu_or(spu_sl(*vec_x,n),spu_rlmask(*vec_x,-
(rshift)));
}
```

[1] http://www-01.ibm.com/chips/techlib/techlib.nsf/products/Cell_Broadband_Engine

[2] http://mirror.cs.wisc.edu/pub/mirrors/ghost/packages/md5.tar.gz

The “F” function used by the first round of MD5 calculations:

**md5.c**

```
#define F(x, y, z) (((x) & (y)) | (~(x) & (z)))
```

**spu_md5.c**

```
vec_uint4 f_round_1(vec_uint4 * vec_x, vec_uint4 * vec_y,
vec_uint4 * vec_z)
{
  vec_uint4 vec_f;
  vec_uint4 vec_f1;
  vec_uint4 vec_f2;
  vec_f1 = spu_and(*vec_x,*vec_y);
  vec_uint4 vec_comp_x = spu_nor (*vec_x,vec_null);
  vec_f2 = spu_and ( vec_comp_x, *vec_z );
  vec_f = spu_or ( vec_f1, vec_f2 );
  return(vec_f);
}

//vec_null == vector of null
```

The first round of MD5 calculations:

**md5.c**

```
#define SET(a, b, c, d, k, s, Ti)\
  t = a + F(b,c,d) + X[k] + Ti;\
  a = ROTATE_LEFT(t, s) + b
```

**spu_md5.c**

```
static void set_round_1 (vec_uint4 * vec_a, vec_uint4 * vec_b,
vec_uint4 * vec_c, vec_uint4 * vec_d, vec_uint4 * vec_k,
unsigned int s, vec_uint4 * Ti)
{
  vec_uint4 vec_t;
  vec_t = f_round_1(vec_b, vec_c, vec_d);
  vec_t = spu_add(vec_t,*vec_a);
  vec_t = spu_add(vec_t,*vec_k);
  vec_t = spu_add(vec_t,*Ti);
  vec_t = rotate_left(&vec_t,s);
  vec_t = spu_add(vec_t,*vec_b);
  *vec_a = vec_t;
}
```

There is very little expertise needed. Problems can arise when there is not a direct equivalent intrinsic available, in which case you would have to come up with an alternative method. One example of such would be the lack of an “spu_not” intrinsic for the bitwise NOT operation. Performing an spu_eqv or spu_nor operation against your designated value and a vector of null would accomplish the same task.

## Final Summary

Pure vector-based cryptographic implementations can aide certain attacks. The most obvious would be password cracking as these operations can be run in parallel quite well. While each individual cryptographic algorithm implementation needs to be rewritten to take advantage of SIMD optimisation, this document demonstrates how easy it is.

Calculation time to brute-force a particular cryptographic function is a commonly-used metric. This time is calculated using the current high-end x86 processors. Due to their ever-increasing performance and ubiquity, using x86-based processors has been reasonable. These results suggest that x86 should possibly not be used as a baseline in the future.

## References

L. Peter Deutsch’s implementation of md5
http://mirror.cs.wisc.edu/pub/mirrors/ghost/packages/md5.tar.gz

IBM Cell Broadband Engine documentation
http://www-01.ibm.com/chips/techlib/techlib.nsf/products/Cell_Broadband_Engine

Security-Assessment.com
http://www.security-assessment.com

## About Security-Assessment.com

Security-Assessment.com is Australasia's leading team of Information Security consultants specialising in providing high quality Information Security services to clients throughout the Asia Pacific region. Our clients include some of the largest globally recognised companies in areas such as finance, telecommunications, broadcasting, legal and government. Our aim is to provide the very best independent advice and a high level of technical expertise while creating long and lasting professional relationships with our clients.

Security-Assessment.com is committed to security research and development, and its team continues to identify and responsibly publish vulnerabilities in public and private software vendor's products. Members of the Security-Assessment.com R&D team are globally recognised through their release of whitepapers and presentations related to new security research.

Security-Assessment.com is an Endorsed Commonwealth Government of Australia supplier and sits on the Australian Government Attorney-General's Department Critical Infrastructure Project panel. We are certified by both Visa and MasterCard under their Payment Card Industry Data Security Standard Programs.

## Copyright Information