CRYPTONOTE STANDARD 008
Category: Main Track

Seigen
Max Jameson
Tuomo Nieminen
Neocortex
Antonio M. Juarez
CryptoNote
March 2013

CryptoNight Hash Function

Abstract

This document is part of the CryptoNote Standards describing a peer-to-peer anonymous payment system. It defines the CryptoNote's default proof-of-work hash function, CryptoNight.

Copyright and License Notice

Table of Contents

1. Introduction

CryptoNight is a memory-hard hash function. It is designed to be inefficiently computable on GPU, FPGA and ASIC architectures. The CryptoNight algorithm's first step is initializing large scratchpad with pseudo-random data. The next step is numerous read/write operations at pseudo-random addresses contained in the scratchpad. The final step is hashing the entire scratchpad to produce the resulting value.

2. Definitions

hash function: an efficiently computable function which maps data of arbitrary size to data of fixed size and behaves similarly to a random function

scratchpad: a large area of memory used to store intermediate values during the evaluation of a memory-hard function

3. Scratchpad Initialization

First, the input is hashed using Keccak [KECCAK] with parameters b = 1600 and c = 512. The bytes 0..31 of the Keccak final state are interpreted as an AES-256 key [AES] and expanded to 10 round keys. A scratchpad of 2097152 bytes (2 MiB) is allocated. The bytes 64..191 are extracted from the Keccak final state and split into 8 blocks of 16 bytes each. Each block is encrypted using the following procedure:

```
for i = 0..9 do:
    block = aes_round(block, round_keys[i])
```

Where aes_round function performs a round of AES encryption, which means that SubBytes, ShiftRows and MixColumns steps are performed on the block, and the result is XORed with the round key. Note that unlike in the AES encryption algorithm, the first and the last rounds are not special. The resulting blocks are written into the first 128 bytes of the scratchpad. Then, these blocks are encrypted again in the same way, and the result is written into the second 128 bytes of the scratchpad. Each time 128 bytes are written, they represent the result of the encryption of the previously written 128 bytes. The process is repeated until the scratchpad is fully initialized.

This diagram illustrates scratchpad initialization:

```
                               +-----+
                               |Input|
                               +-----+
                                  |
                                  V
                             +--------+
                             | Keccak |
                             +--------+
                                  |
                                  V
   +-------------------------------------------------------------+
   |                         Final state                         |
   +-------------+--------------+---------------+----------------+
   | Bytes 0..31 | Bytes 32..63 | Bytes 64..191 | Bytes 192..199 |
   +-------------+--------------+---------------+----------------+
          |                             |
          V                             |
   +-------------+                      V
   | Round key 0 |------------+---+->+-----+
   +-------------+            |   |  |     |
   |      .      |            |   |  |     |
   |      .      |            |   |  | AES |
   |      .      |            |   |  |     |
   +-------------+            |   |  |     |
   | Round key 9 |----------+-|-+-|->+-----+                 +---+
   +-------------+          | | | |     |                    |   |
                            | | | |     +------------------->|   |
                            | | | |     |                    |   |
                            | | | |     V                    |   |
                            | | | +->+-----+                 |   |
                            | | |    |     |                 | S |
                            | | |    |     |                 |   |
                            | | |    | AES |                 | c |
                            | | |    |     |                 |   |
                            | | |    |     |                 | r |
                            | | +--->+-----+                 |   |
                            | |         |                    | a |
                            | |         +------------------->|   |
                            | |         .                    | t |
                            | |         .                    |   |
                            | |         .                    | c |
                            | |         +------------------->|   |
                            | |         |                    | h |
                            | |         V                    |   |
                            | +----->+-----+                 | p |
                            |        |     |                 |   |
                            |        |     |                 | a |
                            |        | AES |                 |   |
```

```
                            |        |     |                 | d |
                            |        |     |                 |   |
                            +------->+-----+                 |   |
                                        |                    |   |
                                        +------------------->|   |
                                                             |   |
                                                             +---+
```

Figure 3: Scratchpad initialization diagram

4. Memory-Hard Loop

Prior to the main loop, bytes 0..31 and 32..63 of the Keccak state are XORed, and the resulting 32 bytes are used to initialize variables a and b, 16 bytes each. These variables are used in the main loop. The main loop is iterated 524,288 times. When a 16-byte value needs to be converted into an address in the scratchpad, it is interpreted as a little-endian integer, and the 21 low-order bits are used as a byte index. However, the 4 low-order bits of the index are cleared to ensure the 16-byte alignment. The data is read from and written to the scratchpad in 16-byte blocks. Each iteration can be expressed with the following pseudo-code:

```
scratchpad_address = to_scratchpad_address(a)
scratchpad[scratchpad_address] = aes_round(scratchpad
  [scratchpad_address], a)
b, scratchpad[scratchpad_address] = scratchpad[scratchpad_address],
  b xor scratchpad[scratchpad_address]
scratchpad_address = to_scratchpad_address(b)
a = 8byte_add(a, 8byte_mul(b, scratchpad[scratchpad_address]))
a, scratchpad[scratchpad_address] = a xor
  scratchpad[scratchpad_address], a
```

Where, the 8byte_add function represents each of the arguments as a pair of 64-bit little-endian values and adds them together, component-wise, modulo 2^64. The result is converted back into 16 bytes.

The 8byte_mul function, however, uses only the first 8 bytes of each argument, which are interpreted as unsigned 64-bit little-endian integers and multiplied together. The result is converted into 16 bytes, and finally the two 8-byte halves of the result are swapped.

This diagram illustrates the memory-hard loop:

```
+-------------------------------------------------------------+
|                         Final state                         |
+-------------+--------------+---------------+----------------+
| Bytes 0..31 | Bytes 32..63 | Bytes 64..191 | Bytes 192..199 |
+-------------+--------------+---------------+----------------+
       |             |
       |   +-----+   |
       +-->| XOR |<--+
           +-----+
            |   |
       +----+   +----+
       |             |
       V             V
     +---+         +---+
     | a |         | b |
     +---+         +---+
       |             |
--------------------- REPEAT 524288 TIMES ---------------------
       |             |                            address +---+
       +-------------|----------------------------------->|   |
       |   +-----+   |                               read |   |
       +-->| AES |<--|------------------------------------|   |
       |   +-----+   V                                    |   |
       |      |   +-----+                                 | S |
       |      +-->| XOR |                                 |   |
       |      |   +-----+                           write | c |
       |      |      |    +------------------------------>|   |
       |      |      +----+                       address | r |
       |      +------------------------------------------>|   |
       |      |  +-----------+                       read | a |
       |      +->| 8byte_mul |<--+------------------------|   |
       |      |  +-----------+   |                        | t |
       |      |        |         |                        |   |
       |      |        V         |                        | c |
       |      |  +-----------+   |                        |   |
       +------|->| 8byte_add |   |                        | h |
              |  +-----------+   |                        |   |
              |        |         |                  write | p |
              |        +---------|----------------------->|   |
              |        |         |                        | a |
              |        V         |                        |   |
              |     +-----+      |                        | d |
              |     | XOR |<-----+                        |   |
              |     +-----+                               |   |
              +------+ |                                  |   |
       +-------------|-+                                  |   |
       |             |                                    +---+
-------------------------- END REPEAT -------------------------
```

```
       |             |
```

Figure 4: Memory-hard loop diagram

5. Result Calculation

After the memory-hard part, bytes 32..63 from the Keccak state are expanded into 10 AES round keys in the same manner as in the first part.

Bytes 64..191 are extracted from the Keccak state and XORed with the first 128 bytes of the scratchpad. Then the result is encrypted in the same manner as in the first part, but using the new keys. The result is XORed with the second 128 bytes from the scratchpad, encrypted again, and so on.

After XORing with the last 128 bytes of the scratchpad, the result is encrypted the last time, and then the bytes 64..191 in the Keccak state are replaced with the result. Then, the Keccak state is passed through Keccak-f (the Keccak permutation) with b = 1600.

Then, the 2 low-order bits of the first byte of the state are used to select a hash function: 0=BLAKE-256 [BLAKE], 1=Groestl-256 [GROESTL], 2=JH-256 [JH], and 3=Skein-256 [SKEIN]. The chosen hash function is then applied to the Keccak state, and the resulting hash is the output of CryptoNight.

The diagram below illustrates the result calculation:

```
+--------------------------------------------------------------+
|                          Final state                         |
+-------------+--------------+---------------+-----------------+
| Bytes 0..31 | Bytes 32..63 | Bytes 64..191 | Bytes 192..199  |
+-------------+--------------+---------------+-----------------+
      |                |             |                |
      |       +--------+             |                |
      |       V        |             |                |
      |+-------------+ |             |                |
      || Round key 0 |-|---+---+     |                |
      |+-------------+ |   |   |     |                |
      ||      .      | |   |   |     |                |
      ||      .      | |   |   |     |                |
      ||      .      | |   |   |     |                |
      |+-------------+ |   |   |     |                |
+---+ || Round key 9 |-|-+-|-+ |     V                |
|   | |+-------------+ | | | | |  +-----+             |
|   |-|----------------|-|-|-|-|->| XOR |             |
|   | |                | | | | |  +-----+             |
| S | |                | | | | |     |                |
|   | |                | | | | |     V                |
| c | |                | | | | +->+-----+             |
|   | |                | | | |    |     |             |
| r | |                | | | |    |     |             |
|   | |                | | | |    | AES |             |
| a | |                | | | |    |     |             |
|   | |                | | | |    |     |             |
| t | |                | | | +--->+-----+             |
|   | |                | | |        |                |
| c | |                | | |        V                |
|   | |                | | |     +-----+             |
| h |-|----------------|-|-|----->| XOR |             |
|   | |                | | |     +-----+             |
| p | |                | | |        |                |
|   | |                | | |        .                |
| a | |                | | |        .                |
|   | |                | | |        .                |
| d | |                | | |        |                |
|   | |                | | |        V                |
|   | |                | | |     +-----+             |
|   |-|----------------|-|-|----->| XOR |             |
|   | |                | | |     +-----+             |
+---+ |                | | |        |                |
      |                | | |        V                |
      |                | | +----->+-----+             |
      |                | |        |     |             |
      |                | |        |     |             |
      |                | |        | AES |             |
```

```
         |                | |        |     |             |
         |                | |        |     |             |
         |                | +------->+-----+             |
         |                |             |                |
         V                V             V                V
   +-------------+--------------+---------------+----------------+
   | Bytes 0..31 | Bytes 32..63 | Bytes 64..191 | Bytes 192..199 |
   +-------------+--------------+---------------+----------------+
   |                       Modified state                        |
   +-------------------------------------------------------------+
                                  |
                                  V
                            +----------+
                            | Keccak-f |
                            +----------+
                             |    |
                 +-----------+    |
                 |                |
                 V                 V
          +-------------+  +-------------+
          | Select hash |->| Chosen hash |
          +-------------+  +-------------+
                                  |
                                  V
                          +--------------+
                          | Final result |
                          +--------------+
```

Figure 5: Result calculation diagram

Hash examples:

Empty string:
eb14e8a833fac6fe9a43b57b336789c46ffe93f2868452240720607b14387e11.

"This is a test":
a084f01d1437a09c6985401b60d43554ae105802c5f5d8a9b3253649c0be6605.

6. References

[AES] "Announcing the ADVANCED ENCRYPTION STANDARD", FIPS 197, 2001.

[BLAKE] Aumasson, J.-P., Henzen, L., Meier, W., and R. C.-W. Phan, "SHA-3 proposal BLAKE", 2010.

[GROESTL] Gauravaram, P., Knudsen, L., Matusiewicz, K., Mendel, F., Rechberger, C., Schlaffer, M., and S. Thomsen, "Groestl - a SHA-3 candidate", 2011.

[JH] Wu, H., "The Hash Function JH", 2011.

[KECCAK] Bertoni, G., Daemen, J., Peeters, M., and G. Van Assche, "The Keccak reference", 2011.

[SKEIN] Ferguson, N., Lucks, S., Schneier, B., Whiting, D., Bellare, M., Kohno, T., Callas, J., and J. Walker, "The Skein Hash Function Family", 2008.