SHA-3: The BLAKE Hash Function

Ryan Toukatly
Rochester Institute of Technology

Abstract—A global competition is currently taking place to select a hash function which will become a new standard in the field of cryptography. The competition was announced in 2007 by the National Institute of Standards and Technology (NIST) to find a successor to its previous Secure Hash Algorithm (SHA) standards, SHA-1 and SHA-2. The submissions have been narrowed down to a set of five, judged by their cryptographic security, speed, design features, and practical considerations. One of these five finalists is the BLAKE hash function, developed by a team based in Switzerland. This document will describe the function in detail, assess its qualities and strengths, and give insight into its implementation and performance. In 2012, the BLAKE algorithm may be selected by NIST to be published as the new SHA-3 standard.

I. INTRODUCTION

Before the BLAKE algorithm is discussed in detail, this section describes hash functions in general, as well as their cryptographic applications.

A. Hash Overview

By definition, hash functions are transformations which produce a numeric “hash” or “digest” of a predefined length from an input message of arbitrary length (both typically measured in bits). These are often used in digital systems for checking the integrity of transmitted data, quickly looking up messages in a database, or mapping messages to table indexes. In the field of cryptography, hashes have further applications, including the creation of authentication codes, digitally signing documents, and generation of statistically random data streams.

A “good” (strong) hashing algorithm can generate a hash from data quickly, yet is computationally infeasible for anyone to determine the original input from the hash itself. It should also be infeasible to find two or more messages that result in the same hash, called a collision. Finally, any change in the input message (regardless of significance) should produce a drastic and seemingly-random change in the output hash. The algorithm is considered weak if it does not meet these requirements. (For example, the Message Digest 5 [MD5] algorithm has been “broken”, as researchers found methods to generate different messages which map to identical hashes.) These qualities distinguish cryptographic hash functions from general purpose hash functions, at the cost of complexity, computation time, and output size. For the purposes of this document, any mention of hash functions refers to cryptographic hash functions specifically.

B. The SHA-3 Competition

The National Institute of Standards and Technology (NIST) has established a set of standardized hash functions called the Secure Hash Algorithm (SHA). The first version, called SHA-0, was published in 1993. Its design was influenced by cryptographer Ronald L. Rivest and his work on the MD4 and MD5 hashes. Since its 1995 revision as SHA-1, it has generally been accepted as the worldwide standard for digital data authentication and signing. A newer function called SHA-2 was published in 2001 to address some potential weaknesses, but it never surpassed the original in widespread adoption. Despite attempts, neither has been broken as of 2011.

NIST announced a global competition to find a new SHA function in 2007, and submissions were accepted for approximately one year. The algorithms are being analyzed and narrowed down through elimination rounds, based on the security, performance, and design of each function. The final round of candidates was announced in 2010, with public analysis encouraged. In 2012, a winner will be announced, and the algorithm will become the official SHA-3 standard for future cryptographic applications.

One of the five finalists is the BLAKE function, whose design and performance will be the focus of this paper.

II. BLAKE EXAMINED

This section discusses the BLAKE algorithm: its history, design, and cryptographic integrity.

A. Background

The BLAKE function was designed by a team of four individuals based in Switzerland. Its principal designer is Jean-Philippe Aumasson, a distinguished cryptography and computer science researcher. BLAKE’s algorithm is based on the “ChaCha” variant of the Salsa20 stream cipher, whose integrity had already been researched and discussed by Aumasson. (Although hashes are fundamentally different from the two-way nature of ciphers, it is common for them to share operations since they require similar cryptographic qualities.) The original specification was submitted to NIST in 2008, but it has been revised slightly through 2010 [1].

B. Structure

There are four official variants of BLAKE, distinguished by their output hash size, as required for the SHA-3 competition. The sizes are 224, 256, 384, and 512 bits, as designated after the algorithm name (eg. BLAKE-256). The sizes were chosen as the SHA-2 sizes doubled, since increases in computing power require increases in hash sizes to maintain acceptable levels of security (against brute force or other attacks). The variants also have slight differences in the size of words processed and maximum message length.
BLAKE is an iterative algorithm based on the well-known and highly-regarded Hash Iterative Framework (HAIFA) structure. An initial hash, \( h^0 \), is loaded from a predefined initialization vector. The BLAKE designers chose to reuse the same vectors as SHA-2 for their strong pseudo-random distributions. The input message is then segmented into a set of \( N \) equally-sized blocks (padded as necessary) and passed with the current hash through a compression function. The result is a new hash, and this process is repeated until the entire message has been “digested”. The last compression result is returned as the final message hash. This structure is summarized in Fig. 1. A major advantage of the HAIFA structure is that a long message can be hashed as it is streamed in, rather than requiring the entire message in available memory at once.

The compress function is a general feature of the HAIFA structure, but its exact functionality is defined by each hash’s specification. BLAKE’s compress is described in the next section. It is important to note that a “timer” value is passed to the function, counting the number of bits processed so far. This ensures that identical blocks at different locations in the input message will return different hash values. The BLAKE designers also chose to include a salt variable in the compression function, adding more security since the user can produce a set of different hashes for any given message. The inclusion of a salt is not required by NIST, and it defaults to zero when not implemented (reducing hardware area slightly) or not specified by the user.

C. BLAKE-256 Algorithm

The details of the BLAKE-256 variant will be described here. For simplicity, the other variants will only be described by the differences between them. (This is the format used by the designers’ specification.)

BLAKE-256 produces a 256-bit hash which is treated as eight 32-bit words. It accepts a message of length \( 0 \leq \ell < 2^{64} \). The message is appended with a 1 bit, a sequence of 0 bits, and its length \( \ell \) in 64-bit representation. The number of 0 bits is determined such that the total stream length is a multiple of 512 bits. This allows it to be partitioned into 512-bit blocks to be passed to the compress function for processing.

The compress function takes four inputs: the 256-bit current hash \( h^i \), a 512-bit block of the input message \( m^i \), a 128-bit salt \( s \), and a cumulative 64-bit message bit count \( (\ell^i) \). Note that \( \ell^i \) does not include padding bits, i.e. a 100-bit message would produce \( \ell^1 = 100 \), not 512. If the final block contains only padding bits, this is a special case and \( \ell^N \) is passed as 0 to ensure it is different from the previous iteration’s value.

An initial hash \( h^0 \) is needed, and it is derived from SHA-2 as mentioned. This is called the initialization vector \( IV \), and is shown in Fig. 2, as eight hexadecimal words.

Within the compress function, a 512-bit state \( v \) is maintained, treated as a 4x4 matrix of 32-bit words. This state is initialized from the current hash, salt value, timer value \( t \), and a 256-bit constant \( c \), as shown in Fig. 3. (The timer is the \( \ell^i \) value passed in; it is called a timer, rather than a length, inside the compress function.) The digits of constant \( c \) are directly taken from the hexadecimal representation of \( \pi \), chosen for its irrational nature (Fig. 4).

After initializing the state matrix, it is iteratively processed through 14 rounds. The BLAKE designers chose security through a small number of complex rounds, whereas some researchers prefer a large number of less complicated rounds. Both paradigms have supporters, but neither has been absolutely proven to provide greater security. (“Security” in this sense implies difficulty to invert, such that function inputs cannot be determined from function outputs.)

Each round consists of eight state transformations, labeled \( G_0 \) through \( G_7 \). These are responsible for the confusion (changes to data) and diffusion (dispersion of data) of the BLAKE algorithm. Each operates on and modifies only 4 of the 16 state words, generalized as \( a, b, c, \) and \( d \). The transformations (consisting of addition, bit-rotations, and exclusive or [XOR] operations) are listed in Fig. 5.

The G transformation is shown visually in Fig. 6. The index \( \sigma_r \) refers to one of ten permutations of the numbers 0 through 15, indexed by the round number \( r \) modulo 10. This is crucial for proper diffusion of input data. This permutation table is shown in Fig. 7.

The state words on which each G transformation operates were specifically chosen for efficiency. The first four operate on independent columns, and can be performed entirely in parallel. The last four operate on independent diagonals, and can be performed in parallel as well. Therefore the round can...
be viewed as two large operations (called the column step and diagonal step), rather than eight sequential operations. This essentially reduces round calculation time by 75% in hardware and some software implementations. This parallelization is illustrated in Fig. 8, showing which words are affected by each operation.

After fourteen rounds of G transformations, the compress function performs a finalization step. This generates the new 256-bit hash, $h_{i+1}$, as the large XOR of the previous hash $h_i$, the salt $s$, and the 512-bit state matrix $v$ (see Fig. 9). If the final block of the padded input stream has been processed, then this result is the final output of the BLAKE hash function.

D. Variant Differences

The other three variants are very similar to BLAKE-256. The 512-bit version doubles the bit sizes of most variables: all intermediate hashes, the compress state matrix, the maximum message size, the salt, and the timer. The initialization vector is taken from SHA-512, and the $c$ constants simply extend to more digits of $\pi$. The $\sigma$ permutation table remains the same. The only non-trivial difference is in the G step: the bit-rotation amounts are defined differently, rather than simply doubling the previous amounts. It is also recommended that the number of rounds be increased from 14 to 16; technically this parameter is selectable by the user as a tunable tradeoff between speed and security.

The 224-bit and 384-bit variants actually utilize the 256-bit and 512-bit algorithms, then truncate the output hash accordingly. This implies performance identical to the two variants already discussed. The only differences are the initialization vectors (matching SHA-224 and SHA-384) and the exclusion of the 1 bit when padding the input message.

E. Strengths

The design of all cryptographic hashes involves a balancing of security and performance (and power consumption, in hardware implementations). The major parameter in BLAKE that balances these is the number of rounds per compression.

With the unofficial goal of providing plenty of security while outperforming SHA-2, the original round counts were 10 (and 14) for the 256 (and 512) bit variants; these were increased to 14 and 16 in December 2010 for further security.

The algorithm was designed with much attention to data diffusion. “Full diffusion” means that a single changed input bit (message, salt, or initialization vector) can affect every output bit, and occurs after two transformation rounds. The compression function and G transformations were also designed to minimize the probability of local collisions, which occurs when two different messages produce the same internal state after the same number of rounds. Furthermore, the salt value (usually kept secret by the user) is passed into every compression (as opposed to just the initialization or finalization) to prevent determination of the salt from a known message-hash pair.
The BLAKE designers also specify four “toy versions”, intentionally weakened versions for cryptoanalysis purposes. These versions are:

- **BLOKE**: all σ permutations are simply 0-15, in order
- **FLAKE**: the compression function does not include the salt or current hash in its final XOR
- **BLAZE**: the G transformations use zeros in place of the c constants derived from π
- **BRAKE**: very weak, includes all three listed changes

Attack methods against these weak versions are being researched along with attacks against the complete function. As of December 2010, the best attack that the designers have confirmed is a preimage attack on 2.5 rounds, on a reduced-round version of BLAKE [2].

### III. IMPLEMENTATION

With the BLAKE algorithm already explained and analyzed, this section describes some methods of practical implementation. The focus of this section is hash performance, rather than its security or hardware requirements.

#### A. BLAKE Software

The BLAKE designers provide various software implementations of the hash in the C programming language. These range from easy-to-understand versions that closely follow the algorithm description, to highly optimized versions for specific key sizes and architectures. The official BLAKE website also links to various approved ports: Perl, PHP, Javascript, etc.

The “lightweight” 512-bit version will be briefly examined here. Since it is compact and designed for one specific variant, it requires minimal software and hardware resources, and can be run on many platforms. It also includes one Built-In Self Test (BIST) which hashes two hard-coded message and compares the result against the known correct hashes.

The hash function is called as

```plaintext
blake512_hash(digest, data, length)
```

where `digest` is a pointer to a 512-bit output buffer, `data` is a pointer to the input message, and `length` is the message length (in bytes here, rather than bits). The function creates a 4x4 state matrix and calls three subroutines: one to initialize it, one to process it, and one to finalize it. These correspond to the main steps described the BLAKE-256 description section.

The `blake512_update` function is responsible for padding the input message, segmenting it into 1024-bit blocks, and iteratively executing the compress function. This is implemented as `blake512_compress`, which executes 16 rounds, each consisting of eight G transformations using arithmetic macros. Note that this software implementation executes all eight sequentially; it does not utilize the parallelization technique described previously.

The hash result is written to `digest` and is typically displayed as a 128 character hexadecimal string. By default, the program hashes a 144 byte (1152 bit) message, but this can easily be adapted for testing different lengths. A timer will be added to the code for measuring hash performance and recording various benchmarks.

#### B. PC Performance

The BLAKE specification includes some performance tests of software implementations on microcontrollers and consumer Central Processing Units (CPUs). However these tests were run on the BLAKE-32 and BLAKE-64 algorithms — earlier versions of BLAKE-256 and BLAKE-512 which include fewer rounds per compression. (The recommended round counts have been increased since the initial 2008 submission.)

A summary of the early benchmarks on an Intel 64-bit 2.4 GHz Core 2 Duo processor are shown in Table I. Note that the 64-bit version is natively faster than the 32-bit version, the number of hash clock cycles per message byte decreases dramatically as the input grows (since computation overhead is distributed across a larger number of processed blocks), and the results for small message lengths (here, 10 bytes) are approximations since small execution variances affected the cycles/byte ratio significantly.

More recent experiments use the updated BLAKE variants. The third-party ECRYPT benchmarking project compared the five SHA-3 finalists on a large number of computer systems, across a wide range of message sizes. Their results are released into the public domain and are available at the project’s website [3].

Table II summarizes their benchmarks for BLAKE-512 on a 64-bit Intel Core 2 Duo CPU, quartile and median values of hash clock cycles per byte, for various message sizes.

<table>
<thead>
<tr>
<th>Message Bytes</th>
<th>Q1</th>
<th>Median</th>
<th>Q2</th>
</tr>
</thead>
<tbody>
<tr>
<td>81</td>
<td>159.00</td>
<td>316.50</td>
<td>316.50</td>
</tr>
<tr>
<td>64</td>
<td>19.69</td>
<td>38.81</td>
<td>39.00</td>
</tr>
<tr>
<td>1536</td>
<td>8.61</td>
<td>8.62</td>
<td>8.63</td>
</tr>
<tr>
<td>4096</td>
<td>8.02</td>
<td>8.03</td>
<td>8.05</td>
</tr>
<tr>
<td>long</td>
<td>7.65</td>
<td>7.69</td>
<td>7.74</td>
</tr>
</tbody>
</table>

1 results varied significantly
2 an approximate asymptotic limit
The SHA-3 candidates have also been analyzed on “exotic” architectures. Researchers at the Laboratory for Cryptographic Algorithms have tested the performance of all second round candidates on two specialized processors: the Cell Broadband Engine developed by Sony, Toshiba, and IBM, and graphical processing units (GPUs) developed by NVIDIA. Both are high-performance, multi-core processors that utilize Single Instruction Multiple Data (SIMD) and Single Instruction Multiple Threads (SIMT) paradigms respectively [4]. The GPU is typically used for heavy graphics processing, while the Cell processor is currently used, for example, in the high-end PlayStation 3 video game system.

These tests used different types of metrics for measuring performance. The candidates were divided into AES inspired (Advanced Encryption Standard, NIST’s current block cipher standard) and non-AES inspired hashes (under which BLAKE is categorized). The tests of the latter group were focused on their compression functions, rather than the entire algorithms. This does not account for setup time, finalization time, or memory copying time, except for the copying of chain values (such as $h^i$, for BLAKE). The researchers also focused on the number of times instructions were called, rather than explicit clock cycles. While some clock-cycles-per-message-byte estimations are extrapolated from this data, the relationship is not directly proportional.

The variant of BLAKE tested was BLAKE-32, the earlier version of BLAKE-256 which executes only ten compression rounds. Therefore the performance is expected to be greater than that of the BLAKE-512 tests above. (In fact, the performance should inherently be greater due to the nature of these specialized processors.) With appropriate optimizations and multi-threading, the compression algorithms were executed on the two platforms and their speeds were monitored. On the Cell processor, BLAKE-32 was the third fastest of the eleven candidates, measured at 5.0 cycles/byte. On the NVIDIA GPU, BLAKE-32 was tied with BMW-256 as the fastest, measured at 0.27 cycles/byte, with a theoretical peak rate of 0.13 cycles per byte [4]. These findings are important, as the winning SHA-3 hash will surely be implemented on a variety of modern, specialized platforms.

### Table III

<table>
<thead>
<tr>
<th>Message Bytes</th>
<th>Total Cycles</th>
<th>Cycles/Byte</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>32520</td>
<td>4065.00</td>
</tr>
<tr>
<td>64</td>
<td>32492</td>
<td>507.69</td>
</tr>
<tr>
<td>576</td>
<td>155367</td>
<td>269.73</td>
</tr>
<tr>
<td>1536</td>
<td>401002</td>
<td>261.07</td>
</tr>
<tr>
<td>4096</td>
<td>1015322</td>
<td>247.88</td>
</tr>
<tr>
<td>10000</td>
<td>2428348</td>
<td>242.83</td>
</tr>
</tbody>
</table>

C. FPGA Performance

For further insight, the lightweight version of BLAKE-512 was tested on a Field Programmable Gate Array (FPGA) system. The FPGA used is a Xilinx Virtex-5, with an integrated PowerPC microprocessor. The hardware platform (designed in Xilinx Platform Studio [XPS] for the ML507 evaluation board) is kept to a minimum: a single processor, one hardware timer, one Universal Asynchronous Receiver/Transmitter (UART) port for output. A 32 KB data/instruction cache is also enabled for greater performance.

Performance tests were run on the embedded processor, similar to the PC tests run by ECRYPT. The results are shown in Table III, as the total clock cycles required to hash messages (sized at 8, 64, 576, 1536, 4096, and 10000 bytes) and cycles per input byte.

The FPGA performance is clearly slower than the Core 2 Duo performance reported in Table II, by an approximate factor of 30. There are some clear reasons for this: the ECRYPT software was compiled with loop unrolling and similar optimizations, and executed on a multi-core processor that natively operates on 64-bit words. The single-core embedded processor operates on 32-bit words, so handling 64-bit values is slower, and its code was compiled with no explicit optimization. There are also some ambiguous differences between tests: the ECRYPT team’s memory access/caching system is not known, and they likely used different source code, perhaps a platform-optimized version. There are some optimizations suggested for...
Intel platforms specifically, such as using the Core 2’s native bit-rotate instruction in place of four previous instructions [5]. Finally, it should be mentioned that the content of the message being hashed has no effect on performance, since the transformations consist of primitive bitwise operations. All input words are processed without computational bias of any nature.

D. Hardware Acceleration

The FPGA performance of BLAKE could be greatly increased by implementing some or all of the algorithm as hardware components. The main improvement would be changing the eight $G$ transformations to the two parallelized “column step” and “diagonal step” operations described in the Algorithm section. This would have a major effect on total execution time, since it is computed 16 (or 14) times per compression, which itself is computed for every segment of the input stream.

Furthermore, the accessing and processing of data is naturally faster on hardware than in software. Many words can be loaded into parallel registers simultaneously, as opposed to the sequential loading of software words. Arithmetic operations such as XOR and addition can be executed in a single clock cycle, whereas software might require four or more. Diffusion operations such as permutation and bit-rotation can be directly implemented by the routing of data wires.

The BLAKE website offers a few VHSIC Hardware Description Language (VHDL) frameworks for hardware implementation. Research into their performance is currently underway, and some tests predict ideal throughput of about 2 clock cycles per message byte when hashing very long messages [6]. This is feasible since a single compression processes 64 (or 128) bytes, and since the cycles required for initialization and finalization become negligible as the input message grows.

IV. Conclusion

This document has introduced the BLAKE hash function, one of five finalists in the SHA-3 hash competition. Its history, uses, and relative strengths have been discussed. Its cryptographic structure and algorithm have also been examined in detail, leading to general conclusions about its security and performance. Software and hardware tests on real-world systems have been performed to judge its practical performance, allowing for informed comparisons with other hashes.

The future of the BLAKE algorithm and its potential widespread adoption will be determined in 2012, when NIST announces which hash function will become the new SHA-3 standard. There is still time for much analysis and discussion; all five candidates remain under intense public scrutiny. This will ensure that the selection provides an optimal balance between performance, resources, efficiency, and security.

REFERENCES