Quo vadis BLAKE?
Outline of this talk:

Specification

Design rationale

Security

Software performance

Hardware performance

Conclusion
CARLTON DRAUGHT CLASSIC MOMENTS IN COMMENTARY

"IT'S DEJA VU ALL OVER AGAIN."

CARLTON DRAUGHT MADE FROM BEER.
Find all info on

http://131002.net/blake/
http://en.wikipedia.org/wiki/BLAKE_(hash_function)
http://www.nist.gov/hash-competition
http://bench.cr.yp.to/results-sha3.html
http://ehash.iaik.tugraz.at/wiki/The_SHA-3_Zoo
http://xbx.das-labor.org
http://cryptography.gmu.edu/athenadb/table_view

First and Second SHA3 Conference presentations
Next talks of today and tomorrow
Etc.
The single most asked question...
The single most asked question... 

Why “BLAKE”? 

The single most asked question . . .

Why “BLAKE”?  
Why BLAKE?
The single most asked question... 

Why “BLAKE”? 
Why BLAKE? 
Why for SHA3?
PART 1: From LAKE to BLAKE
It all started with **LAKE**...
It all started with **LAK**E... 

“LAK**E” is the name I proposed in 2006 for what became ’was refused... (but the paper was accepted to ACISP’07)
It all started with **LAKE**... “LAKE” is the name I proposed in 2006 for what became ’was refused... (but the paper was accepted to ACISP’07)

Since worked hard to find this name, determined to put it on a future design...
2007 Oct: we submit the hash function LAKE to FSE 2008
2007 Nov: NIST announces the SHA3 competition

LAKE innovations: HAIFA, built-in salt, local wide-pipe
LAKE wasn’t good enough
Flaws in the compression, though hash unattacked

Collisions for Round-Reduced LAKE *

Florian Mendel and Martin Schläffer

Graz University of Technology
Institute for Applied Informatics and Cryptology
{florian.mendel, martin.schlaaffer}@tugraz.at

Cryptanalysis of the LAKE Hash Family

Alex Biryukov¹, Praveen Gauravaram³, Jian Guo², Dmitry Khovratovich¹, San Ling², Krystian Matusiewicz³, Ivica Nikolić¹, Josef Pieprzyk¹, and Huaxiong Wang²

Starting the development of BLAKE after FSE (Feb ’08)
From LAKE to BLAKE...

Keep HAIFA: counter & salt, avoids length extension

Keep the local wide-pipe and global narrow-pipe
  ► Straightforward no-collision proof for fixed block
  ► Larger state allows to add redundancy, counter, salt
  ► Narrow-pipe attacks not a concern in practice

Keep the compression algorithm, NOT
  ► Complete redesign needed
General design philosophy:

▶ KISS
▶ Think to users and implementers
▶ Don’t optimize
▶ Don’t reinvent the wheel

Understand the needs of SHA3

▶ Who will be the SHA3 users?
▶ Properties that are mandatory/desirable/superfluous?

Remember that SHA3 is an engineering competition, and not the place for experimental, untested, and inefficient designs (however interesting and technically deep)
BLAKE’s core: a robust, previously-analyzed design

The ChaCha family of stream ciphers

The ChaCha family of stream ciphers, also known as Salsa20, is a variant of the Salsa20 family of stream ciphers. The following paper introduces ChaCha and compare


The Salsa20 directory in version 2008.01.20 of the eSTREAM benchmarking suite includes several implementations of ChaCha8 (and ChaCha12 and ChaCha20):

- ctest, a reference implementation: `chaCha.c`, `Makefile`, `encrypt-sync.h`
- regen: `chaCha.c`, `Makefile`, `encrypt-sync.h`. Similar to ctest but uses separate temporary variables instead of a temporary array.
- mergen: `chaCha.c`, `Makefile`, `encrypt-sync.h`. Similar to regen but merges the ChaCha core.
- x86-1, specific to the Pentium, Athlon, and other x86 chips: `chaCha.c`, `chaCha.s`, `Makefile`, `encrypt-sync.h`
- x86-64x8, specific to the Pentium Pro and other CPUs with MMX registers: `chaCha.c`, `chaCha.s`, `Makefile`, `encrypt-sync.h` Uses some MMX registers as substitutes for x36-x8, specific to the Pentium 4 and other CPUs with SSE2 instructions: `chaCha.c`, `chaCha.s`, `Makefile`, `encrypt-sync.h`
- x36-x8, specific to the Pentium 4 and other CPUs with SSE2 instructions: `chaCha.c`, `chaCha.s`, `Makefile`, `encrypt-sync.h`. Selects different SSE2 instructions.
- x36-64x8, specific to the Pentium 4 and other CPUs with SSE2 instructions: `chaCha.c`, `chaCha.s`, `Makefile`, `encrypt-sync.h` Handles four blocks in parallel. Based on x36-x8, specific to the Athlon 64, Core 2 Duo, and other AMD64 chips: `chaCha.c`, `chaCha.s`, `Makefile`, `encrypt-sync.h`
- amd64-x8, specific to the Athlon 64, Core 2 Duo, and other AMD64 chips: `chaCha.c`, `chaCha.s`, `Makefile`, `encrypt-sync.h`. Translation of x36-x8, with some added
- x86-64x8, specific to the PowerPC G4 and other CPUs with Altivec instructions: `chaCha.c`, `Makefile`, `encrypt-sync.h`. Based on analogous Salsa20 code by Matt
- space, specific to the UltraSPARC and other 64-bit SPARC chips: `chaCha.c`, `chaCha.s`, `Makefile`, `encrypt-sync.h`

ChaCha’s core is a strong well-analyzed 4-word map

After several prototype designs, decided to extend the ChaCha permutation to form BLAKE’s core
Previous project (FSE’08)

Amazed at ChaCha/Salsa20’s simplicity and efficiency
Intrinsic $4 \times$ parallelism, faster diffusion in ChaCha

Motivations for ARX:

- Performance tradeoff HW/SW
- Easy to implement
- Fast confusion/diffusion
ChaCha’s simplistic “quarterround” function
Bijective transform of four 32-bit words (a,b,c,d)

\[
\begin{align*}
    a +&= b & d &= (a \oplus d) \ll 16 \\
    c +&= d & b &= (b \oplus c) \ll 12 \\
    a +&= b & d &= (a \oplus d) \ll 8 \\
    c +&= d & b &= (b \oplus c) \ll 7
\end{align*}
\]
BLAKE-256’s $G$ function

Repeated 112 times in BLAKE-256 (32-bit words)

\[
\begin{align*}
a &+= m_i \oplus k_i \\
a &+= b & d &= (a \oplus d) \gg 16 \\
c &+= d & b &= (b \oplus c) \gg 12 \\
a &+= m_j \oplus k_j \\
a &+= b & d &= (a \oplus d) \gg 8 \\
c &+= d & b &= (b \oplus c) \gg 7
\end{align*}
\]
BLAKE-512’s \( G \) function

Repeated 128 times in BLAKE-512 (64-bit words)

\[
\begin{align*}
a & += m_i \oplus k_i \\
a & += b \quad d = (a \oplus d) \ggg 32 \\
c & += d \quad b = (b \oplus c) \ggg 25 \\
a & += m_j \oplus k_j \\
a & += b \quad d = (a \oplus d) \ggg 16 \\
c & += d \quad b = (b \oplus c) \ggg 11
\end{align*}
\]
Counting ARX ops:

<table>
<thead>
<tr>
<th></th>
<th>BLAKE-256</th>
<th>BLAKE-512</th>
</tr>
</thead>
<tbody>
<tr>
<td>Word</td>
<td>32-bit</td>
<td>64-bit</td>
</tr>
<tr>
<td>+</td>
<td>672</td>
<td>768</td>
</tr>
<tr>
<td>⊕</td>
<td>672</td>
<td>768</td>
</tr>
<tr>
<td>≪</td>
<td>448</td>
<td>512</td>
</tr>
<tr>
<td>Total</td>
<td>1792</td>
<td>2048</td>
</tr>
<tr>
<td>Ops/word</td>
<td>112</td>
<td>128</td>
</tr>
<tr>
<td>Ops/byte</td>
<td>3.5</td>
<td>2.0</td>
</tr>
</tbody>
</table>
BLAKE’s $4 \times 4$ internal state

Initialized with chaining value, salt, counter, constants
A BLAKE round:

Apply the $G$ function to each column
A BLAKE round:

Apply the $\mathbf{G}$ function to each column
A BLAKE round:

Apply the $\mathbf{G}$ function to each column
A BLAKE round:

Apply the $G$ function to each column

```
+----------------+----------------+----------------+----------------+
|                |                |                |                |
|                |                |                |                |
|                |                |                |                |
|                |                |                |                |
|                |                |                |                |
|                |                |                |                |
|                |                |                |                |
|                |                |                |                |
+----------------+----------------+----------------+----------------+```
A BLAKE round:

Apply the \textbf{G} function to each column (in parallel)
A BLAKE round:

Apply the $G$ function to each diagonal
A BLAKE round:
Apply the $G$ function to each diagonal
A BLAKE round:
Apply the **G** function to each diagonal
A BLAKE round:

Apply the $G$ function to each diagonal
A BLAKE round:

Apply the \textbf{G} function to each diagonal (in parallel)
Why the name “BLAKE”? 

- Expresses the LAKE legacy
- Can be understood as “Better LAKE” (unintentional)
- Short, simple to write and to correctly pronounce
- No negative meaning or translation
- Reference to William Blake
More popular Blake’s (according to Google):
PART 2: BLAKE’s unique qualities
Simplicity

Versatility

Security
Simplicity

Easy-to-understand specs
- Simplified HAIFA mode
- Familiar $4 \times 4$ state representation
- A single core function: $G$
- Only ops used are standard $\oplus$, $\ll$
- Repetition of just 3 lines of code
Simplicity

Easy-to-implement

- Clean version in 185 lines of C
- Small “attack surface” for coding errors
- Only need implement $G$, plus administrative code
- Reduces production costs (debug time, etc.)

“simple and clear design”, in NIST 2nd Round Report
void blake256_compress( state *S, const u8 *block ) {

    u32 v[16], m[16], i;
#define ROT(x,n) (((x)<<((32-n))|( (x)>>((n)) )))
#define G(a,b,c,d,e) \
    v[a] += (m[sigma[i][e]] ^ cst[sigma[i][e+1]]) + v[b]; \
    v[d] = ROT( v[d] ^ v[a],16); \
    v[c] += v[d]; \
    v[b] = ROT( v[b] ^ v[c],12); \
    v[a] += (m[sigma[i][e+1]] ^ cst[sigma[i][e]])+v[b]; \
    v[d] = ROT( v[d] ^ v[a], 8); \
    v[c] += v[d]; \
    v[b] = ROT( v[b] ^ v[c], 7);

    for(i=0; i<16;++i) m[i] = U8TO32(block + i*4);
    for(i=0; i< 8;++i) v[i] = S->h[i];
    v[ 8] = S->s[0] ^ 0x243F6A88; v[12] = 0xA4093822;
    v[ 9] = S->s[1] ^ 0x85A308D3; v[13] = 0x299F31D0;

    if (S->nullt == 0) {
        v[12] ^= S->t[0]; v[13] ^= S->t[0];
    }
}
#define G(a,b,c,d,e)  
  v[a] += (m[sigma[i][e]] ^ cst[sigma[i][e+1]]) + v[b];  
  v[d] = ROT(v[d] ^ v[a],16);  
  v[c] += v[d];  
  v[b] = ROT(v[b] ^ v[c],12);  
  v[a] += (m[sigma[i][e+1]] ^ cst[sigma[i][e]]) + v[b];  
  v[d] = ROT(v[d] ^ v[a],8);  
  v[c] += v[d];  
  v[b] = ROT(v[b] ^ v[c],7);  

for(i=0; i<16;++i) m[i] = U8T032(block + i*4);  
for(i=0; i<8;++i) v[i] = S->h[i];  

v[8] = S->s[0] ^ 0x243F6A88; v[12] = 0xA4093822;  
v[9] = S->s[1] ^ 0x85A308D3; v[13] = 0x299F31D0;  
if (S->nullt == 0) {  
  v[12] ^= S->t[0]; v[13] ^= S->t[0];  
}  
for(i=0; i<14; ++i) {  
  G(0, 4, 8,12, 0);  
  G(1, 5, 9,13, 2);  
  G(2, 6,10,14, 4);
v[c] += v[d];
v[b] = ROT(v[b] ^ v[c], 12); 

v[a] += (m[sigma[i][e+1]] ^ cst[sigma[i][e]])+v[b]; 

v[d] = ROT(v[d] ^ v[a], 8);  

v[c] += v[d]; \  

for(i=0; i<16;++i) m[i] = U8TO32(block + i*4); 

for(i=0; i< 8;++i) v[i] = S->h[i];  

v[ 8] = S->s[0] ^ 0x243F6A88; v[12] = 0xA4093822;  

v[ 9] = S->s[1] ^ 0x85A308D3; v[13] = 0x299F31D0;  


if (S->nullt == 0) {  
    v[12] ^= S->t[0]; v[13] ^= S->t[0];  
}

for(i=0; i<14; ++i) {  
    G( 0, 4, 8,12, 0);  
    G( 1, 5, 9,13, 2);  
    G( 2, 6,10,14, 4);  
    G( 3, 7,11,15, 6);  
    G( 3, 4, 9,14,14);  
    G( 2, 7, 8,13,12);
v[d] = ROT( v[d] ^ v[a], 8); \\
v[c] += v[d]; \\
v[b] = ROT( v[b] ^ v[c], 7);

for(i=0; i<16;++i) m[i] = U8T032(block + i*4);
for(i=0; i< 8;++i) v[i] = S->h[i];
v[ 8] = S->s[0] ^ 0x243F6A88; v[12] = 0xA4093822;
v[ 9] = S->s[1] ^ 0x85A308D3; v[13] = 0x299F31D0;
if (S->nullt == 0) {
    v[12] ^= S->t[0]; v[13] ^= S->t[0];
}
for(i=0; i<14; ++i) {
    G( 0, 4, 8,12, 0);
    G( 1, 5, 9,13, 2);
    G( 2, 6,10,14, 4);
    G( 3, 7,11,15, 6);
    G( 3, 4, 9,14,14);
    G( 2, 7, 8,13,12);
    G( 0, 5,10,15, 8);
    G( 1, 6,11,12,10);
}
for(i=0; i<16;++i) m[i] = U8TO32(block + i*4);
for(i=0; i<8;++i) v[i] = S->h[i];

v[ 8] = S->s[0] ^ 0x243F6A88; v[12] = 0xA4093822;
v[ 9] = S->s[1] ^ 0x85A308D3; v[13] = 0x299F31D0;

if (S->nullt == 0) {
    v[12] ^= S->t[0]; v[13] ^= S->t[0];
}

for(i=0; i<14; ++i) {
    G( 0, 4, 8,12, 0);
    G( 1, 5, 9,13, 2);
    G( 2, 6,10,14, 4);
    G( 3, 7,11,15, 6);
    G( 3, 4, 9,14,14);
    G( 2, 7, 8,13,12);
    G( 0, 5,10,15, 8);
    G( 1, 6,11,12,10);
}

for(i=0; i<16;++i) S->h[i%8] ^= v[i];
for(i=0; i<8 ;++i) S->h[i] ^= S->s[i%4];
BLAKE is not optimized for any specific platform

- 32- and 64-bit versions
- 256-bit digest may be produced by truncation of BLAKE-512
- Rotations multiple of 8 to simplify 8- and 16-bit implementations
- HW-friendly structure
  - Single building block $\mathbf{G}$ allows compact impl
  - Straightforward parallelism
BLAKE can be compact in FPGA

A few days ago at the ECRYPT2 Hash 2011 Workshop: Kerckhof et al., *Compact FPGA Implementations of the Five SHA-3 Finalists:*

On Virtex 6:

<table>
<thead>
<tr>
<th>Properties</th>
<th>BLAKE</th>
<th>Grøstl</th>
<th>JH</th>
<th>Keccak</th>
<th>Skein</th>
<th>AES</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input block message size</td>
<td>512</td>
<td>512</td>
<td>512</td>
<td>1088</td>
<td>256</td>
<td>128</td>
</tr>
<tr>
<td>Clock cycles per block</td>
<td>1182</td>
<td>176</td>
<td>688</td>
<td>2137</td>
<td>230</td>
<td>44</td>
</tr>
<tr>
<td>Clock cycles overhead (pre/post)</td>
<td>12/8</td>
<td>122</td>
<td>16/20</td>
<td>17/16</td>
<td>5/234</td>
<td>8/0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Area</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of LUTs</td>
<td>417</td>
<td>907</td>
<td>789</td>
<td>519</td>
<td>770</td>
<td>658</td>
</tr>
<tr>
<td>Number of Registers</td>
<td>211</td>
<td>566</td>
<td>411</td>
<td>429</td>
<td>158</td>
<td>364</td>
</tr>
<tr>
<td>Number of Slices</td>
<td>117</td>
<td>285</td>
<td>240</td>
<td>144</td>
<td>240</td>
<td>205</td>
</tr>
<tr>
<td>Frequency (MHz)</td>
<td>274</td>
<td>815</td>
<td>288</td>
<td>250</td>
<td>160</td>
<td>222</td>
</tr>
<tr>
<td>Throughput (Mbit/s)</td>
<td>105</td>
<td>815</td>
<td>214</td>
<td>128</td>
<td>179</td>
<td>646</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Timing</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of LUTs</td>
<td>500</td>
<td>966</td>
<td>1034</td>
<td>610</td>
<td>1039</td>
<td>845</td>
</tr>
<tr>
<td>Number of Registers</td>
<td>284</td>
<td>571</td>
<td>463</td>
<td>533</td>
<td>506</td>
<td>524</td>
</tr>
<tr>
<td>Number of Slices</td>
<td>175</td>
<td>293</td>
<td>304</td>
<td>188</td>
<td>291</td>
<td>236</td>
</tr>
<tr>
<td>Frequency (MHz)</td>
<td>347</td>
<td>330</td>
<td>299</td>
<td>285</td>
<td>200</td>
<td>250</td>
</tr>
<tr>
<td>Throughput (Mbit/s)</td>
<td>132</td>
<td>960</td>
<td>222</td>
<td>145</td>
<td>223</td>
<td>727</td>
</tr>
</tbody>
</table>
BLAKE is hardware-friendly

A few days ago at the ECRYPT2 Hash 2011 Workshop:
Homsirikamol, Rogawski, Gaj, Comparing Hardware Performance of Round 3 SHA-3 Candidates using Multiple Hardware Architecture in Xilinx and Altera FPGAs:

“BLAKE is the algorithm with the highest flexibility, and the largest number of potential architectures. It can be easily folded horizontally and vertically by factors of two and four. It can also be easily pipelined even in the folded architectures. It is also the only algorithm that has a relatively efficient architecture that is smaller than the basic iterative architecture of SHA-2. Finally, BLAKE is the only algorithm that can benefit substantially from using embedded block memories of both Xilinx and Altera FPGAs.”
BLAKE is often faster than SHA2 in SW

amd64, 2833MHz, Intel Core 2 Quad Q9550 (10677), 2008, berlekamp, supercop-20110508

<table>
<thead>
<tr>
<th>Cycles/byte for long messages</th>
<th>Cycles/byte for 4096 bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>quartile</td>
<td>median</td>
</tr>
<tr>
<td>6.47</td>
<td>6.48</td>
</tr>
<tr>
<td>6.84</td>
<td>6.85</td>
</tr>
<tr>
<td>7.50</td>
<td>7.55</td>
</tr>
<tr>
<td>7.65</td>
<td>7.70</td>
</tr>
<tr>
<td>7.68</td>
<td>7.77</td>
</tr>
<tr>
<td>7.74</td>
<td>7.90</td>
</tr>
<tr>
<td>8.65</td>
<td>8.82</td>
</tr>
<tr>
<td>10.24</td>
<td>10.26</td>
</tr>
<tr>
<td>10.26</td>
<td>10.27</td>
</tr>
<tr>
<td>10.89</td>
<td>10.90</td>
</tr>
<tr>
<td>11.65</td>
<td>11.69</td>
</tr>
<tr>
<td>12.59</td>
<td>12.64</td>
</tr>
<tr>
<td>15.14</td>
<td>15.26</td>
</tr>
<tr>
<td>6.65</td>
<td>6.65</td>
</tr>
<tr>
<td>7.14</td>
<td>7.15</td>
</tr>
<tr>
<td>7.85</td>
<td>7.86</td>
</tr>
<tr>
<td>7.91</td>
<td>7.95</td>
</tr>
<tr>
<td>8.00</td>
<td>8.02</td>
</tr>
<tr>
<td>7.99</td>
<td>8.05</td>
</tr>
<tr>
<td>8.99</td>
<td>9.03</td>
</tr>
<tr>
<td>10.79</td>
<td>10.80</td>
</tr>
<tr>
<td>10.80</td>
<td>10.81</td>
</tr>
<tr>
<td>11.39</td>
<td>11.40</td>
</tr>
<tr>
<td>12.15</td>
<td>12.16</td>
</tr>
<tr>
<td>13.10</td>
<td>13.12</td>
</tr>
<tr>
<td>15.72</td>
<td>15.73</td>
</tr>
</tbody>
</table>

Speedup from SSE (2 – 4.1) and XOP instructions, but very fast without (cf. SPHLIB code)
BLAKE is low-memory on microcontrollers

On 8-bit ATmega1281, from Wenzel-Benner 2010 slides
NIST does not disagree

In the 2nd Round Report:

“BLAKE is among the top performers in software across most platforms for long messages. BLAKE-32 is the best performer on software platforms for very short message”

“This flexibility allows cost-effective tradeoffs in area usage, with limited impact on the throughput-to-area ratio”
Plenty of cryptanalysis:

Dunkelman, Khovratovich (Hash 2011)
A. Leurent, Meier, Mendel, Mouha, Phan, Sasaki, Susil (Hash 2011)
Biryukov, Nikolic, Roy (FSE 2011)
Ming, Qiang, Zeng (ICCIS 2010)
Turan, Uyan (2nd SHA3 Conf)
Vidali, Nose, Pasalic (IPL 110(14-15))
Su, Wu, Dong (ePrint 2010/355)
A. Guo, Knellwolf, Matusiewicz, Meier (FSE 2010)
Guo, Matusiewicz (WEWoRC 2009)
Ji, Liangyu (ePrint 2009/238)
Security

Best attack on the compression function:
- Boomerang distinguisher by Biryukov/Nikolic/Roy
- 7 rounds in $2^{232}$ (BLAKE-256)

Best attack on the hash function:
- Preimage attack by Ji/Liangyu
- 2.5 rounds in $2^{241}$, $2^{481}$

14 rounds in BLAKE-256

Security margin compares favorably with other finalists
CONCLUSION
Why BLAKE for SHA3?
Why BLAKE for SHA3?

- High security margin
- User-oriented design
- Often faster than SHA2

Which hash function will become SHA-3?

A: Groestl  B: JH
Keccak  Skein
Why BLAKE for SHA3?

- High security margin
- User-oriented design
- Often faster than SHA2
- Performance versatile
- Fastest on small messages
- Fast portable C code
Even Schneier’s blog commenters like BLAKE!

Congrats, Bruce, although I think they're going to end up going with BLAKE.

Posted by: bloke at March 2, 2011 1:49 PM

@ Clive

No disrespect to Bruce, but my favorites are BLAKE and Keccak. Their performance characteristics make them more versatile in modern organizations with so many different devices and systems. The military could also make use of them because the developers of Type 1 encryption have switched from hardwired implementations to side-channel resistant RISC processors custom designed for crypto algorithms. This is so the algorithms can be changed without buying new hardware. I'm sure these two ciphers would have excellent performance on Type 1 devices, commercial FPGA's, desktops, servers, and mobile devices alike.

Posted by: Nick P at March 2, 2011 2:28 PM
Recent implementations of BLAKE

Python (by Larry Bugbee)

http://tinyurl.com/pyblake

def G(a, b, c, d, i):
    va = v[a]  # it's faster to deref and reref later
    vb = v[b]
    vc = v[c]
    vd = v[d]

    sri = SIGMA[round][i]
    sr1 = SIGMA[round][i+1]

    va = ((va + vb) + (m[sri] ^ cxx[sr1])) & MASK
    x = vd ^ va
    vd = (x >> rot1) | ((x << (WORDBITS-rot1)) & MASK)
    vc = (vc + vd) & MASK
    x = vb ^ vc
    vb = (x >> rot2) | ((x << (WORDBITS-rot2)) & MASK)

    va = ((va + vb) + (m[sr1] ^ cxx[sri])) & MASK
    x = vd ^ va
    vd = (x >> rot3) | ((x << (WORDBITS-rot3)) & MASK)
    vc = (vc + vd) & MASK
    x = vb ^ vc
    vb = (x >> rot4) | ((x << (WORDBITS-rot4)) & MASK)

    v[a] = va
    v[b] = vb
    v[c] = vc
    v[d] = vd

PHP (by Danniel Correa)

http://tinyurl.com/phpblake1

http://tinyurl.com/phpblake2

```php
<?php
    echo blake('sinfocol', BLAKE_224) . PHP_EOL;
    echo blake('sinfocol', BLAKE_256) . PHP_EOL;
    echo blake('sinfocol', BLAKE_384) . PHP_EOL;
    echo blake('sinfocol', BLAKE_512) . PHP_EOL;
    $blake = blake_init(BLAKE_256, '0123456789abcdef');
    blake_update($blake, 'sinfocol');
    echo blake_final($blake);
*/

// Screenshot
```
BLAKE