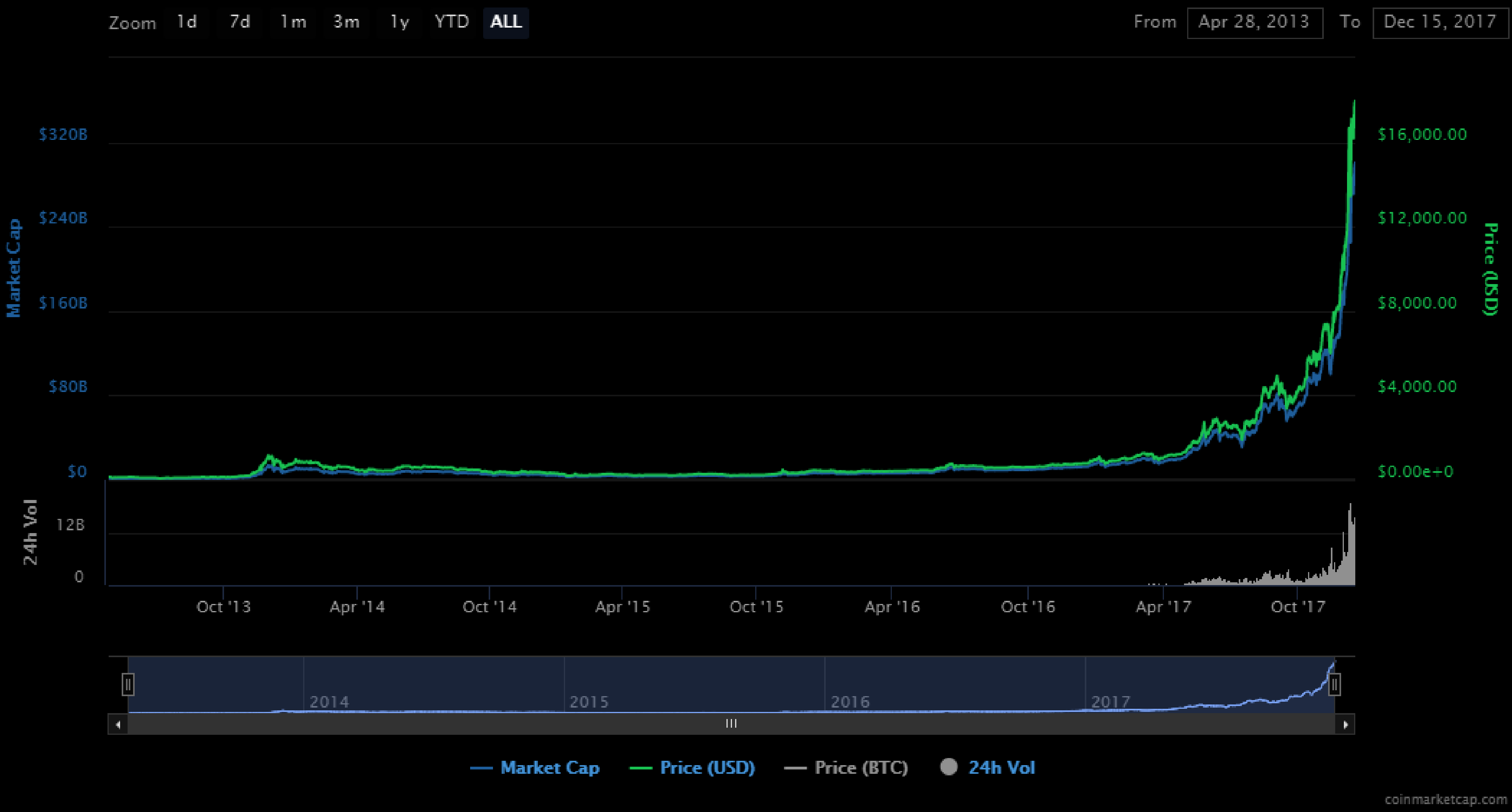
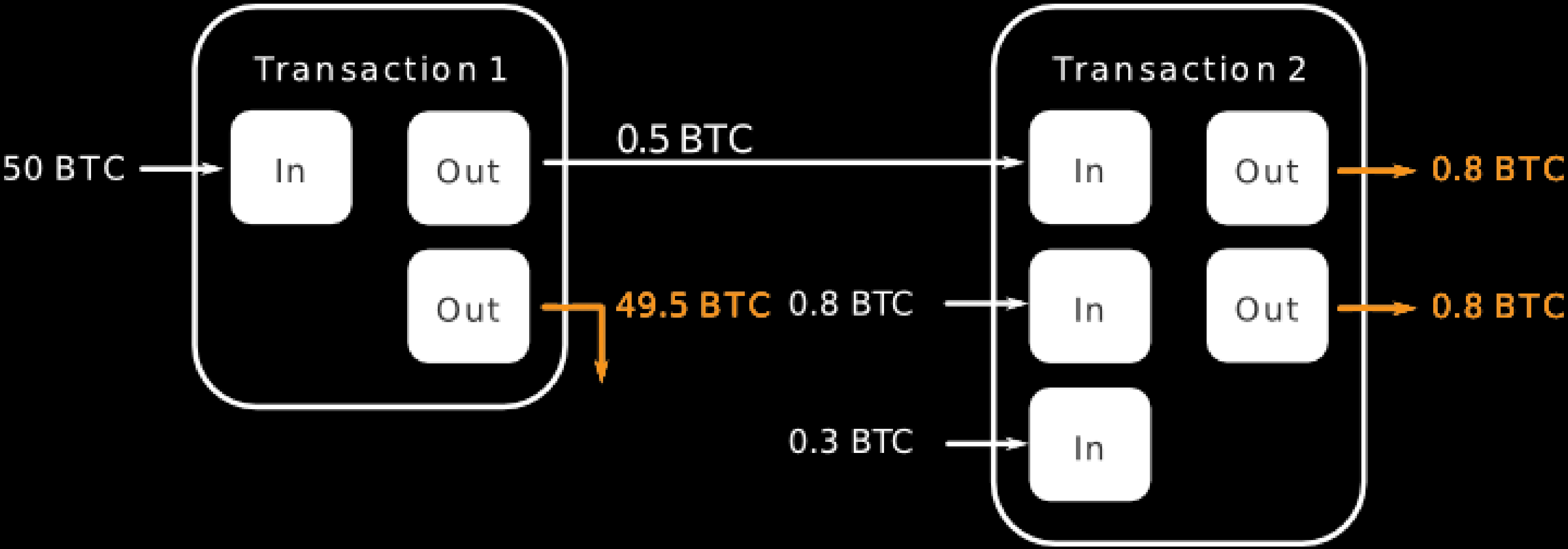
Mining with SHA-256

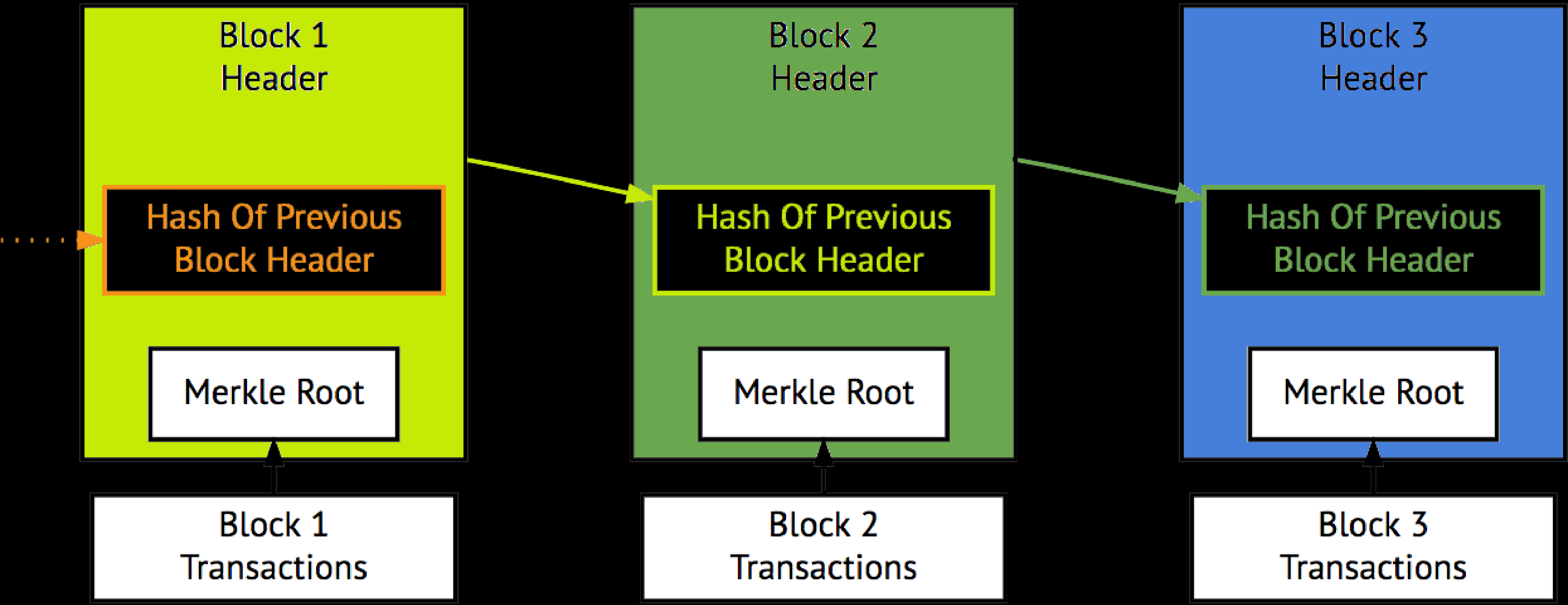
Jeremy Maxey-Vesperman
Zach Butler

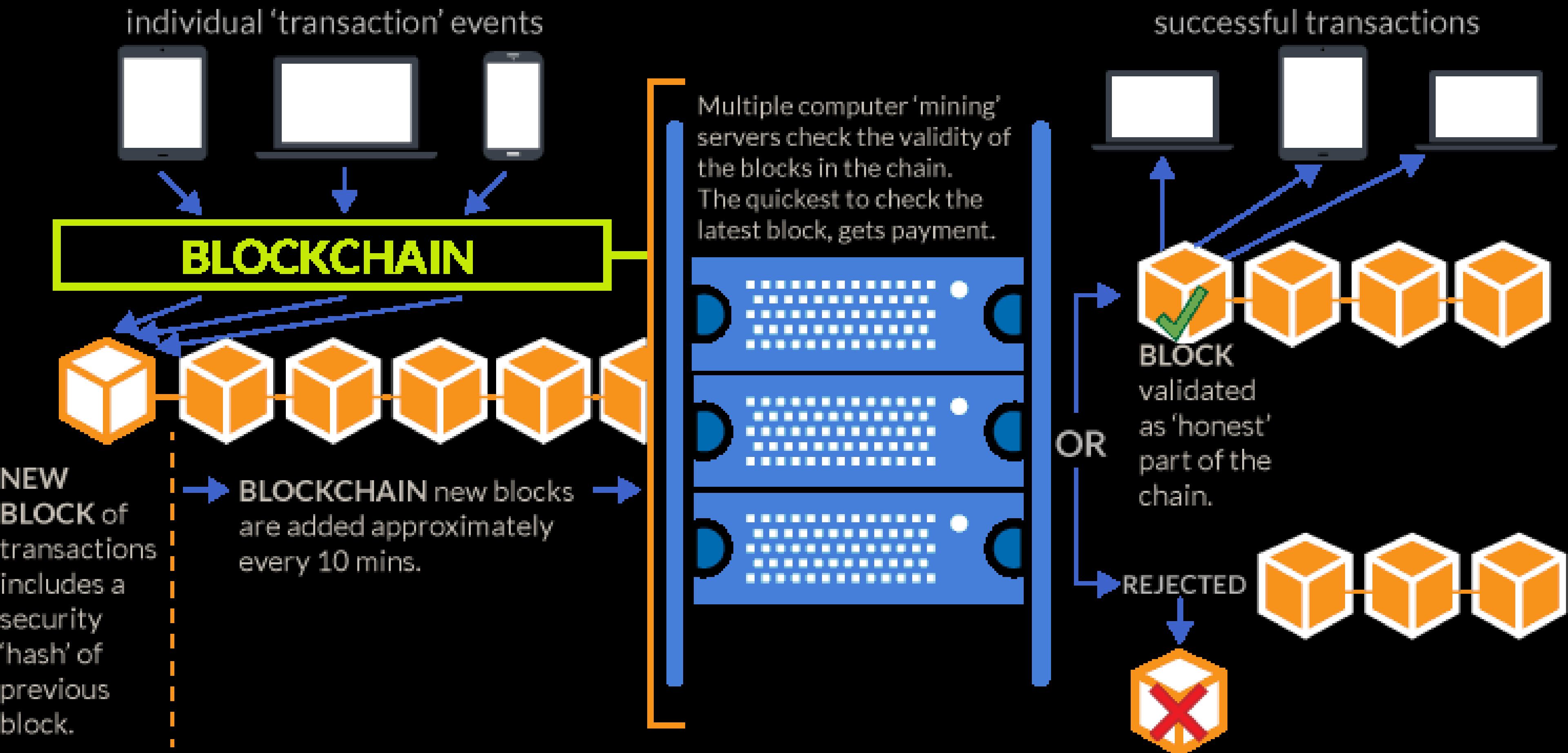
2017-12-15
CE-412

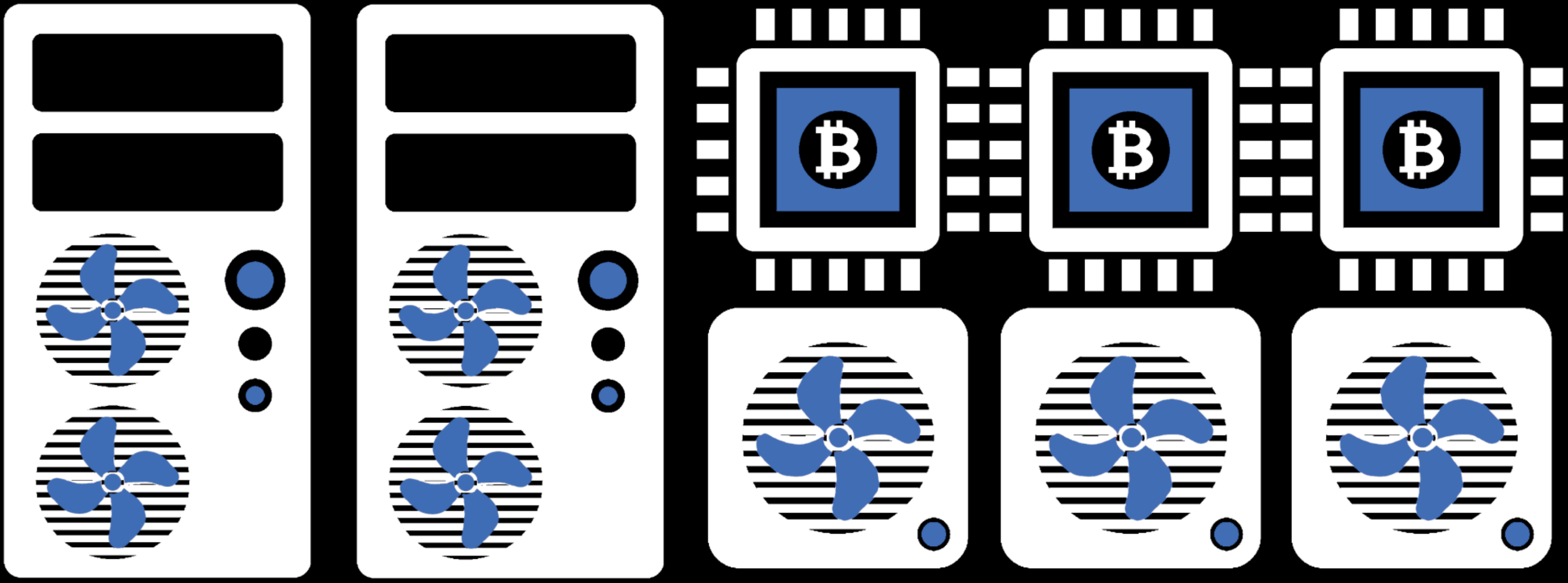


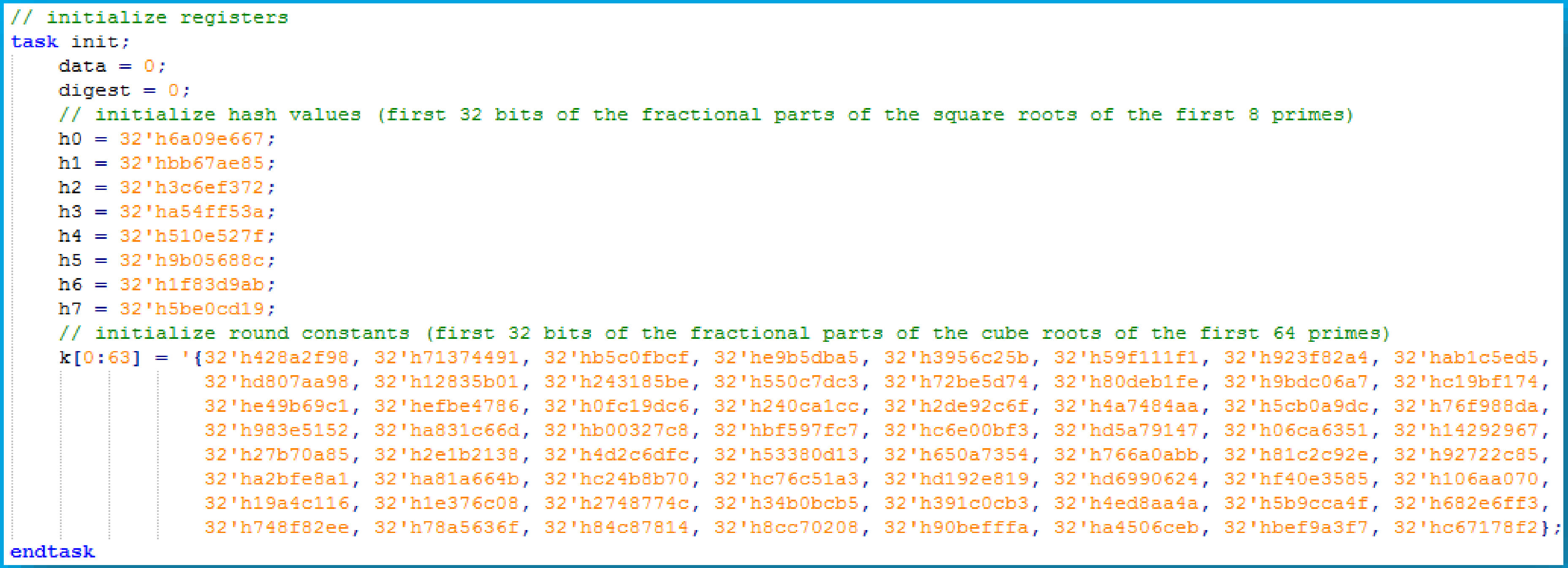
Zoom
1d
7d
1m
3m
1y
YTD
ALL
From
Apr 28, 2013
To
Dec 15, 2017
Market Cap
$320B
$240B
$160B
$80B
$0
Price (USD)
$16,000.00
$12,000.00
$8,000.00
$4,000.00
$0.00e+0
24h Vol
12B
0
Oct '13
Apr '14
Oct '14
Apr '15
Oct '15
Apr '16
Oct '16
Apr '17
Oct '17
2014
2015
2016
2017
Market Cap
Price (USD)
Price (BTC)
24h Vol
coinmarketcap.com


Transaction 1
50 BTC
In
Out
Out
0.5 BTC
49.5 BTC
Transaction 2
0.8 BTC
0.3 BTC
In
In
In
Out
Out
0.8 BTC
0.8 BTC

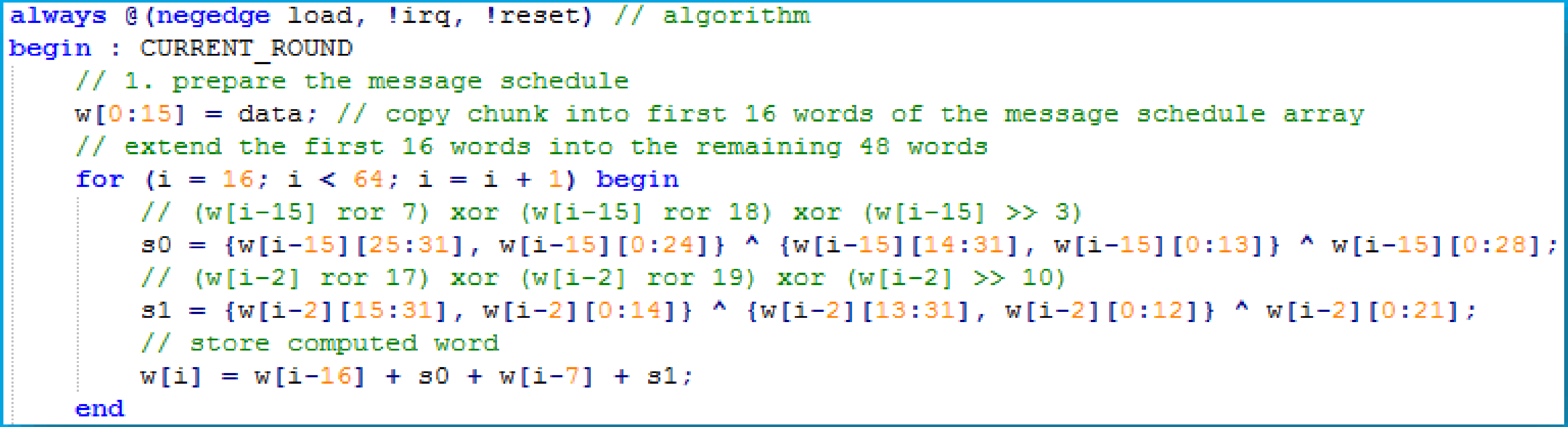

Block 1
Header
Hash Of Previous
Block Header
Merkle Root
Block 1
Transactions
Block 2
Header
Hash Of Previous
Block Header
Merkle Root
Block 2
Transactions
Block 3
Header
Hash Of Previous
Block Header
Merkle Root
Block 3
Transactions


individual 'transaction' events
BLOCKCHAIN
NEW BLOCK of transactions includes a security 'hash' of previous block.
BLOCKCHAIN new blocks are added approximately every 10 mins.
Multiple computer 'mining' servers check the validity of the blocks in the chain. The quickest to check the latest block, gets payment.
OR
successful transactions
BLOCK validated as 'honest' part of the chain.
REJECTED

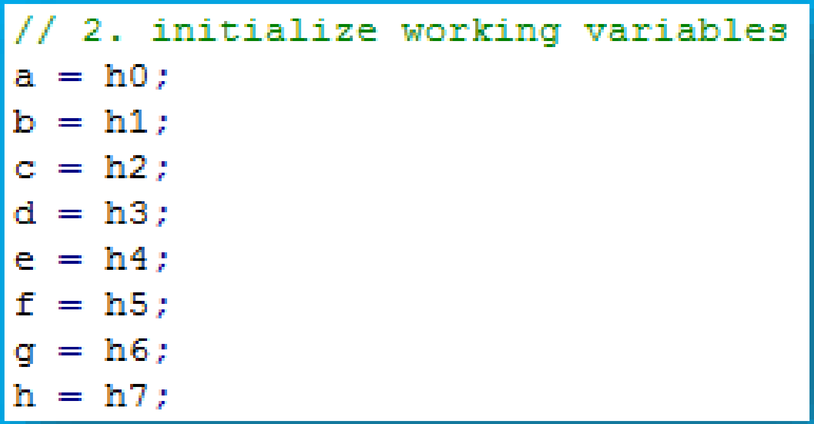




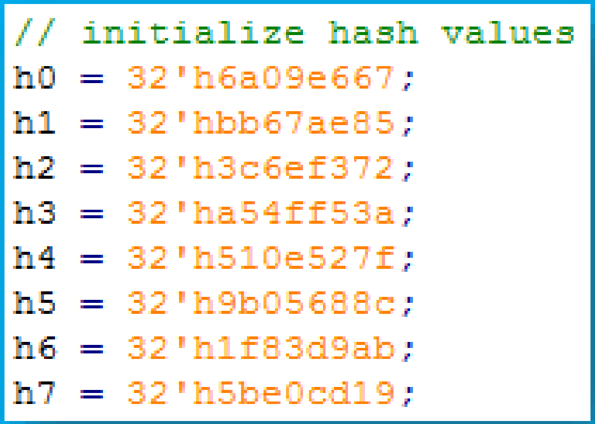
```verilog
// initialize registers
task init;
    data = 0;
    digest = 0;
    // initialize hash values (first 32 bits of the fractional parts of the square roots of the first 8 primes)
    h0 = 32'h6a09e667;
    h1 = 32'hbb67ae85;
    h2 = 32'h3c6ef372;
    h3 = 32'ha54ff53a;
    h4 = 32'h510e527f;
    h5 = 32'h9b05688c;
    h6 = 32'h1f83d9ab;
    h7 = 32'h5be0cd19;
    // initialize round constants (first 32 bits of the fractional parts of the cube roots of the first 64 primes)
    k[0:63] = '{32'h428a2f98, 32'h71374491, 32'hb5c0fbcf, 32'he9b5dba5, 32'h3956c25b, 32'h59f111f1, 32'h923f82a4, 32'hab1c5ed5,
                32'hd807aa98, 32'h12835b01, 32'h243185be, 32'h550c7dc3, 32'h72be5d74, 32'h80deb1fe, 32'h9bdc06a7, 32'hc19bf174,
                32'he49b69c1, 32'hefbe4786, 32'h0fc19dc6, 32'h240ca1cc, 32'h2de92c6f, 32'h4a7484aa, 32'h5cb0a9dc, 32'h76f988da,
                32'h983e5152, 32'ha831c66d, 32'hb00327c8, 32'hbf597fc7, 32'hc6e00bf3, 32'hd5a79147, 32'h06ca6351, 32'h14292967,
                32'h27b70a85, 32'h2e1b2138, 32'h4d2c6dfc, 32'h53380d13, 32'h650a7354, 32'h766a0abb, 32'h81c2c92e, 32'h92722c85,
                32'ha2bfe8a1, 32'ha81a664b, 32'hc24b8b70, 32'hc76c51a3, 32'hd192e819, 32'hd6990624, 32'hf40e3585, 32'h106aa070,
                32'h19a4c116, 32'h1e376c08, 32'h2748774c, 32'h34b0bcb5, 32'h391c0cb3, 32'h4ed8aa4a, 32'h5b9cca4f, 32'h682e6ff3,
                32'h748f82ee, 32'h78a5636f, 32'h84c87814, 32'h8cc70208, 32'h90befffa, 32'ha4506ceb, 32'hbef9a3f7, 32'hc67178f2};
endtask
```



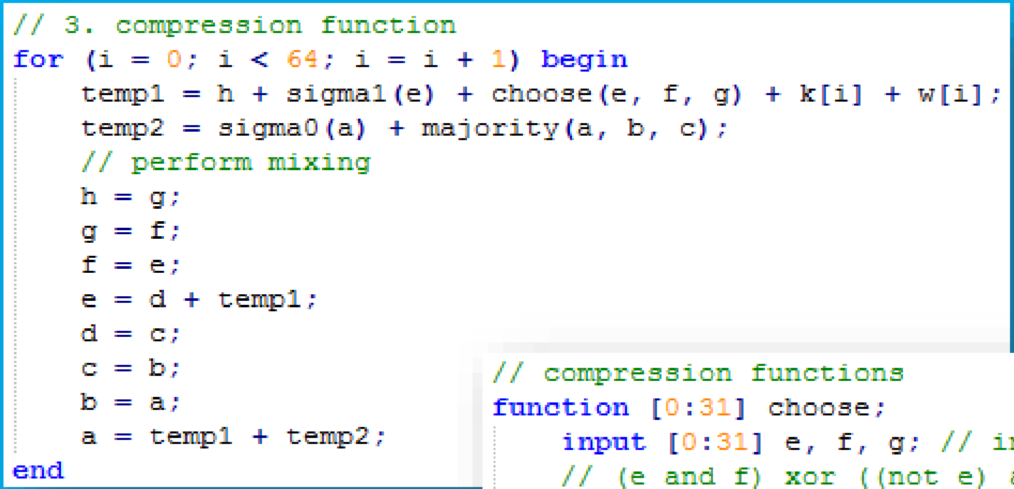
```
always @(negedge load, !irq, !reset) // algorithm
begin : CURRENT_ROUND
    // 1. prepare the message schedule
    w[0:15] = data; // copy chunk into first 16 words of the message schedule array
    // extend the first 16 words into the remaining 48 words
    for (i = 16; i < 64; i = i + 1) begin
        // (w[i-15] ror 7) xor (w[i-15] ror 18) xor (w[i-15] >> 3)
        s0 = {w[i-15][25:31], w[i-15][0:24]} ^ {w[i-15][14:31], w[i-15][0:13]} ^ w[i-15][0:28];
        // (w[i-2] ror 17) xor (w[i-2] ror 19) xor (w[i-2] >> 10)
        s1 = {w[i-2][15:31], w[i-2][0:14]} ^ {w[i-2][13:31], w[i-2][0:12]} ^ w[i-2][0:21];
        // store computed word
        w[i] = w[i-16] + s0 + w[i-7] + s1;
    end
```



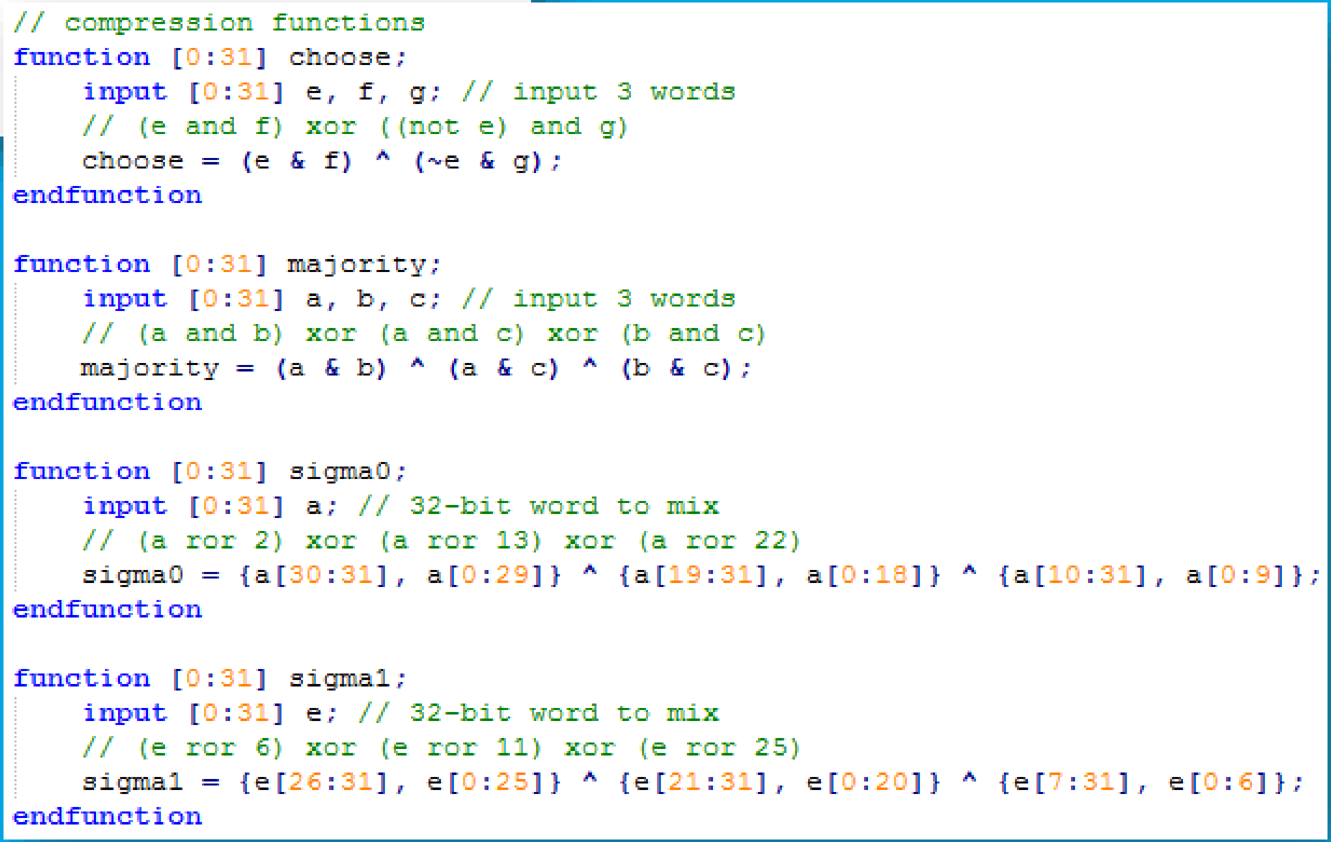
```
// 2. initialize working variables
a = h0;
b = h1;
c = h2;
d = h3;
e = h4;
f = h5;
g = h6;
h = h7;
```



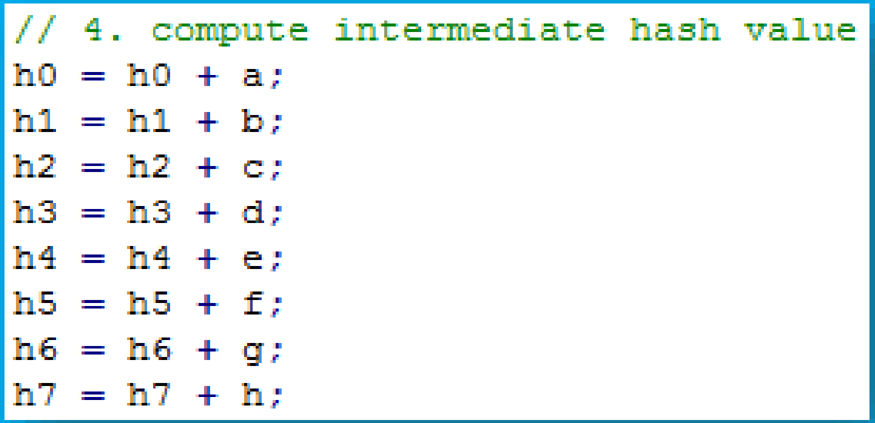
```
// initialize hash values
h0 = 32'h6a09e667;
h1 = 32'hbb67ae85;
h2 = 32'h3c6ef372;
h3 = 32'ha54ff53a;
h4 = 32'h510e527f;
h5 = 32'h9b05688c;
h6 = 32'h1f83d9ab;
h7 = 32'h5be0cd19;
```



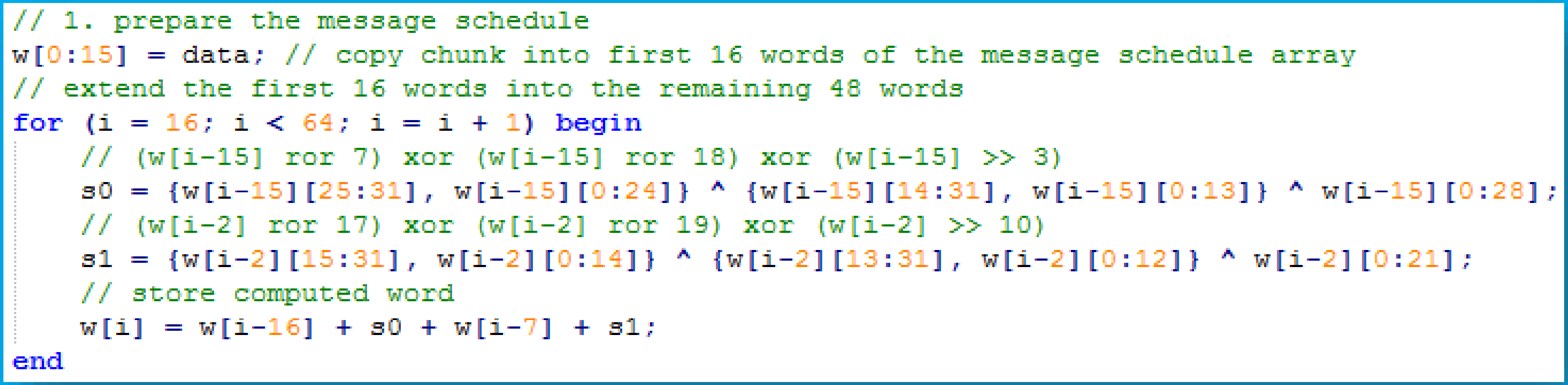
```
// 3. compression function
for (i = 0; i < 64; i = i + 1) begin
    temp1 = h + sigma1(e) + choose(e, f, g) + k[i] + w[i];
    temp2 = sigma0(a) + majority(a, b, c);
    // perform mixing
    h = g;
    g = f;
    f = e;
    e = d + temp1;
    d = c;
    c = b;
    b = a;
    a = temp1 + temp2;
end
```



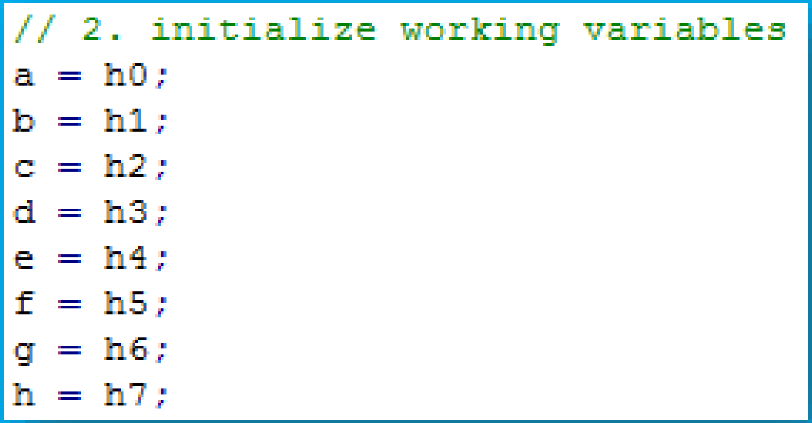
```
// compression functions
function [0:31] choose;
    input [0:31] e, f, g; // input 3 words
    // (e and f) xor ((not e) and g)
    choose = (e & f) ^ (~e & g);
endfunction

function [0:31] majority;
    input [0:31] a, b, c; // input 3 words
    // (a and b) xor (a and c) xor (b and c)
    majority = (a & b) ^ (a & c) ^ (b & c);
endfunction

function [0:31] sigma0;
    input [0:31] a; // 32-bit word to mix
    // (a ror 2) xor (a ror 13) xor (a ror 22)
    sigma0 = {a[30:31], a[0:29]} ^ {a[19:31], a[0:18]} ^ {a[10:31], a[0:9]};
endfunction

function [0:31] sigma1;
    input [0:31] e; // 32-bit word to mix
    // (e ror 6) xor (e ror 11) xor (e ror 25)
    sigma1 = {e[26:31], e[0:25]} ^ {e[21:31], e[0:20]} ^ {e[7:31], e[0:6]};
endfunction
```



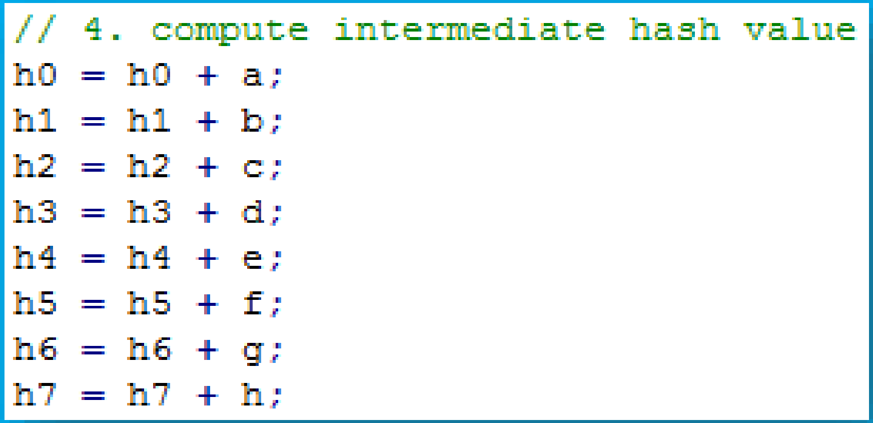
```
// 4. compute intermediate hash value
h0 = h0 + a;
h1 = h1 + b;
h2 = h2 + c;
h3 = h3 + d;
h4 = h4 + e;
h5 = h5 + f;
h6 = h6 + g;
h7 = h7 + h;
```



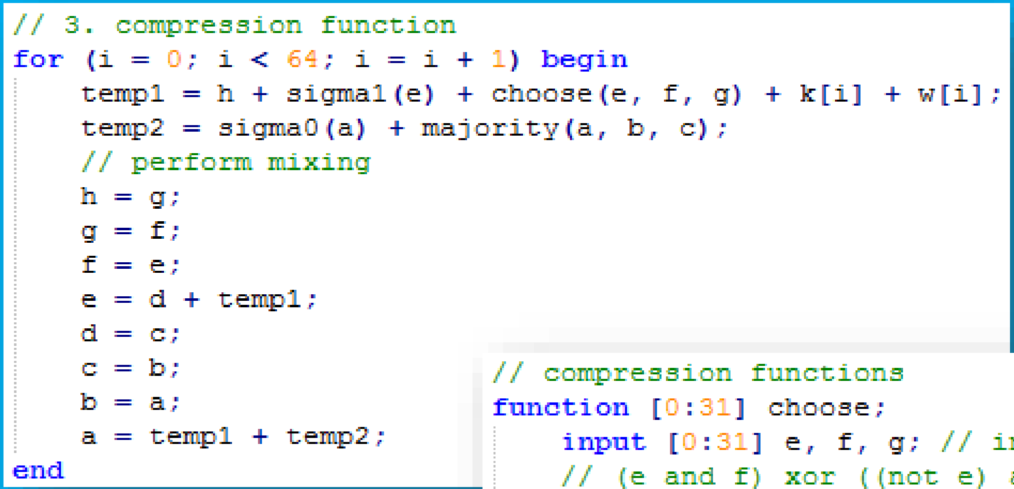
```
// 1. prepare the message schedule
w[0:15] = data; // copy chunk into first 16 words of the message schedule array
// extend the first 16 words into the remaining 48 words
for (i = 16; i < 64; i = i + 1) begin
    // (w[i-15] ror 7) xor (w[i-15] ror 18) xor (w[i-15] >> 3)
    s0 = {w[i-15][25:31], w[i-15][0:24]} ^ {w[i-15][14:31], w[i-15][0:13]} ^ w[i-15][0:28];
    // (w[i-2] ror 17) xor (w[i-2] ror 19) xor (w[i-2] >> 10)
    s1 = {w[i-2][15:31], w[i-2][0:14]} ^ {w[i-2][13:31], w[i-2][0:12]} ^ w[i-2][0:21];
    // store computed word
    w[i] = w[i-16] + s0 + w[i-7] + s1;
end
```



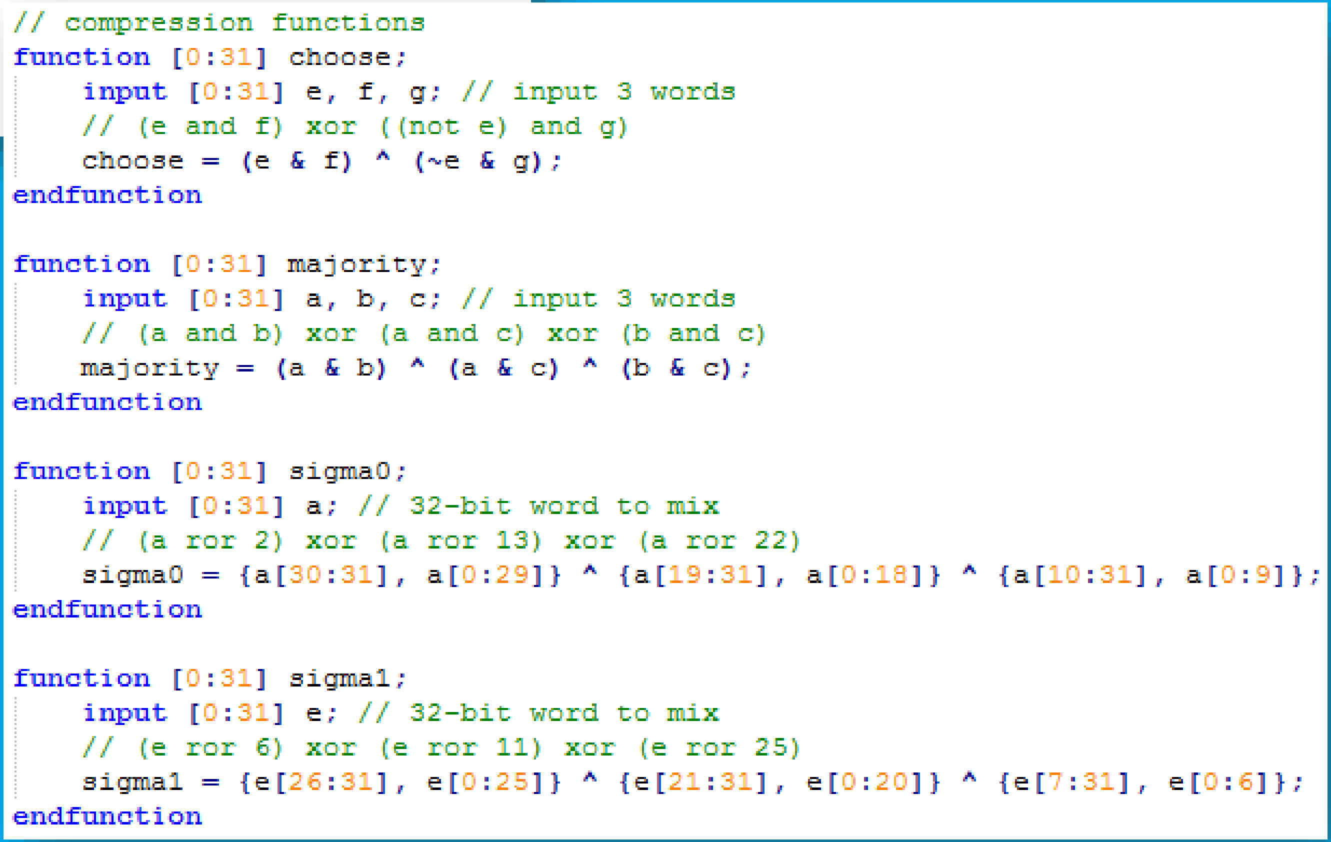
```
// 2. initialize working variables
a = h0;
b = h1;
c = h2;
d = h3;
e = h4;
f = h5;
g = h6;
h = h7;
```



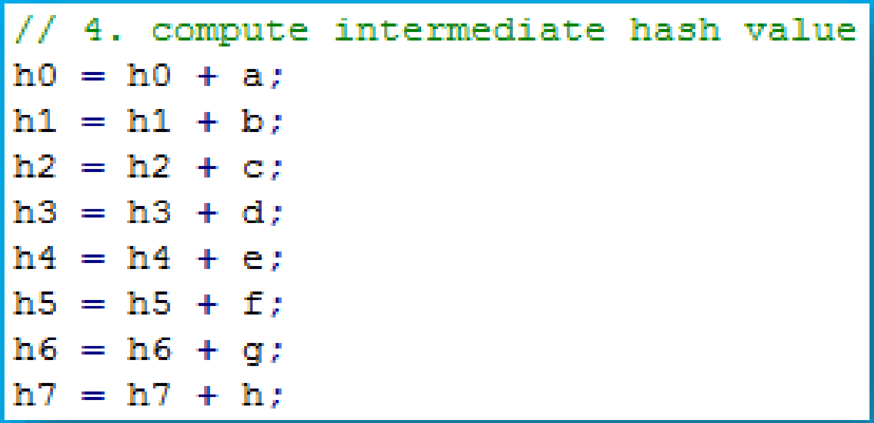
```
// 4. compute intermediate hash value
h0 = h0 + a;
h1 = h1 + b;
h2 = h2 + c;
h3 = h3 + d;
h4 = h4 + e;
h5 = h5 + f;
h6 = h6 + g;
h7 = h7 + h;
```



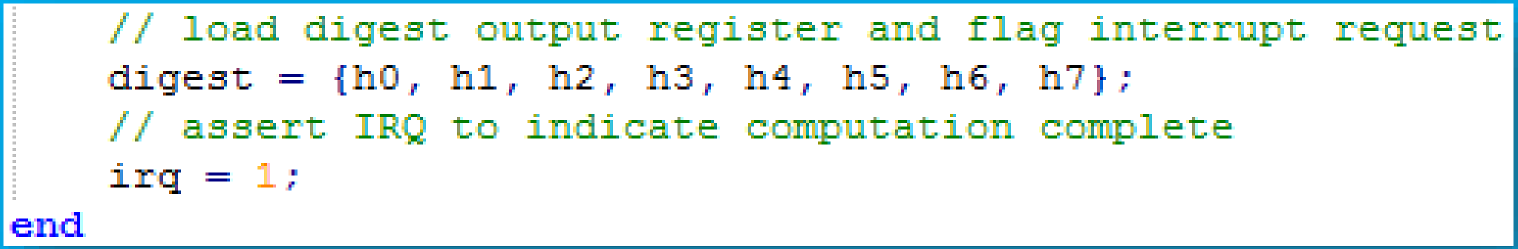
```
// 3. compression function
for (i = 0; i < 64; i = i + 1) begin
    temp1 = h + sigma1(e) + choose(e, f, g) + k[i] + w[i];
    temp2 = sigma0(a) + majority(a, b, c);
    // perform mixing
    h = g;
    g = f;
    f = e;
    e = d + temp1;
    d = c;
    c = b;
    b = a;
    a = temp1 + temp2;
end
```



```
// compression functions
function [0:31] choose;
    input [0:31] e, f, g; // input 3 words
    // (e and f) xor ((not e) and g)
    choose = (e & f) ^ (~e & g);
endfunction

function [0:31] majority;
    input [0:31] a, b, c; // input 3 words
    // (a and b) xor (a and c) xor (b and c)
    majority = (a & b) ^ (a & c) ^ (b & c);
endfunction

function [0:31] sigma0;
    input [0:31] a; // 32-bit word to mix
    // (a ror 2) xor (a ror 13) xor (a ror 22)
    sigma0 = {a[30:31], a[0:29]} ^ {a[19:31], a[0:18]} ^ {a[10:31], a[0:9]};
endfunction

function [0:31] sigma1;
    input [0:31] e; // 32-bit word to mix
    // (e ror 6) xor (e ror 11) xor (e ror 25)
    sigma1 = {e[26:31], e[0:25]} ^ {e[21:31], e[0:20]} ^ {e[7:31], e[0:6]};
endfunction
```



```
// 4. compute intermediate hash value
h0 = h0 + a;
h1 = h1 + b;
h2 = h2 + c;
h3 = h3 + d;
h4 = h4 + e;
h5 = h5 + f;
h6 = h6 + g;
h7 = h7 + h;
```



```
    // load digest output register and flag interrupt request
    digest = {h0, h1, h2, h3, h4, h5, h6, h7};
    // assert IRQ to indicate computation complete
    irq = 1;
end
```

EXCHANGES

coinbase.com/gdax.com

bittrex.com

binance.com

shapeshift.io

NEWS

coindesk.com

CURRENCIES

coinmarketcap.com

REDDIT

/r/cryptocurrency

/r/blockchain

/r/ethereum

/r/cryptomarkets