

Collisions for CubeHash1/45 and CubeHash2/89

Wei Dai

www.weidai.com

Abstract. Collisions were found for the hash functions CubeHash1/45-512 and CubeHash2/89-512. Attack code is included.

1 CubeHash

Bernstein's CubeHash[2] is a family of hash functions submitted to the NIST SHA-3 hash competition. The previous best attacks were collisions on CubeHash1/106 and CubeHash2/114, reported by Aumasson[1].

2 CubeHash1/45 Collision

2.1 message 1

```
9DD1994A9062ED74D877279D1AB1B38BD81FB5E4F1B1AB4067EF4E8165EAB8678173CC5F3E8CD39C
D51C6369350A3AE132AF2C98ACFAA1D159DB052BED5B3A884876501B2975BE48F95A1020D9550327
564B3D7C1A9FA16D8AE5F84F32DD478B91B3BDD838EF800F1AABC0AC4F3E7F3032E9E7CC73129E64
0F29102DF9080F6F2DB505CD2255EA5152C59D3BC5F136968F278C24E9E16F739D458EBC13935289
5807D36DD4512476C124D22DF3468BB05E794FFE26D815E6FB5B9610BC08AC1779BD0D27640CDBE
A4224D7B6ACD415DC136AB2632E1168298F74C365E68E19B70
```

2.2 message 2

```
9DD0994A9062ED74D876279D1AB1B38BD81FBDE4F1B1AB4067EF468165EAB8678173CC5F3E8CD39C
D51C6369350A3AE172AF2C98ACFBA1D199DB052BED5B3A804876501B2975BE30F95A1020D9758327
564B3D7C1ABF216C8AE5F84F32DD478B91B3BDD838EF800F1A2BC0AC4F3E7F3022E9E7CC73129E64
3F29102DF9082F6F2DB505CD22558A5652C51D3BD5F136968F270C24F9E16F739D458EBE13935289
5807D363DD4512476C124D02DF3468BB05E794DFE25D815E6FB5B9610BD08AC1779BD0D27640CDBE
A4224D7B6ACD415DC136AB2632E1168298F74C365E68E19B70
```

2.3 digest

```
962F9C2DFF26E65AFC257F5AA633CA2891A172CB38EED485BACD53956BF033031AD5DE4B9626CE10
831F81607DB106795182313E5B90034B6B7F5FA5CBBE5A70
```

2.4 notes

In an approximation of CubeHash where addition is replaced with XOR, an initial 4-bit difference at the least significant bits of (0-based) state bytes 1 and 9 and the fourth least significant bits of state bytes 18 and 26 eventually leads to zero differences if, after each round, all differences in the first 45 state bytes are eliminated using the next input block. In actual CubeHash, this occurs with high enough probability that a few seconds of unoptimized computation found a pair of colliding messages for CubeHash1/45.

3 CubeHash2/89 Collision

3.1 message 1

```
9608840653E4F8D2B2BADB3C7D7E6078A7A63845828B6273321345A2628DFC4C35DCF1067FD8AD02
36F1EED92316FA87DDD534B7F8C3ED427A1E7CF28EB1918B7DF8C60CD9857732FCA2D43EB13834F8
30EFADF45097E114F8A1B85D6E9DBB4A370C61DA2D17E6E5C88CC799462C496B178CC601DE6F3A1D
4DC2125CDB3C8BD3F6CAB2E6E01F95AB33F34B76B237FE614CF5E2F6D100D107D78A01B7A78AC1A4
E62DF6EFC89B3A7AD1639CBE0B5CD437D3E
```

3.2 message 2

```
9608840653E5F8D2B2BADB3C7D7F6078A7A63845828B6273321345A2628DFC4C35DCF1067FD8AD02
36F1EED92316FA87DDD534B7F8C3ED427A1E7CF28EB1918B7DF8C60CD9847732FCA2D43EB13934F8
30EFADF45097E114F8A1B85D6E9DAB8A370C61DA2D17F625CB8CC799462C496B178CC601DE6F3A1D
4DC2125CDB1C8BD3F6CAB2E6E07F95AB33F34B76BE37FE6148F5E2F6DD00D107D38A0177A78AC1A4
E62DF62FCE89B3A7AD1439CBE0B5CD437D3C
```

3.3 digest

```
FCF89436ADF177203C3DD16D2FB58C78DCAFC709113E043D55F35756B36C0652120CF7A2FD272BA2
6E85AEACFB6775DC54D1A7C8C7D651F4C7B0FF2DD0F29320
```

3.4 notes

A 4-bit difference at the least significant bits of (0-based) state bytes 5, 13, 69, and 77 has a good chance of causing zero differences in state bytes 89 to 127 after two rounds, thus allowing an attack on CubeHash2/b with $b \geq 89$.

References

1. Jean-Philippe Aumasson, Willi Meier, María Naya-Plasencia, and Thomas Peyrin. Inside the hypercube. <http://eprint.iacr.org/2008/486>, 2008.
2. Daniel J. Bernstein. Cubehash specification (2.b.1). <http://cubehash.cr.yp.to/submission/spec.pdf>, 2008.

A Attack code

This attack code is meant to be used with the “Optimized” implementation of CubeHash, also known as simple.h and simple.c.

```
int main() {
    BitSequence data1[5*CUBEHASHBLOCKBYTES], data2[5*CUBEHASHBLOCKBYTES];
    BitSequence hash1[64], hash2[64];
    hashState state1, state2;
    unsigned int diff[128];

    int i;
    #if CUBEHASHROUNDS == 1
    do
    {
        for (i=0; i<CUBEHASHBLOCKBYTES; i++)
            data2[i] = data1[i] = rand()%256;

        data2[0*4+1] ^= 0x1 << 0;
        data2[2*4+1] ^= 0x1 << 0;
        data2[4*4+2] ^= 0x1L << 3;
        data2[6*4+2] ^= 0x1L << 3;

        Init(&state1, 512);
        Init(&state2, 512);

        Update(&state1, data1, CUBEHASHBLOCKBYTES*8);
        Update(&state2, data2, CUBEHASHBLOCKBYTES*8);

        for (i=0; i<32; i++)
            diff[i] = state1.x[i] ^ state2.x[i];

        for (i=0; i<CUBEHASHBLOCKBYTES; i++)
        {
            data2[i + CUBEHASHBLOCKBYTES] = data1[i +
                CUBEHASHBLOCKBYTES] = rand()%256;
            data1[i + CUBEHASHBLOCKBYTES] ^= ((unsigned char *)
                state1.x)[i];
            data2[i + CUBEHASHBLOCKBYTES] ^= ((unsigned char *)
                state2.x)[i];
        }

        Update(&state1, data1 + CUBEHASHBLOCKBYTES, CUBEHASHBLOCKBYTES*8);
        Update(&state2, data2 + CUBEHASHBLOCKBYTES, CUBEHASHBLOCKBYTES*8);

        for (i=0; i<32; i++)
            diff[i] = state1.x[i] ^ state2.x[i];

        for (i=0; i<CUBEHASHBLOCKBYTES; i++)
        {
            data2[i + 2*CUBEHASHBLOCKBYTES] = data1[i + 2*
                CUBEHASHBLOCKBYTES] = rand()%256;
            data1[i + 2*CUBEHASHBLOCKBYTES] ^= ((unsigned char *)
                state1.x)[i];
            data2[i + 2*CUBEHASHBLOCKBYTES] ^= ((unsigned char *)
                state2.x)[i];
        }

        Update(&state1, data1 + 2*CUBEHASHBLOCKBYTES, CUBEHASHBLOCKBYTES*8);
        Update(&state2, data2 + 2*CUBEHASHBLOCKBYTES, CUBEHASHBLOCKBYTES*8);

        for (i=0; i<32; i++)
            diff[i] = state1.x[i] ^ state2.x[i];
    }
}
```

```

    for (i=0; i<CUBEHASHBLOCKBYTES; i++)
    {
        data2[i + 3*CUBEHASHBLOCKBYTES] = data1[i + 3*
            CUBEHASHBLOCKBYTES] = rand()%256;
        data1[i + 3*CUBEHASHBLOCKBYTES] ^= ((unsigned char *)
            state1.x)[i];
        data2[i + 3*CUBEHASHBLOCKBYTES] ^= ((unsigned char *)
            state2.x)[i];
    }

    Update(&state1, data1 + 3*CUBEHASHBLOCKBYTES, CUBEHASHBLOCKBYTES*8);
    Update(&state2, data2 + 3*CUBEHASHBLOCKBYTES, CUBEHASHBLOCKBYTES*8);

    for (i=0; i<32; i++)
        diff[i] = state1.x[i] ^ state2.x[i];
}
while (memcmp(((unsigned char *)state1.x)+CUBEHASHBLOCKBYTES, ((
    unsigned char *)state2.x)+CUBEHASHBLOCKBYTES, 128-
    CUBEHASHBLOCKBYTES) != 0);

memcpy(data1 + 4*CUBEHASHBLOCKBYTES, state1.x, CUBEHASHBLOCKBYTES);
memcpy(data2 + 4*CUBEHASHBLOCKBYTES, state2.x, CUBEHASHBLOCKBYTES);

printf("\nmessage 1:\n\n");
for (i=0; i<CUBEHASHBLOCKBYTES*5; ++i)
    printf("%02X", data1[i]);
printf("\n\nmessage 2:\n\n");
for (i=0; i<CUBEHASHBLOCKBYTES*5; ++i)
    printf("%02X", data2[i]);
printf("\n\n");

Hash( 512, data1, CUBEHASHBLOCKBYTES*5*8, hash1 );
Hash( 512, data2, CUBEHASHBLOCKBYTES*5*8, hash2 );

printf("\ndigest:\n\n");

for (i=0; i<64; ++i)
    printf("%02X", hash1[i]);
printf("\n\n");
for (i=0; i<64; ++i)
    printf("%02X", hash2[i]);
printf("\n\n");

for (i=0; i<64; ++i)
    if (hash1[i] != hash2[i]) {
        printf("\ndifferent digests!\n\n");
    }
return 0;
}
printf("\nsame digests!\n\n");
#endif

#ifdef CUBEHASHROUNDS == 2
do
{
    for (i=0; i<CUBEHASHBLOCKBYTES; i++)
        data2[i] = data1[i] = rand()%256;

    data2[1*4+1] ^= 0x1;
    data2[3*4+1] ^= 0x1;
    data2[(16+1)*4+1] ^= 0x1;
    data2[(16+3)*4+1] ^= 0x1;

    Init(&state1, 512);
    Init(&state2, 512);

    Update(&state1, data1, CUBEHASHBLOCKBYTES*8);

```

```

Update(&state2 , data2 , CUBEHASHBLOCKBYTES*8);

for (i=0; i<32; i++)
    diff[i] = state1.x[i] ^ state2.x[i];
}
while (memcmp(((unsigned char *)state1.x)+CUBEHASHBLOCKBYTES, ((
    unsigned char *)state2.x)+CUBEHASHBLOCKBYTES, 128-
    CUBEHASHBLOCKBYTES) != 0);

memcpy(data1 + CUBEHASHBLOCKBYTES, state1.x, CUBEHASHBLOCKBYTES);
memcpy(data2 + CUBEHASHBLOCKBYTES, state2.x, CUBEHASHBLOCKBYTES);

printf("\nmessage 1:\n\n");
for (i=0; i<CUBEHASHBLOCKBYTES*2; ++i)
    printf("%02X", data1[i]);
printf("\n\nmessage 2:\n\n");
for (i=0; i<CUBEHASHBLOCKBYTES*2; ++i)
    printf("%02X", data2[i]);
printf("\n\n");

Hash( 512, data1, CUBEHASHBLOCKBYTES*2*8, hash1 );
Hash( 512, data2, CUBEHASHBLOCKBYTES*2*8, hash2 );

printf("\ndigest:\n\n");

for (i=0; i<64; ++i)
    printf("%02X", hash1[i]);
printf("\n\n");
for (i=0; i<64; ++i)
    printf("%02X", hash2[i]);
printf("\n\n");

for (i=0; i<64; ++i)
    if (hash1[i] != hash2[i]){
        printf("\ndifferent digests!\n\n");
    }
return 0;
}
printf("\nsame digests!\n\n");
#endif

return 0;
}

```