Collisions for Schnorr's Hash Function FFT-Hash Presented at Crypto '91

Joan Daemen, Antoon Bosselaers,
René Govaerts and Joos Vandewalle

Katholieke Universiteit Leuven, Laboratorium ESAT,
Kardinaal Mercierlaan 94, B–3001 Heverlee, Belgium.

Abstract

A method is described to generate collisions for the hash function FFT-Hash that was presented by Claus Schnorr at Crypto '91. A set of colliding messages is given that was obtained by this method.

1 Introduction

In the Rump Session of Crypto '91 Claus Schnorr presented FFT-Hash. This is a function that hashes messages of arbitrary length into a 128 bit hash value. It consists of two rounds, where every round is the combination of a Fast Fourier Transform over GF(2^{16}+1) and a nonlinear recursion. It was claimed that producing a pair of messages that yield the same hashvalue is computationally infeasible. We have written a program that outputs a set of 384 bit messages that all have the same hash value for FFT-Hash. The CPU-time consumed is of the order of a few hours. An optimized version of the program is expected to take only a few minutes on a modern PC. The first collision was produced on October 3rd '91.

2 Description of FFT-Hash

Padding: The message is padded with a single "1" followed by a suitable number of "0" bits followed by the binary representation of its original length. The padded message can then be seen as the concatenation of a number of 128-bit blocks: $M_0 \parallel M_1 \ldots \parallel M_{n-1}$.

Appeared in *Advances in Cryptology – ASIACRYPT 1991*, Lecture Notes in Computer Science 739, H. Imai, T. Matsumoto, and R. L. Rivest (eds.), Springer-Verlag, pp. 477–480, 1993.

Algorithm for the hash function h**:** $H_i = g(H_{i-1} \parallel M_{i-1})$ for $i = 1, \ldots, n$. $H_i \in \{0,1\}^{128}$ and initial value H_0 = 0123 4567 89ab cdef fedc ba98 7654 3210 (hex.). The output of $h(M) = H_n$.
Algorithm for the function g**:** Let $p = 2^{16} + 1$. The input to g is split up into 16 components $(e_0, \ldots, e_{15})$ with each component e_i consuming 16 bits. These e_i are treated as representations of integers modulo p. Define the FFT-transformation $FT_8(a_0, \ldots, a_7) = (b_0, \ldots, b_7)$ as

$$b_i = \sum_{j=0}^{7} 2^{4ij} a_j \bmod p \quad \text{for} \quad i = 0, \ldots, 7 \tag{1}$$

1. $(e_0, e_2, \ldots, e_{14}) = FT_8(e_0, e_2, \ldots, e_{14})$ This step is called a *FFT-step*
2. `FOR( i=0 ; i<16 ; i++ )` $e_i = e_i + e_{i-1}e_{i-2} + e_{e_{i-3}} + 2^i \bmod p$
   All indices are taken modulo 16. This step is called a *recursion step.*
3. Second round: repeat step 1 and 2

The output of g is the 128-bit string $e_8 \parallel e_9 \ldots \parallel e_{15}$ where all occurrences of $p - 1 = 2^{16}$ are substituted by 0.

3 Weaknesses of FFT-Hash

1. The FFT step only affects the components with even index. For odd-indexed components no diffusion takes place.
2. The linearity of the FFT step can be used to impose certain values upon a number of output components. If for certain subsets of no more than 8 components, belonging to either the output or the input, the values are fixed, values for the remaining components can be computed such that equation 1 holds. This computation involves linear algebra alone.
3. The diffusion resulting from the recursion step can be completely eliminated by imposing 0 values to certain components. Suppose $(e_0, \ldots, e_{15})$ is the 16-tuple that has just undergone a recursion step. Suppose $e_5 = e_7 = 0$. Suppose also that e_6 was never addressed in the indirect indexing term $e_{e_{i-3}}$, hence $e_{i-3} \neq 6 \pmod{16}$ for all i at the moment they are used. Then the 12 MSB bits of e_6 only appear in the calculation for the new value of e_6. This can easily be seen because when $e_5 = e_7 = 0$ a product term $e_{i-1}e_{i-2}$ containing e_6 must be zero. Because the 12 MSB bits of e_6 can be altered without affecting the outcome of other components when the recursion is applied, e_6 will be called *isolated.* This can be applied to any component. Hence isolation of a component in a recursion step requires that the

Appeared in *Advances in Cryptology – ASIACRYPT 1991*, Lecture Notes in Computer Science 739, H. Imai, T. Matsumoto, and R. L. Rivest (eds.), Springer-Verlag, pp. 477–480, 1993.

two neighboring components are 0 *and* that it is not addressed in the term $e_{e_{i-3}}$ for any i.

4 The Attack

The attack is based on the fact that it is possible to isolate a component during all four steps of g. The colliding messages consist of 3 blocks: M_0, M_1 and M_2. All effort goes into the search for appropriate M_1 and M_2 values. The attack is probabilistic. A subset of messagebits are given random values thereby fixing the remaining bits through a number of imposed relations. Starting from $H_1 = g(H_0 \parallel M_0)$ we have:

1. Calculation of M_1. The values are chosen in a way that the second component of M_1 ($= e_9$) has a maximum probability of staying isolated throughout the calculation of g. Certain changes in the 12 MSB bits of this component affect the intermediate hash value H_2 only in the second component. On the average 2^{23} different H_1, obtained by trying different M_0 values have to be tested. Only about 2^{11} of these survive a first check. For each of these remaining M_1 values 2^{15} trials have to be performed by varying ϕ (see figure).

2. Calculation of M_2. The values of M_2 are chosen in such a way that the second component of H_2 ($= e_1$) has maximum probability of being isolated and thus does not affect H_3. About 2^{22} different values of ϕ_1 and ϕ_2 have to be tried.

The figure illustrates the internal relations during the hashing process of the colliding messages. Q indicates the component that is isolated throughout the whole calculation.

The first result obtained by this method was a set of 805 colliding messages (in hexadecimal notation)

```
00a1 0000 0000 0000 0000 0000 000c 5b18
9156 XXXd 9e89 67e8 35f8 e2b0 12ec 26c0
570b 06ee ba21 8da5 6ec4 c27e 5d5d e6be
```

where `XXX` ranges over `1b5` to `4d9` that all hash to

```
527d c019 d8cb 1d92 162b f04c cfff 26c6
```

References

[1] C Schnorr, FFT-Hash, An Efficient Cryptographic Hash Function,*Rump Session Crypto '91.*

Appeared in *Advances in Cryptology – ASIACRYPT 1991*, Lecture Notes in Computer Science 739, H. Imai, T. Matsumoto, and R. L. Rivest (eds.), Springer-Verlag, pp. 477–480, 1993.

An arrow from e_i to e_j means $e_{e_i} = e_j$ or $e_i = j \pmod{16}$

Boxes containing a constant indicate the value that is imposed upon the component

Boxes containing a greek letter indicate variables that are isolated (denoted by ■) until t used (as indicated in the down left corner) to impose a certain value to a component

A $\star$ in the down left corner indicates that we depend upon luck (prob: 2^{-16})

□ indicates that the component is fixed by an FFT relation

An empty box denotes a component that is fixed by initial values and/or internal relatio

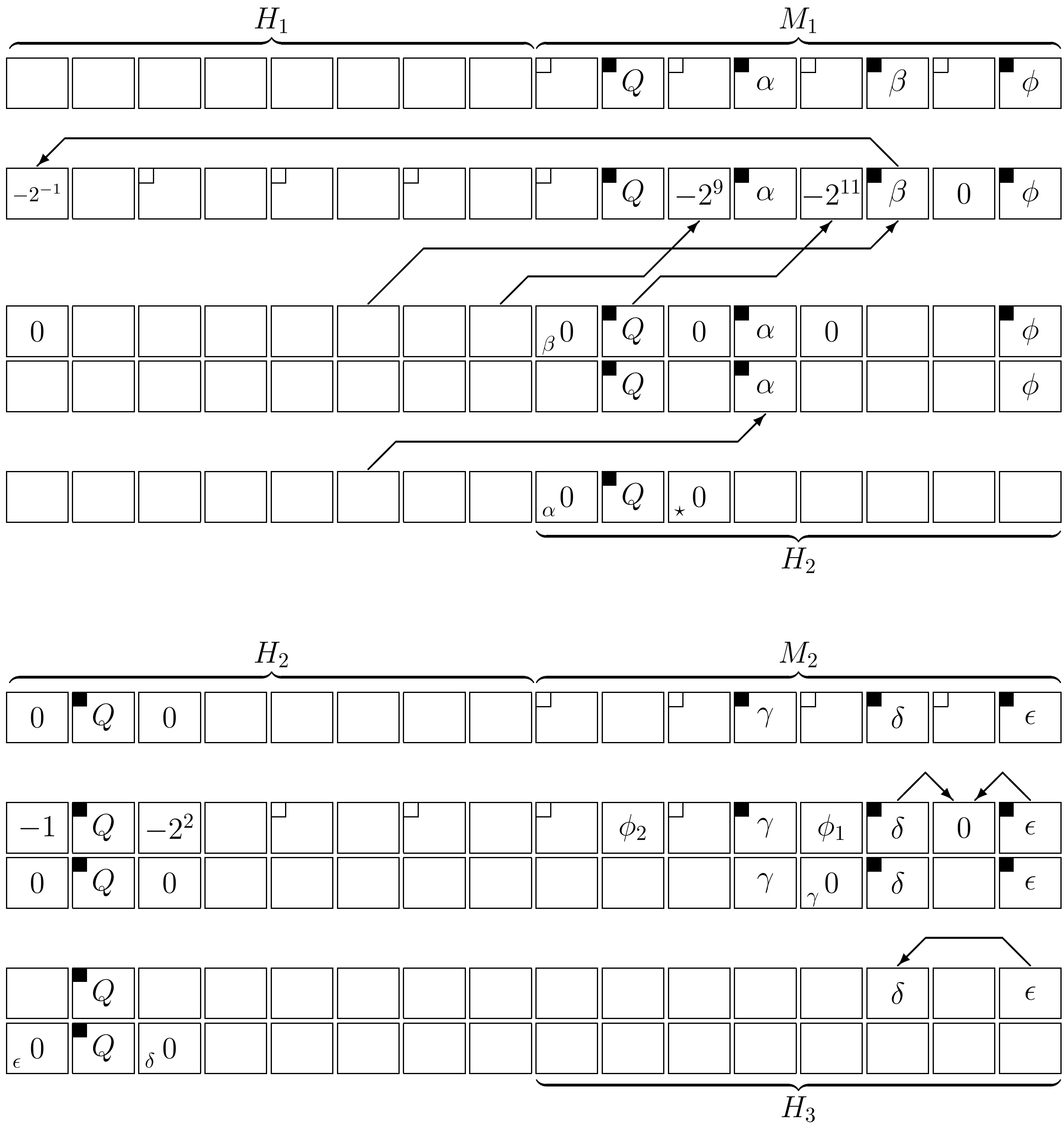

Figure 1: Schematic overview of the collisions of FFT-Hash. The state $(e_0, \ldots, e_{15})$ is depicted before and after every step.

Appeared in *Advances in Cryptology – ASIACRYPT 1991*, Lecture Notes in Computer Science 739, H. Imai, T. Matsumoto, and R. L. Rivest (eds.), Springer-Verlag, pp. 477–480, 1993.