# Second preimage attack on *MeshHash*

Søren S. Thomsen
crypto@znoren.dk

December 1, 2008

## 1 Introduction

We describe a second preimage attack on the SHA-3 candidate *MeshHash*. MeshHash was designed by Björn Fay. For a detailed description of the MeshHash hash function, we refer to the specification [1]. Here, we use the same notation as in the specification.

We briefly describe one of the building blocks of MeshHash.

### 1.1 The normal round function

Consider MeshHash-$n$, which is the version of MeshHash that produces a message digest of $n$ bits. MeshHash-$n$ operates with a state consisting of $P$ pipes called $\texttt{pipe}[\texttt{i}]$, $\texttt{i} = 0, \ldots, P-1$, where $P = \lfloor n/64 \rfloor + 1$. Each pipe is a 64-bit word. A 64-bit message block $\texttt{data}$ updates the $P$ pipes via the *normal round function*. This function does the following (for $\texttt{i} = 0, \ldots, P-1$):

$$\texttt{pipe}[\texttt{i}] \leftarrow \mathsf{SBox}(\mathsf{RotR}^{37i}(\texttt{pipe}'[\texttt{i}] \oplus (i \boxdot \mathsf{0101010101010101}_h) \oplus \texttt{data})) \boxplus \texttt{pipe}'[\texttt{i} + 1 \bmod P].$$

$\texttt{pipe}'[\texttt{i}]$ denotes the original values of the pipes, before the normal round function is applied to any of them. $\mathsf{SBox}$ is a 64-bit s-box computed as described below. $\mathsf{RotR}^x$ means right-rotation by $x$ positions. The symbol '$\boxdot$' denotes multiplication modulo $2^{64}$, and '$\boxplus$' denotes addition modulo $2^{64}$. 64-bit constants are written in hexadecimal.

It is seen from the normal round function that each pipe is a function of the previous value of the pipe, one other pipe, and the message block.

The s-box is defined as follows. On input $x$, carry out the following sequence of computations:

$$\begin{aligned}
x &\leftarrow x \boxdot \mathsf{9e3779b97f4a7bb9}_h \\
x &\leftarrow x \boxplus \mathsf{5e2d58d8b3bcdef7}_h \\
x &\leftarrow \mathsf{RotR}^{37}(x) \\
x &\leftarrow x \boxdot \mathsf{9e3779b97f4a7bb9}_h \\
x &\leftarrow x \boxplus \mathsf{5e2d58d8b3bcdef7}_h \\
x &\leftarrow \mathsf{RotR}^{37}(x).
\end{aligned}$$

For completeness, we state how the inverse s-box, $\mathsf{SBox}^{-1}$, may be computed.

$$\begin{aligned}
x &\leftarrow \mathsf{RotR}^{27}(x) \\
x &\leftarrow x \boxminus \mathsf{5e2d58d8b3bcdef7}_h \\
x &\leftarrow x \boxdot \mathsf{693622400cab1a89}_h \\
x &\leftarrow \mathsf{RotR}^{27}(x) \\
x &\leftarrow x \boxminus \mathsf{5e2d58d8b3bcdef7}_h \\
x &\leftarrow x \boxdot \mathsf{693622400cab1a89}_h.
\end{aligned}$$

Here, '$\boxminus$' denotes subtraction modulo $2^{64}$.

We have not mentioned the "final block round", which is applied for every $P$ normal rounds. In this function, each pipe is updated using a message block counter. The function is efficiently invertible, and makes no difference to the attacks described here.

# 2 Second preimage attack

The normal round function can be inverted in time about $2^{64}$. This leads to a second preimage attack using the meet-in-the-middle method.

## 2.1 Inverting the normal round function

Given the $P$ pipes $\texttt{pipe}[\texttt{i}]$ and a message block $\texttt{data}$, the original values $\texttt{pipe}'[\texttt{i}]$ of the pipes, which are mapped to $\texttt{pipe}[\texttt{i}]$ by the message block $\texttt{data}$, may be found in time about $2^{64}$ as follows.

1. Choose an arbitrary 64-bit value of a variable $\texttt{p0}$.
2. Compute $\texttt{pipe}'[P-1] \leftarrow \mathsf{RotR}^{27(P-1)}(\mathsf{SBox}^{-1}(\texttt{pipe}[P-1] \boxminus \texttt{p0})) \oplus \texttt{data} \oplus ((P-1) \boxdot \mathsf{0101010101010101}_h)$.
3. Compute, for $i$ from $P-2$ down to 0,

   $$\texttt{pipe}'[\texttt{i}] \leftarrow \mathsf{RotR}^{27i}(\mathsf{SBox}^{-1}(\texttt{pipe}[\texttt{i}] \boxminus \texttt{pipe}[\texttt{i}+1 \bmod 5])) \oplus \texttt{data} \oplus (i \boxdot \mathsf{0101010101010101}_h)$$

4. If $\texttt{p0} = \texttt{pipe}'[0]$, then the normal round function has been successfully inverted. Otherwise, start over.

The probability that $\texttt{p0} = \texttt{pipe}'[0]$ for an arbitrary value of $\texttt{p0}$ is estimated to be about $2^{-64}$. Hence, the expected complexity of inverting the normal round function is $2^{64}$. This is independent of the number $P$ of pipes.

## 2.2 Meet-in-the-middle attack

Consider MeshHash-$n$, where $n$ is a multiple of 64. Assume we are given a message $M$ of at least $P$ blocks, and we want to find a second preimage of the hash value $H(M)$. We compute the intermediate hash values when processing $M$. We then carry out a meet-in-the-middle attack based on the inversion algorithm described in the previous section. We invert the normal round function starting from one of the last intermediate hash values when processing $M$, using $2^{n/2}$ different $\lceil n/128 \rceil$-block messages. We need messages of length at least $n/128$ blocks in order to have enough degrees of freedom, and therefore the total complexity is about $(n/128) \times 2^{n/2+64}$.

We also partially hash $2^{n/2+64}$ messages in the forward direction, each message being at least $n/128+1$ blocks in length. The length must be chosen such that the final message has the same length as $M$. This takes time about $(n/128+1) \times 2^{n/2+64}$.

The $2^{n/2+64}$ intermediate hash values thus produced may match any of the $2^{n/2}$ intermediate hash values computed by inverting the normal round function. Since there are $2^{n/2+64} \times 2^{n/2} = 2^{n+64}$ pairs of intermediate hash values that may match in $P \times 64 = n + 64$ bits, we expect to find a second preimage. The total time complexity is about $(n/64+1) \times 2^{n/2+64}$. With, e.g., $n = 256$, the complexity is about $2^{194.3}$, well below the claimed second preimage security level of $2^{256}$, and also (for all practical message lengths) well below the required complexity of $2^{256-k}$ to find a second preimage matching a first preimage of $2^k$ blocks. Other complexities can be

Table 1: Complexities of the second preimage attack for various output sizes $n$ (claimed resistance is $2^n$).

| $n$ | Second preimage complexity |
|---|---|
| 256 | $2^{194.3}$ |
| 320 | $2^{226.6}$ |
| 384 | $2^{258.8}$ |
| 448 | $2^{291.0}$ |
| 512 | $2^{323.2}$ |

found in Table 1. We note that memory requirements are about $2^{n/2}$. Memoryless variants of the meet-in-the-middle attack exist [2, 3]; it is unclear, however, what the effect in terms of running time of the attack would be.

### 2.3 Preimage attacks

We have not found a method of producing preimages in MeshHash. The reason is that the function producing the output of MeshHash does not seem to be easily invertible. However, if a method of inverting the output function is found, then the above attack can be applied directly.

## References


[1] B. Fay. MeshHash. SHA-3 Algorithm Submission. Available: `http://ehash.iaik.tugraz.at/uploads/5/5a/Specification_DIN-A4.pdf` (2008/12/01).

[2] H. Morita, K. Ohta, and S. Miyaguchi. A Switching Closure Test to Analyze Cryptosystems. In J. Feigenbaum, editor, *Advances in Cryptology – CRYPTO '91, Proceedings*, volume 576 of *Lecture Notes in Computer Science*, pages 183–193. Springer, 1992.

[3] J.-J. Quisquater and J.-P. Delescaille. How Easy is Collision Search. New Results and Applications to DES. In G. Brassard, editor, *Advances in Cryptology – CRYPTO '89, Proceedings*, volume 435 of *Lecture Notes in Computer Science*, pages 408–413. Springer, 1990.