# Crypto Engineering (GBX9SY03)
# TP — Generic second preimage attacks on long messages for narrow-pipe Merkle-Damgård hash functions

Created by Pierre Karpman

2025-11-05

## Grading

This TP is graded as part of the *contrôle continu*. You must send a written report (in a portable format) detailing your answers to the questions, and the corresponding source code, *including all tests*, **with compilation and execution instructions** by 2025-11-19T18:00+0100) to:

tom.briand@univ-grenoble-alpes.fr.

Working in teams of two is allowed (but not mandatory), in which case only one report needs to be sent, with the name of both students clearly mentioned.

## Introduction

The goal of this TP is to implement a generic second preimage attack for long messages for Merkle-Damgård hash functions, in the specific case where the compression function used within the hash function follows a so-called *Davies-Meyer* (henceforth, "DM") construction. This is a particular case of the attacks (re-)designed by Kelsey and Schneier [KS05].

You will have to implement and run the full attack on a toy hash function with 48-bit hashes based on the Speck48/96 block cipher [BSS+13]. While this hash function will obviously be vulnerable to brute-force attacks (and maybe even more), it is simple to implement and it reasonably behaves like a random function, which is all we need here.

## Part one: preparatory work

Download the specifications of Speck48/96 at http://eprint.iacr.org/2013/404 and the tarball at https://membres-ljk.imag.fr/Pierre.Karpman/cry_eng2023_tp_second_preim.tar.bz2. This tarball contains the file `second_preim_48_fillme.c` that already partially implements some functions, which you will be required to complete.

### Question 1

Finish to implement the function `speck48_96` of the encryption with SPECK48/96, and verify the correctness of your implementation with the test values provided in [BSS+13, App. C], by using the (already implemented) function `test_vector_okay`.

### Question 2

Implement the function `speck48_96_inv` of the decryption by SPECK48/96. Write a test function `int test_sp48_inv(void)` that verifies that `speck48_96_inv` and `speck48_96` are inverses of each other.

### Question 3

Implement the function `cs48_dm` that transforms SPECK48/96 into a compression function using a DM construction with an XOR feedforward, i.e. defining $\mathcal{F}(h, m)$ as $\mathcal{E}(m, h) \oplus h$. Note that for later convenience in the attack functions, the 48-bit input and output of this function is now stored in the low bits of a single 64-bit word. Write a test function `int test_cs48_dm(void)` to check that the result of calling the compression function with IV `0x010203040506ULL` and message `{0,1,2,3}` is `0x5DFD97183F91ULL`.

### Question 4

Implement the function `get_cs48_dm_fp` that returns the unique fixed-point `fp` of the function `cs48_dm` for the message `m`; that is, compute the unique 48-bit string `fp` such that `cs48_dm(m,fp) == fp`. Write a test function `int test_cs48_dm_fp(void)`.

N.B.: Such a fixed-point is very easy to compute for a DM compression function: all one needs is $h$ such that $\mathcal{E}(m, h) \oplus h = h$.

## Part two: the attack

The main idea of the second-preimage attack that we consider here is that given a (long) target message, one searches for collisions with one of its intermediate chaining values.

Given an $\ell$-block message $M = m_1 \| \dots \| m_\ell$, the computation of $\mathcal{H}(M)$ with a narrow-pipe Merkle-Damgård hash function $\mathcal{H}$ with compression function $\mathcal{F}$ requires the computation of $\ell$ chaining values $h_i := \mathcal{F}(h_{i-1}, m_i)$ (with $h_0$ being the fixed IV of the function). Assuming first a simplistic function $\mathcal{H}$ that does not use any sort of padding, one can see that an attacker who has found $M'$ (with a size multiple of the block length) s.t. $\mathcal{H}(M') = h_i$ for some $0 \leqslant i \leqslant \ell$ may form the message $M'' = M' \| m_{i+1} \| \dots \| m_\ell$ leading to $h_\ell = \mathcal{H}(M)$, and thus found a second preimage $M''$ for $M$. The expected number of tries for finding a collision between an $n$-bit chaining value with $\ell$ targets being $2^{n - \log(\ell)}$, this attack is better than a generic search (but still costs at least $2^{n/2}$ compression function calls).

In reality, $\mathcal{H}$ will most likely use some form of padding that includes the length of the message being hashed.* If $M$ and its potential second preimage $M''$ are not of the same length, their padding (added after their last message block and before the computation of

*For instance to prevent collisions from DM fixed points.

the hash) will be distinct and thus lead to different hashes with overwhelming probability, thus invalidating the attack.

One way to go around this problem is to first compute an *expandable message* E that will allow to "lengthen" $M''$ up to the boundary of the block i while preserving the collision with $h_i$. One may then build $M''' = E||M''$ which is of the same length as M and thus has the same padding, resulting in a true second-preimage.

You are required to implement this attack for the hash function `hs48`, already implemented in the provided file. This function takes messages made of an integral number of 96-bit *blocks*, each represented as an array `uint32_t b[4]` where only the 24 lowest bits of each 32-bit word are set.

## Question 1

Implement the function `find_exp_mess`. This function must return two one-block messages `m1` and `m2` such that there exists a value `h` equal to both `cs48_dm(m1, IV)` and `get_cs48_dm_fp(m2)`. In other words, for any message `m` made of one copy of `m1` followed by `n >= 1` copies of `m2`, one has `hs48(m,n+1,0,0) == h`. Write a test function `int test_em(void)` to validate your function.

Give a detailed description of your implementation of `find_exp_mess` (for instance describing your choices for the data structures) and evaluate its performance (both theoretically and practically).

**How to proceed.** To answer this question, you are advised to perform a *meet-in-the-middle* search: compute N possible chaining values for N random first-block messages `m1`, and then compute fixed-points for random messages until one of them collides with a previously obtained chaining value. You may use the pseudo-random number generator provided in the header file `xoshiro.h`.[†]

Don't forget to enable the optimisations of your compiler (for instance using flags `-O3 -march=native`). With a reasonably good implementation, finding an expandable message should take significantly less than one minute on average.

## Question 2

Implement the function `attack`. This function must return a second preimage for the message `mess` of $2^{18}$ blocks generated by the following code:

```
for (int i = 0; i < (1 << 20); i+=4)
{
        mess[i + 0] = i;
        mess[i + 1] = 0;
        mess[i + 2] = 0;
        mess[i + 3] = 0;
}
```

whose hash is equal to `0x7CA651E182DBULL`.

[†]See also http://xoshiro.di.unimi.it/.

Give a detailed description of your implementation and an analysis of its performance (both theoretical and practical). Be careful to document enough the output of this function so that it provides all the information needed to characterise the second preimage (you may for instance freely use the "verbose" option of `hs48`).

**How to proceed.** You should first compute an expandable message with associated fixed-point `fp`, and then search for a collision block `cm` s.t. `cs48_dm(cm,fp)` is equal to one of the chaining values of `mess`. Once such a block is found, you need to form the second preimage message `mess2` by expanding the expandable message to an appropriate number of blocks and suffixing the colliding block and the remaining blocks identical to the ones of `mess`. Finally compute the hash of `mess2` to validate your attack.

With a reasonably good implementation, finding a full attack should take significantly less than ten minutes on average.

## References

[BSS+13] Ray Beaulieu, Douglas Shors, Jason Smith, Stefan Treatman-Clark, Bryan Weeks, and Louis Wingers, *The SIMON and SPECK Families of Lightweight Block Ciphers*, IACR Cryptology ePrint Archive **2013** (2013), 404.

[KS05] John Kelsey and Bruce Schneier, *Second Preimages on n-Bit Hash Functions for Much Less than $2^n$ Work*, Advances in Cryptology — EUROCRYPT 2005 (Ronald Cramer, ed.), Lecture Notes in Computer Science, vol. 3494, Springer, 2005, pp. 474–490.