Introduction to Cryptology (GBIN8U16)
TP — Multicollisions for narrow-pipe Merkle-Damgård hash functions

2022-03/04

Grading

This TP is graded as the *contrôle continu* of this course. 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 the end of April, (2022-04-29T18:00+0200) to:

pierre.karpman@univ-grenoble-alpes.fr.

Working in teams of two is allowed but not mandatory. In that case only a single report must be sent, with the two team members clearly identified.

1 Description of the attack

Let $\mathcal{H} : \{0,1\}^* \rightarrow \{0,1\}^n$ be a *narrow-pipe Merkle-Damgård hash function* based on a compression function $\mathcal{F} : \{0,1\}^b \times \{0,1\}^n \rightarrow \{0,1\}^n$ (where we assume for simplicity that $b > n/2$), a padding π and using an IV ι, meaning that if $\pi(m)$ writes $m_1||m_2||\cdots||m_\ell$ with all the m_i's in $\{0,1\}^b$, define $h_1 := \mathcal{F}(m_1, \iota)$, $h_i := \mathcal{F}(m_i, h_{i-1})$ for $1 < i \leq \ell$, and $\mathcal{H}(m) := h_\ell$.

One may then observe that if $m_1^{(0)}, m_1^{(1)}, \ldots, m_d^{(0)}, m_d^{(1)} \in \{0,1\}^b$ are such that $\mathcal{F}(m_1^{(0)}, \iota) = \mathcal{F}(m_1^{(1)}, \iota) =: h_1$ and $\mathcal{F}(m_i^{(0)}, h_{i-1}) = \mathcal{F}(m_i^{(1)}, h_{i-1}) =: h_i$ for $1 < i \leq d$, then the 2^d $d \times b$-bit-long messages $m_s := m_1^{(s[1])}||\cdots||m_d^{(s[d])}$ indexed by $s \in \{0,1\}^d$ form a 2^d-*collision* for $\mathcal{H}$, i.e. $\forall s$ $\mathcal{H}(m_s) = c$ for some constant $c \in \{0,1\}^n$.

2 Theoretical study

Q.1: Let $\mathcal{H}$ be as in the previous section, what is the time cost of computing a 2^d-collision using the above attack, assuming that $\mathcal{F}$ is ideally random?*

Q.2: Assume now that $\mathcal{H}$ itself is ideal, what is the complexity of computing a 2^d-collision for "small" values of 2^d, where you may use the following (actually incorrect) approximations:

*Meaning that for all x, y, the outputs $\mathcal{F}(x, y)$ are uniformly and independently distributed.

— $\binom{q}{2^d} \approx q^{2^d}$ (quite wrong, esp. for large (w.r.t. q) values of 2^d);

— If $\mathcal{L}$ is a set of uniformly and independently distributed random variables over $\mathcal{D}$, then all its $\binom{\#\mathcal{L}}{2^d}$ size-2^d subsets are uniformly and independently distributed over $\mathcal{D}^{2^d}$.

Q.3: Does a narrow-pipe Merkle-Damgård hash function with an ideal compression function behave like an ideal hash function?

Q.4: What is the the time cost of the attack from Section 1 for $\mathcal{H}$ a *wide-pipe* Merkle-Damgård hash function, with a hash size half the chaining value size (that is, where $\mathcal{H} : \{0,1\}^* \rightarrow \{0,1\}^n$ is as in Section 1 except that it uses a compression function $\mathcal{F} : \{0,1\}^b \times \{0,1\}^{2n} \rightarrow \{0,1\}^{2n}$ and $\mathcal{H}(m) := \lfloor h_\ell \rfloor_n$ (with $\lfloor \cdot \rfloor_x$ denoting truncation to the x least significant bits)), where we still assume $\mathcal{F}$ to be ideal?

How does this compare with a narrow-pipe Merkle-Damgård hash function? With an ideal hash function?

3 Implementing the attack

Download the tarball https://membres-ljk.imag.fr/Pierre.Karpman/mc.tar.bz2. The file `mc48.h` defines a function `tcz48_dm` which implements a toy compression function with 128-bit message blocks and 48-bit chaining values, and an associated narrow-pipe Merkle-Damgård hash function `ht48`. The file `xoshiro256starstar.h` defines a pseudo-random number generator `xoshiro256starstar_random` that you may use in your program.

Q.5: Implement in C the multi-collision attack described in Section 1 for the hash function `ht48`. You must write this as a function `void attack(int d)` which takes as input an argument `d` and writes on the standard output a list of 2^d colliding messages. An example of output with basic formatting is given in the tarball. Note that you are ***not*** allowed to rely on external software or library functions to implement the data structures that you may need.
ADVICE:

— Start by writing a function:

```
void find_col(uint8_t h[6], uint8_t m1[16], uint8_t m2[16])
```

that searches for a collision for the *compression function* `tcz48_dm`.

— For the considered hash function output size of 48 bits, an algorithm using a lot of memory is acceptable, but choose your data structures wisely.

— The full attack should not need much more than a hundred lines to be implemented.

— Don't forget to use optimisation flags when compiling.

— As an indication of acceptable performance, it took 208 seconds on an average laptop to produce the example output found in the tarball.

Q.6: Compute a few (e.g. up to 10) 2, 4, 8, and 16-collisions, and compare the experimental performance of your attack with the theoretical analysis you carried out in **Q.1**.