Assignment: Keccak

PV079, Autumn 2022

Deadline OCT 17 2022, 23:59 (CET)

Introduction

The topic of this assignment is the cryptographic hash function Keccak. Your goal will be to familiarize yourself with the algorithm and analyze the collision resistance and the avalanche effect of Keccak. The assignment is divided into four tasks with one bonus subtask. You can gain up to 10 points (+1 bonus point).

Important notes:

- Implementation of Keccak can be found in https://keccak.team. Note that you can use other third-party implementations of Keccak, provided they are correct. Furthermore, for some of the tasks, you will need to analyze the internal states of the algorithm, so don't be afraid to modify the implementation in any appropriate way.
- The choice of a programming language is up to you. However, if marking your solution will require a deep investigation or even modification of your code, we support only the following: Python, C/C++, and Rust.
- For tasks demanding some non-trivial computing power, please observe the provider's rules of the computing power you will use and do not overload these resources at the expense of other users. There will be no excuses of abuse caused by your work on this assignment.
- Do not confuse Keccak for Keccak[c], Keccak-f[b] or SHA-3. Keccak is a family of sponge functions. Keccak[c] is an instance of Keccak (e.g., Keccak[224]). Keccak-f[b] denotes the underlying permutation of Keccak. Finally, SHA-3 is the particular instance of Keccak that has been standardized by NIST and is also the main focus of this assignment. To further clarify, the delimitedSuffix for SHA-3 is equal to 0x06.
- Please, be precise and concise with your explanations and descriptions. Try to avoid vague sentences. For each task, the commentary should not exceed 500 chars.

Task 1 (1 point)

Shortly describe how the security of Keccak depends on its parameters. Find the setting of Keccak and message (preimage) with the following hash value[1]:

0x94bd25c4cf6ca889126df37ddd9c36e6a9b28a4fe15cc3da6debcdd7

Task 2 (2 points)

Find a collision (two messages `m1, m2`) for the capacity `c = 0` and the hash size (at least) 128 bits. Explain **shortly** why the messages form collision *regardless of the hash size*.

Task 3 (4 points)

Find a collision `m3, m4` for the greatest possible capacity `c` $\geq$ `32` and the hash size (at least) 128 bits. Describe **shortly** what algorithm you used to find the collision. Note that the choice of the algorithm can significantly affect the search time. We recommend using an algorithm[2] based on the birthday paradox. The time to find the collision for the lowest accepted value `c = 32` should not take more than a couple of minutes with an appropriate implementation.

Bonus: Solutions with the greatest capacity value will be awarded by one extra point.

Task 4 (3 points)

In this task you will illustrate the avalanche effect of the underlying Keccak permutation function f (Keccak-$f[1600]$): Show that a change of single bit of the input to the f function results in change of approximately 50% of output bits. Take an input block

$$I(\text{199 of zero (0x00) Bytes followed by 1 Byte = (your UCO \% 256)}),$$

do a single bitflip to I (obtaining I') and apply f to I and I'. Compute the number of changed bits ($f(I')$ vs $f(I)$). Perform the same for all possible (1600) bitflips of I and compute the maximal, minimal, and average number of changed bits.

[1] Hint: NIST publishes test vectors as part of their Cryptographic Algorithm Validation Program.

[2] More interested students can look up the Floyd's cycle detection algorithm.

Submission

Pack all the source code you used to one ZIP file with file `results_XXXXXX.json` where XXXXXX stands for your UCO. This JSON file will contain:

Task 1: Keccak description, message, capacity and hash.

Task 2: An explanation about the length, two messages `m1, m2` of the same hash with `c = 0`, hash.

Task 3: Two messages `m3, m4` of the same hash, capacity and their hash.

Task 4: Maximal, minimal and average number of changed bits.

The JSON file `results_XXXXXX.json` **must be** of the following format:

```
{
"D1": "Keccak setting is ...",
"m0" : "0x1234abcd...",
"c0"  : 12,
"hash0" : "0x94bd25...",

"D2" : "The reason for the ..",
"m1" : "0x414a4bc...",
"m2" : "0x212cdef...",
"c12" : 0,
"hash12" : "0xa8e8...",

"D3" : "For a fixed m3 I used brute-force to find m4 ...",
"m3" : "0x43ed3bc...",
"m4" : "0xe1cc75d...",
"c34" : 64,
"hash34" : "0xbd4ac...",

"max" : 840,
"min" : 760,
"avg" : 140.05
}
```

For correct submission, just replace the values with corresponding keys. In particular, do not remove or add keys. Each value should be one of the following:

- integer (capacities `c0, c12, c34` and `max, min`)
- decimal point number (`avg` rounded to 2 decimal places with decimal point)

- hexadecimal string[3] (messages `m0, m1, m2, m3, m4` and hashes `hash0, hash12, hash34`)
- human readable string (descriptions `D1, D2, D3`)
- If you do not have the value (e.g. you skipped given task) use null instead e.g. `"avg" : null`

For your convenience, we have prepared a short script `validate_format.py` that will check that your submission has the correct format (A zip file containing the JSON file `results_XXXXXX.json`).

```
python validate_format.py your_zip.zip
```

Make sure to check your submission as any deviations from the stated format will be heavily **penalized**. As always, feel free to ask any question in the discussion forum for this assignment.

Good luck!

PV079 Team

[3] Hexadecimal string should be here represented with the 0x prefix, in lower case and without spaces.