OBSERVATION: An explicit form for a class of second preimages for any message M for the SHA-3 candidate Keccak

Danilo Gligoroski[1], Rune Steinsmo Ødegård[2], and Rune Erlend Jensen[2,3]

[1] Department of Telematics, NTNU, Trondheim, Norway
danilo.gligoroski@item.ntnu.no
[2] Centre for Quantifiable Quality of Service in Communication Systems - Q2S, NTNU, Trondheim, Norway
rune.odegard@q2s.ntnu.no
[3] Department of Computer and Information Science, NTNU, Trondheim, Norway
runeerle@stud.ntnu.no

Abstract. In this short note we give an observation about the SHA-3 candidate KECCAK$[r, c, d]$, where the parameters r, c and d receive values from the formal proposal for the KECCAK hash function (with the hash output of $n = \frac{c}{2}$ bits). We show how an attacker that will spend a one-time effort to find a second preimage for the value $z_0 =$ KECCAK$[r, c, d](0^r)$ will actually get infinite number of second preimages for free, for any message M. Our observation is an adaptation of similar attacks that have been reported by Aumasson et.al and Ferguson et.al for the SHA-3 candidate CubeHash. By this observation we do not contradict security claims present in the official KECCAK submission, but we allocate a property in the design of the function: we get an explicit form for a class of second preimages for any message M. As far as we know, this kind of property is not known neither for MD5, SHA-1, SHA-2 nor the other SHA-3 candidates.

1 Description of the observation

Hash function designs based on sponge functions are recent design concept invented in 2007 by Bertoni, Daemen, Peeters and Van Assche [1]. The design concept has attracted big interest by cryptographic hash designers and in the ongoing SHA-3 competition there are four sponge (or sponge-like) designs.

As an introduction to this note we want to recall the remark that sponge function designers have written in their paper [1]: *"More recently, a series of attacks [9, 10, 5, 12] has shown that certain hash function constructions do not offer as much security as expected, leading to the introduction of yet other criteria, such as chosen target forced prefix preimage resistance. As was already predicted in [1], there is no reason to assume*

that no new criteria will appear, so the design of a hash function seems like a moving target.".

In this observation we will show one property that is present in sponge function designs based on permutations, and as far as we know it is not present in other hash designs (like wide-pipe or narrow-pipe Merkle-Damgård designs). From the point of view of the "moving target" that sponge designers were talking about in their paper, it seems that one property present in older designs (MD5, SHA-1, SHA-2) that we took for granted and there was no attempt to define it precisely as a property (or requirement) is not present in sponge function designs based on permutations.

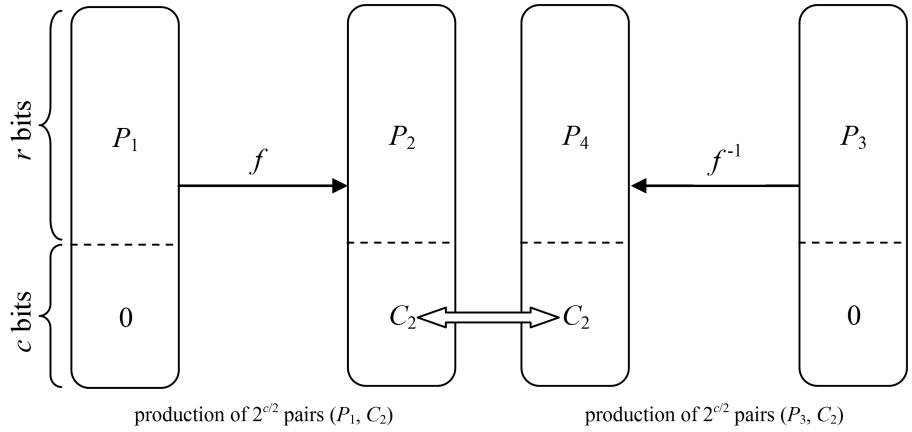

Fig. 1. One time effort of $2 \times 2^{c/2} = 2 \times 2^n$ calls to the permutation f and its inverse f^{-1} in order to find the messages P_1, P_2, P_3 and P_4 that give collision in the part C_2. This computational effort is independent of the messages that will be attacked afterwards.

Namely, the essence of the observation is the following: *If we are requested to find a second preimage of the zero message* 0^r *for the hash function* H *(where* H *is any hash function from the set* {*MD5, SHA-1, SHA-2* } *we will need approximately* 2^n *calls to its compression function. This effort should be non-correlated with our efforts to find the second preimage for the hash* $H(M)$ *of another message* M *i.e. for finding that second preimage we will need again* 2^n *calls to the compression function. However, in this observation we will show that for the the Second Round SHA-3 candidate* KECCAK*[2] an attacker that will find a second preim-*

age of the message 0^r *will have for free second preimages for any other message* M*.*

We will consider the official variant of KECCAK$[r, c, d]$, where the parameters r, c and d receive values from the official proposal for the KECCAK hash function and where the hash output is $n = \frac{c}{2}$ bits.

Our observation is in fact based on the observations and attacks in [3] that Aumasson, Meier, Naya-Plasencia and Peyrin did against CubeHash[4] hash function and on attacks in [5] that Ferguson, Lucks and McKay did also against CubeHash hash function.

The idea is presented in Figure 1 and the goal is to find one internal collision in the part that has c bits (the part that represents the capacity of the hash function). More concretely, we want to find two r-bit messages P_1 and P_3 such that:

$$\begin{cases} f[r+c](P_1||0^c) = (P_2||C_2), \\ f^{-1}[r+c](P_3||0^c) = (P_4||C_2). \end{cases} \tag{1}$$

If the design of the permutation f is such that it behaves as a pseudo-random permutation, then finding the required collision will need approximately $2 \times 2^{c/2} = 2 \times 2^n$ calls to the permutation f and its inverse f^{-1}. However, this effort can be performed independently of the messages for which we will launch afterwards a second-preimage attack.

Now, once we have found the messages P_1, P_2, P_3 and P_4 let us investigate how many second preimages we have for the hash value $z_0 = \text{KECCAK}[r, c, d](0^r)$.

Proposition 1. *The message* $P_1||(P_2 \oplus P_4)||P_3$ *is the second-preimage for* $z_0 = \text{KECCAK}[r, c, d](0^r)$*.*

Proof. For digesting the message 0^r we have the following iterative process:

KECCAK$[r, c, d](0^r)$

INITIALIZATION AND PADDING

$S_0^{(1)} = 0^{r+c}$,

$M = 0^r||PADDING \equiv$

$\equiv 0^r||\underbrace{(\texttt{0x01}||byte(d)||byte(r/8)||\texttt{0x01}||\texttt{0x00}||...||\texttt{0x00})}_{r \text{ bits}}$,

ABSORBING PHASE

$S_1^{(1)} \leftarrow S_0^{(1)} \oplus (0^r||0^c)$,

$S_2^{(1)} \leftarrow f(S_1^{(1)})$

$S_3^{(1)} \leftarrow S_2^{(1)} \oplus (PADDING||0^c)$,

$S_4^{(1)} \leftarrow f(S_3^{(1)})$

SQUEEZING PHASE

$z_0^{(1)} \leftarrow$ first $\frac{c}{2}$ bits of $S_4^{(1)}$.

For digesting the message $P_1||(P_2 \oplus P_4)||P_3$ we have the following iterative process (in absorbing phase we are applying relations (1)):

KECCAK$[r, c, d](P_1||(P_2 \oplus P_4)||P_3)$
INITIALIZATION AND PADDING
$S_0^{(2)} = 0^{r+c}$,
$M = P_1||(P_2 \oplus P_4)||P_3||PADDING \equiv$
$\equiv P_1||(P_2 \oplus P_4)||P_3||\underbrace{(\texttt{0x01}||byte(d)||byte(r/8)||\texttt{0x01}||\texttt{0x00}||...||\texttt{0x00})}_{r \text{ bits}}$,
ABSORBING PHASE
$S_1^{(2)} \leftarrow S_0 \oplus (P_1||0^c)$,
$S_2^{(2)} \leftarrow f(S_1^{(2)}) \equiv (P_2||C_2)$
$S_3^{(2)} \leftarrow S_2^{(2)} \oplus (P_2 \oplus P_4||0^c) \equiv (P_4||C_2)$,
$S_4^{(2)} \leftarrow f(S_3^{(2)}) \equiv (P_3||0^c)$
$S_5^{(2)} \leftarrow S_4^{(2)} \oplus (P_3||0^c) \equiv 0^{r+c} \equiv S_1^{(1)}$,
$S_6^{(2)} \leftarrow f(S_5^{(2)}) \equiv S_2^{(1)}$
$S_7^{(2)} \leftarrow S_6^{(2)} \oplus (PADDING||0^c) \equiv S_3^{(1)}$,
$S_8^{(2)} \leftarrow f(S_7^{(2)}) \equiv S_4^{(1)}$
SQUEEZING PHASE
$z_0^{(2)} \leftarrow$ first $\frac{c}{2}$ bits of $S_8^{(2)} =$ first $\frac{c}{2}$ bits of $S_4^{(1)} = z_0^{(1)}$.

□

Beside the message $P_1||(P_2 \oplus P_4)||P_3$ that is a second preimage for the value $z_0 = \text{KECCAK}[r, c, d](0^r)$ we have an infinite set of second preimages defined with the following Proposition:

Proposition 2. *Let $\mathcal{M}_0$ be the set of messages defined by the following definition:*

$$\mathcal{M}_0 = \{P_1||(P_2 \oplus P_4)||\big((P_3 \oplus P_1)||(P_2 \oplus P_4)\big)^i||P_3 \mid \textit{for } i \geq 1\}, \qquad (2)$$

where the expression $\big((P_3 \oplus P_1)||(P_2 \oplus P_4)\big)^i$ means i times concatenation of the message $(P_3 \oplus P_1)||(P_2 \oplus P_4)$.

Every message $M_0 \in \mathcal{M}_0$ is a second-preimage for the value $z_0 =$ KECCAK$[r, c, d](0^r)$. □

The role of the values P_1, P_2, P_3 and P_4 do not stop here. We can use them to define infinite classes of second preimages for the digests computed for every message M.

Proposition 3. *Let $z_0 =$ KECCAK$[r, c, d](M)$ be the hash digest for an arbitrary message M, and let $M' = M_0||M_1 \ldots||M_k$ be the message obtained from the message M after a proper padding as it is defined in the function* KECCAK.

Let $\mathcal{M}_{any}$ be the set of messages defined by the following definition:

$$\mathcal{M}_{any} = \{P_1||(P_2 \oplus P_4)||\big((P_3 \oplus P_1)||(P_2 \oplus P_4)\big)^i||(P_3 \oplus M_0) \mid \textit{for } i \geq 1\}. \quad (3)$$

Every message $M_{any} \in \mathcal{M}_{any}$ is a second-preimage for the value $z_0 =$ KECCAK$[r, c, d](M)$. □

2 Conclusions and comments

Our opinion is that the designers' decision not to include explicitly the length of the message in the $PADDING$ part in combination with the invertibility of the permutation f and the capacity $c = 2n$ gives opportunities to define huge sets of second preimages for any message.

The observed property is unreachable for any practical computational effort since to find the values of P_1, P_2, P_3 and P_4 in relations (1) one must spend 2×2^n calls to the permutation f and its inverse f^{-1}. The designers of KECCAK in [1] have discussed similar scenarios of finding inner collisions and basically they consider that finding an inner collision should be hard. Additionally in the introduction of their original paper [1] they have already stated that an inner collision allows generating colliding messages ad libitum.

Here, with this observation we extend that finding not just to generating colliding messages ad libitum but generating second preimages ad libitum. Or, we can interpret our observation from the following point of view: For all other known cryptographic hash functions (MD5, SHA-1, SHA-2, SHA-3 candidates), there is not known explicit form for a class of second preimages for an arbitrary message M. However, for KECCAK one class of second preimages is known and is defined in Proposition 3.

We can give here two research directions that are logically popping up from this observation.

1. Our observation is solely based on the general design decisions made by the designers of KECCAK, and are not exploiting the internal definitions of the permutation f, that can possibly lead into reducing the computational efforts of 2×2^n calls to the permutation f and its inverse f^{-1} in relations (1).
2. It would be interesting to investigate applicability of this observation to other sponge functions that are in the Second Round of SHA-3 competition.

Acknowledgement

We would like to thank the KECCAK team for valuable discussions while preparing this note that helped to improve the quality of the text a lot.

References

1. G. Bertoni, J. Daemen, M. Peeters and G. Van Assche: "Sponge Functions", Ecrypt Hash Workshop 2007.
2. Guido Bertoni, Joan Daemen, Michaël Peeters, Gilles Van Assche: "Keccak sponge function family main document, Submission to NIST (Round 2)". Available: `http://csrc.nist.gov/groups/ST/hash/sha-3/Round2/documents/Keccak_Round2.zip`
3. Jean-Philippe Aumasson, Eric Brier, Willi Meier, Mari'a Naya-Plasencia, Thomas Peyrin:"Inside the Hypercube", In Proceedings of ACISP, LNCS 5594, pp. 202-213, Springer, 2009
4. Daniel J. Bernstein: "CubeHash, Submission to NIST (Round 2)". Available: `http://csrc.nist.gov/groups/ST/hash/sha-3/Round2/documents/CubeHash_Round2.zip`
5. Niels Ferguson, Stefan Lucks, Kerry A. McKay:"Symmetric States and their Structure: Improved Analysis of CubeHash", Cryptology ePrint Archive, Report 2010/273, 2010