Narrow-pipe SHA-3 candidates differ significantly from ideal random functions defined over big domains

Danilo Gligoroski[1]

[1]Faculty of Information Technology, Mathematics and Electrical Engineering, Institute of Telematics, Norwegian University of Science and Technology, Trondheim, Norway, e-mail: Danilo.Gligoroski@item.ntnu.no

Abstract

In the SHA-3 competition 4 candidates in the second round are narrow-pipe designs. Those are: BLAKE, Hamsi, SHAvite-3 and Skein. In this paper we show that there exist many concrete cases when these functions differ significantly from ideal random functions $H : \{0,1\}^N \rightarrow \{0,1\}^n$ that map bit strings from a big domain where $N = n + m,\ m \geq n$ ($n = 256$ or $n = 512$). Our observation is simple: For an ideal random function with a big domain space $\{0,1\}^N$ and a finite co-domain space $Y = \{0,1\}^n$, for every element $y \in Y$, the probability $Pr\{H^{-1}(y) = \varnothing\} \approx e^{-2^m} \approx 0$ where $H^{-1}(y) \subseteq \{0,1\}^N$ and $H^{-1}(y) = \{x \mid H(x) = y\}$ (in words - the probability that elements of Y are "unreachable" is negligible). However, for the aforementioned hash functions, for certain values of N (the values that are causing the last padded block that will be processed by the compression function of these functions to have no message bits), there exists a huge non-empty subset $Y_\varnothing \subseteq Y$ with a volume $|Y_\varnothing| \approx e^{-1}|Y| \approx 0.36|Y|$ for which it is true that for every $y \in Y_\varnothing,\ H^{-1}(y) = \varnothing$.

This observation shows that these functions differ a lot from an ideal random functions with big domains $\{0,1\}^N,\ N = n + m,\ m \geq n$, and they can not be used as a concrete instantiation in any protocol that proves its security using the random oracle model.

1 Introduction

The importance of cryptographic functions with arbitrary input-length have been confirmed and re-confirmed numerous times in hundreds of scenarios in information security. The most important properties that these functions have to have are collision-resistance, preimage-resistance and second-preimage resistance. However, several additional properties such as multi-collision resistance, being pseudo-random function, or being a secure MAC, are also considered important.

All practical cryptographic hash function constructions have iterative design and they use a supposed (or conjectured to be close to) ideal finite-input random

function (called compression function) $C : \{0,1\}^m \rightarrow \{0,1\}^l$ where $m > l$, and then the domain of the function C is extended to the domain $\{0,1\}^*$ in some predefined iterative chaining manner.[1]

The way how the domain extension is defined reflects directly to the properties that the whole cryptographic function has. For example domain extension done by the well known Merkle-Damgård construction transfers the collision-resistance of the compression function to the extended function. However, as it was shown in recent years, some other properties of this design clearly show non-random behavior (such as length-extension vulnerability, vulnerability on multi-collisions e.t.c.).

The random oracle model has been proposed to be used in cryptography in 1993 by Bellare and Rogaway [1]. Although it has been shown that there exist some bogus and impractical (but mathematically correct) protocols that are provably secure under the random oracle model, but are completely insecure when the ideal random function is instantiated by any concretely designed hash function [2], in the cryptographic practice the random oracle model gained a lot of popularity. It has gained that popularity during all these years, by the simple fact that protocols proved secure in the random oracle model when instantiated by concrete "good" cryptographic hash functions, are sound and secure and broadly employed in practice.

In this note we show that four of the SHA-3 [3] second round candidates: BLAKE [4], Hamsi [5], SHAvite-3 [6] and Skein [7] act pretty differently than an ideal random function $H : \mathcal{D} \rightarrow \{0,1\}^n$ where $\mathcal{D} = \bigcup_{i=0}^{\text{maxbitlength}} \{0,1\}^i$ and "maxbitlength" is the maximal bit length specified for the concrete functions i.e. $2^{64} - 1$ bits for BLAKE-32, Hamsi, and SHAvite-3-256, $2^{128} - 1$ bits for BLAKE-64 and SHAvite-3-512 and $2^{99} - 8$ bits for Skein. All our claims and proofs are of "existential" nature. Namely we show their deviance from an ideal random function with existential proofs, not by any concrete design of sets of elements that have abnormal non-random behavior. Thus, we are not disproving the security claims for these functions (collision resistance, preimage and second preimage resistance), but we clearly put them in a category of cryptographic hash functions that can not replace the ideal random function in the security proofs based on the random oracle model.

2 Some basic mathematical facts for ideal random functions

We will discuss the properties of ideal random functions over finite and infinite domains.[1] More concretely we will pay our attention for:

[1] The infinite domain $\{0,1\}^*$ in all practical implementations of cryptographic hash functions such as SHA-1 or SHA-2 or the next SHA-3 is replaced by some huge practically defined finite domain such as the domain $\mathcal{D} = \bigcup_{i=0}^{\text{maxbitlength}} \{0,1\}^i$, where maxbitlength $= 2^{64} - 1$ or maxbitlength $= 2^{128} - 1$.

Finite narrow domain: Ideal random functions $C : X \to Y$ mapping the domain of n-bit strings $X = \{0,1\}^n$ to itself i.e. to the domain $Y = \{0,1\}^n$, where $n > 1$ is a natural number;

Finite wide domain: Ideal random functions $W : X \to Y$ mapping the domain of $n+m$-bit strings $X = \{0,1\}^{n+m}$ to the domain $Y = \{0,1\}^n$, where $m \geq n$;

Proposition 1 *Let $\mathcal{F}_C$ be the family of all functions $C : X \to Y$ and let for every $y \in Y$, $C^{-1}(y) \subseteq X$ be the set of preimages of y i.e. $C^{-1}(y) = \{x \in X \mid C(x) = y\}$. For a function $C \in \mathcal{F}_C$ chosen uniformly at random and for every $y \in Y$ the probability that the set $C^{-1}(y)$ is empty is approximately e^{-1} i.e.*

$$Pr\{C^{-1}(y) = \varnothing\} \approx e^{-1}. \tag{1}$$

Proof: We can look the function C as a table

$$C \equiv \begin{cases} C(0) & = & C_0 \\ C(1) & = & C_1 \\ & \vdots & \\ C(2^{n-1}) & = & C_{2^{n-1}}. \end{cases}$$

Since we take the function $C \in \mathcal{F}_C$ to be chosen uniformly at random, we can consider the process of the definition of its table as a random process

$$C(i) \leftarrow U_{2^{-n}}(i)$$

where $U_{2^{-n}}(i)$ is a uniformly distributed function over the interval of values $\{0, 1, \ldots, 2^{n-1}\}$ i.e.

$$Pr\{U_{2^{-n}}(i) = j \mid i, j \in \{0, 1, \ldots, 2^{n-1}\}\} = \frac{1}{2^n}.$$

Now, for every $y \in Y$, $Pr\{C^{-1}(y) = \varnothing\}$ is equal to the probability for any $i \in \{0, 1, \ldots, 2^n - 1\}$, $i \notin \{C_j \mid j = 0, 1, \ldots, 2^n - 1\}$ i.e. we want the opposite event

$$U_{2^{-n}}(i) \neq j$$

which has a probability

$$1 - \frac{1}{2^n}$$

to appear 2^n times. From here it follows:

$$Pr\{C^{-1}(y) = \varnothing\} = \left(1 - \frac{1}{2^n}\right)^{2^n} \approx e^{-1}.$$

□

Corollary 1 *If the function $C \in \mathcal{F}_C$ is chosen uniformly at random, then there exists a set $Y^C_\varnothing \subseteq Y$ such that for every $y \in Y^C_\varnothing$, $C^{-1}(y) = \varnothing$ and*

$$|Y^C_\varnothing| \approx e^{-1}|Y| \approx 0.36|Y|$$

.

Proposition 2 *Let $\mathcal{F}_W$ be the family of all functions $W : X \to Y$ where $X = \{0,1\}^{n+m}$ and $Y = \{0,1\}^n$. Let for every $y \in Y$, $W^{-1}(y) \subseteq X$ be the set of preimages of y i.e. $W^{-1}(y) = \{x \in X \mid W(x) = y\}$. For a function $W \in \mathcal{F}_W$ chosen uniformly at random and for every $y \in Y$ the probability that the set $W^{-1}(y)$ is empty is approximately e^{-2^m} i.e.*

$$Pr\{C^{-1}(y) = \varnothing\} \approx e^{-2^m}. \tag{2}$$

Proof: We can repeat the proof technique used in previous case, but now the number of elements in the table that would define W is 2^{n+m}. From there would follow that

$$Pr\{C^{-1}(y) = \varnothing\} = \left(1 - \frac{1}{2^n}\right)^{2^{n+m}} = \left(\left(1 - \frac{1}{2^n}\right)^{2^n}\right)^{2^m} \approx e^{-2^m}.$$

□

In what follows for the sake of clarity we will work on bit-strings of length which is multiple of n. Namely we will be interested on strings $M = M_1||\ldots||M_i$ where every $|M_j| = n, j = 1,\ldots,i$. Further, we will be interested in practical constructions of cryptographic hash functions that achieve a domain extension from a narrow-domain to the full infinite domain. We will need the following Lemma:

Lemma 1 *Let $\mathcal{F}_{C_\nu}$ be a countable family of functions $C_\nu : X \to Y$, $\nu \in \mathbb{N}$ and let $C : X \to Y$ is one particular function, where C_ν and C are chosen uniformly at random. Let us have a function Rule $: \mathbb{N} \times Y \to \mathcal{F}_{C_\nu}$ that chooses some particular random function from the family $\mathcal{F}_{C_\nu}$ according to a given index and a value from Y. If we define a function $H : (\{0,1\}^n)^i \to Y$ that maps the finite strings $M = M_1||\ldots||M_i$ to the set of n-bit strings $Y = \{0,1\}^n$ as a cascade of functions:*

$$\begin{aligned} H(M) = H(M_1||\ldots||M_i) = \; & C_{Rule(1,IV)}(M_1) \circ C_{Rule(2,C_{Rule(1,IV)}(M_1))}(M_2) \circ \\ & \circ \cdots \circ \\ & \circ\, C_{Rule(i,C_{Rule(i-1,\cdot)}(M_{i-1}))}(M_i) \circ \\ & \circ\, C \end{aligned} \tag{3}$$

then for every $y \in Y$ the probability that the set $H^{-1}(y)$ is empty is approximately e^{-1}.

Proof: What this Lemma claims is that it does not matter what type of clever chaining rule someone will use in the domain extension definition, as long as there is a final invocation of a function $C : X \rightarrow Y$ which employment is independent from the content of the message M and that is mapping a narrow domain of n-bit strings $X = \{0,1\}^n$ to itself i.e. to the domain $Y = \{0,1\}^n$. Then, using the Proposition 1 the proof follows. □

3 The case of the cryptographic function BLAKE

Let us analyze the iterated procedure defined in BLAKE-32 (and the case for BLAKE-64 is similar). First, a message M is properly padded:

$$M \leftarrow M||1000\ldots0001\langle l_{64}\rangle$$

and then is parsed into N, 512-bit chunks:

$$M \equiv m^0, \ldots, m^{N-1}.$$

The variable l^i is defined as a number of processed bits so far. We quote the description of the padding from [4]:

> For example, if the original (non-padded) message is 600-bit long, then the padded message has two blocks, and $l^0 = 512$, $l^1 = 600$. A particular case occurs when the last block contains no original message bit; for example a 1020-bit message leads to a padded message with three blocks (which contain respectively 512, 508, and 0 message bits), and we set $l^0 = 512$, $l^1 = 1020$, $l^2 = 0$.

Now, let us take that we want to hash just 1020-bit long messages M with BLAKE-32 (the size of 1020 is chosen to fit the example in the original documentation, but it can be also 1024, or any multiple of 512, which is a common action if BLAKE-32 would be used as a PRF or KDF hashing a pool of randomness that is exactly multiple of 512 bits). The iterative procedure will be:

$h^0 = \text{IV}$

for $i = 0, \ldots, 2$

$h^{i+1} = \textbf{compress}(h^i, m^i, s, l^i)$

return h^2

or equivalently:

$$BLAKE\text{-}32(M) = \textbf{compress}(\textbf{compress}(\textbf{compress}(h^0, m^0, s, 512), m^1, s, 1020), m^2, s, 0).$$

Note that $m^2 = const$ does not have any bit from the original 1020-bit message M. So, we have that the final 256-bit hash value that is computed is:

$$BLAKE\text{-}32(M) = \mathbf{compress}(h^1, const, s, 0).$$

If we suppose that the compression function of $BLAKE$-32 is ideal, from the Proposition 1 and Lemma 1 we get that there is a huge set $Y_\varnothing \subseteq \{0,1\}^{256}$, with a volume $|Y_\varnothing| \approx 0.36 \times 2^{256} \approx 2^{254.55}$ i.e.

$$Pr\{BLAKE\text{-}32^{-1}(M) = \varnothing\} = e^{-1}.$$

On the other hand, for an ideal random function $W : \{0,1\}^{1020} \to \{0,1\}^{256}$ from Proposition 2 we have that

$$Pr\{W^{-1}(M) = \varnothing\} = e^{-2^{764}}.$$

4 The case of the cryptographic function Hamsi

Let us analyze the iterated procedure defined in Hamsi-256 (and the case for Hamsi-512 is similar). Let M be a properly padded message i.e.

$$M = M_1 || \dots M_{l-1} || M_l,$$

where the last block M_l does not contain message bits but the 64-bit encoding of the length in bits of the original message M i.e. we have that $M_l = \langle l_{64} \rangle$.

Let us hash messages M of length 1024-bits. Then the padded message will have 35 blocks of 32-bits and will have the following form:

$$M = M_1 || \dots M_{32} || 1000 \dots 000 || 000 \dots 000 || 00000000000000000000010000000000$$

The iterative procedure for hashing these messages will be:

$$h_i = (T \circ P \circ C(E(M_i), h_{i-1})) \oplus h_{i-1}, \quad h_0 = iv_{256}, \quad 0 < i < 35,$$

$$\text{Hamsi-256}(M) = (T \circ P_f \circ C(E(M_l), h_{l-1})) \oplus h_{l-1}$$

For the precise definition of the variables used in the iterative process see the Hamsi documentation [5].

From the definition of Hamsi-256 it is obvious that it can act at most as an ideal random function with narrow-domain, but obviously the last call of the compression function for messages of length 1024 bits has no message bits, thus the deviance from an ideal function that maps 1024 bits to 256-bit digest is huge as it is shown in Proposition 1, Proposition 2 and Lemma 1.

5 The case of the cryptographic function SHAvite-3

We will analyze 256-bit version of SHAvite-3, SHAvite-3256 (and the 512-bit version is similar). It uses the HAsh Iterative FrAmework - HAIFA. Hashing with HAIFA has three steps:

1. Message padding, according to the HAIFA padding scheme.
2. Compressing the message using a HAIFA-compatible compression function.
3. Truncating the output to the required length.

Since we will work with 256-bit version of SHAvite-3, the third truncating step will be omitted (which is crucial for our analysis).

We give here the description of the hashing by SHAvite-3 extracted from the documentation in [6]:

The compression is done using a compression function with four inputs:

- A chaining value (of length m_c),
- A message block (of length n),
- The number of bits hashed so far including the current block (a counter of length c),
- A salt (of length s).

In order to compute $HAIFA^C_{salt}(M)$ using the compression function $C : \{0,1\}^{m_c} \times \{0,1\}^n \times \{0,1\}^b \times \{0,1\}^s \mapsto 0,1^{m_c}$ the message is first padded, and divided into l blocks of n bits each, $pad(M) = M_1||M_2||\ldots||M_l$ Now, the user:

1. Sets h_0 as the initial value (according to the procedure defined in Section 3.3).
2. Computes iteratively
$$h_i = C(h_{i-1}, M_i, \#bits, salt).$$
3. Truncates h_l (according to the procedure defined in Section 3.3).
4. Output the truncated value as $HAIFA^C_{salt}(M)$.

The padding rule in SHAvite-3 works to pad the original message such that it is multiple of n bits where $n = 512$ for SHAvite-3-256 or $n = 1024$ for SHAvite-3-256. The padding of a message M has the following steps:

1. Pad with a single bit of 1.
2. Pad with as many 0 bits as needed such that the length of the padded message (with the 1 bit and the 0s) is congruent modulo n to $(n - (t + r))$.

3. Pad with the message length encoded in t bits.

4. Pad with the digest length encoded in r bits.

When a full padding block is added (i.e., the entire original message was already processed by the previous calls to the compression function, and the full message length was already used as an input to the previous call as the #bits parameter), the compression function is called with the #bits parameter set to **zero**.

So, let us hash messages M that are 1024-bits long (this analysis works with messages that are multiple of 512 or 1024 bits) with SHAvite-3-256. The padded message M is:

$$pad(M) = M_1||M_2||M_3,$$

where the final padding block M_3 does not have any message bits and the truncation phase is omitted. Thus,

$$\text{SHAvite-3-256}(M) = HAIFA^C_{salt}(M) = C(C(C(h_0, M_1, 512, salt), M_2, 1024, salt), M_3, 0, salt).$$

Since the final padding block M_3 does not have any message bits, for messages of length 1024 bits, we can treat it as $M_3 = const$ and

$$\text{SHAvite-3-256}(M) = HAIFA^C_{salt}(M) = C(h_2, const, 0, salt).$$

This is exactly the case that is covered by the Proposition 1 (or Lemma 1). Under the assumption that SHAvite-3-256 compression function acts as ideal finite-narrow-domain random function that maps 256 bits to 256 bits, we conclude that SHAvite-3-256(M) differs significantly from an ideal finite-wide-domain random function that maps strings of 1024 bits to hash values of 256 bits.

6 The case of the cryptographic function Skein

The subject of this analysis are the variants Skein-256-256 and Skein-512-512 (which according to the documentation is the primary proposal of the designers [7]). It is an interesting fact that the designs Skein-512-256 and Skein-1024-512 which are double-pipe designs are not suffering from the defects of narrow-pipe compression functions that are extended to the infinite domain and are not affected by this analysis.

The main point of our analysis of Skein-256-256 and Skein-512-512 is the fact that Skein is using a final invocation of UBI (Unique Block Iteration) without an input by any message bits. Namely it uses the output function $Output(G, N_0)$ which takes as parameters the final chaining value G and the number of required output bits N_0. The output is simple run of the UBI function in a counter mode:

$$\begin{aligned} O := \quad & UBI(G, ToBytes(0, 8), T_{out}2^{120})|| \\ & UBI(G, ToBytes(1, 8), T_{out}2^{120})|| \\ & UBI(G, ToBytes(2, 8), T_{out}2^{120})|| \\ & \ldots \end{aligned}$$

Now let us use Skein-256-256 (the case for Skain-512-512 is similar). In that case the chaining value G has 256 bits, and the UBI() is called only once. We can treat that one call of UBI() as a finite-narrow-domain mapping that maps 256 bits of G to 256 bits of O. Thus, from the point of view of Proposition 1 and Proposition **??** we have a clear difference between an ideal function that maps the huge domain $\mathcal{D}$ into the set $\{0,1\}^{256}$ and the function of Skein-256-256. Namely, under the assumption that UBI() acts as an ideal finite-narrow-domain function, from Proposition 1 and Lemma 1 we have that there exist a huge set $Y_{\varnothing} \subseteq \{0,1\}^{256}$, with a volume $|Y_{\varnothing}| \approx 0.36 \times 2^{256} \approx 2^{254.55}$ i.e.

$$Pr\{\text{Skein-256-256}^{-1}(M) = \varnothing\} = e^{-1}.$$

7 Practical consequences of the observed defects of the narrow-pipe designs

We have mentioned that our findings are of "existential" nature i.e. our proofs are not constructional and we are not giving a procedure that will allocate the unreachable set $Y_{\varnothing}$.

An immediate conclusion can be that in the second preimage attack, if one is given as challenge a message whose length implies that the last block processed by the iterated hash consists only of padding bits, then the generic search runs in $0.632 * 2^n$ instead of 2^n (eg, $2^{255.34}$ for 256-bit hashes).

However, the observed defect can be more devastating in some other cases. For example, if we are using the aforementioned hash functions in HMAC where authenticated messages are such that the last block processed by the iterated hash consists only of padding bits, then the volume of the "unreachable" set from the HMAC output grows up to $\approx 53\%$. We note that this is not the case for the double-pipe hash designs.

We are sure that very soon after publishing of this paper, a close look at the consequences of the observed defect of these functions will follow by the cryptographic community.

8 Conclusions and future cryptanalysis directions

We have shown that four narrow-pipe SHA-3 candidates differ significantly from ideal random functions defined over huge domains. The first consequence from this is that they can not be used as an instantiation in security proofs based on random oracle model.

Several other consequences are also evident but will require further careful investigation and elaboration and we leave them as directions for future cryptanalysis. Namely, any use of these functions as PRFs or KDFs or MACs (or HMACs) will

result in outputs that from theoretical point of view differ significantly from outputs of ideal random functions defined over huge domains. Practical consequences of that non-random behavior in numerous algorithms and protocols are yet to be determined.

Acknowledgement

An acknowledgement goes to Sean O'Neil for his private message back in January 2009, expressing suspicion on the cryptographic quality of narrow-pipe hash designs. Back in that time, although I shared his suspicion, I did not know how to show the deviance of these designs from ideal random functions.

My big thanks goes to Alexander Kholosha that recently has pointed out to me that he would not trust cryptographic hash functions that provably have points in their co-domain that are not images of any message from the bigger domain. Connecting his remark with the remark of Sean O'Neil about narrow-pipe hash designs was not so hard.

I would also like to thank Jean-Philippe Aumasson, Eli Biham and Orr Dunkelman for their great comments for improving the clarity of this text.

References

[1] M. Bellare and P. Rogaway: "Random oracles are practical: A paradigm for designing efficient protocols," in CCS 93: Proceedings of the 1st ACM conference on Computer and Communications Security, pp. 6273, 1993.

[2] R. Canetti, O. Goldreich, S. Halevi: "The random oracle methodology, revisited", 30th STOC 1998, pp. 209–218.

[3] National Institute of Standards and Technology: "Announcing Request for Candidate Algorithm Nominations for a New Cryptographic Hash Algorithm (SHA-3) Family". Federal Register, 27(212):62212–62220, November 2007. Available: `http://csrc.nist.gov/groups/ST/hash/documents/FR_Notice_Nov07.pdf` (2009/04/10).

[4] Jean-Philippe Aumasson, Luca Henzen, Willi Meier, Raphael C.-W. Phan: "SHA-3 proposal BLAKE, Submission to NIST (Round 2)". Available: `http://csrc.nist.gov/groups/ST/hash/sha-3/Round2/documents/BLAKE_Round2.zip` (2010/05/03).

[5] Özgül Kücük: "The Hash Function Hamsi, Submission to NIST (Round 2)". Available: `http://csrc.nist.gov/groups/ST/hash/sha-3/Round2/documents/Hamsi_Round2.zip` (2010/05/03).

[6] Eli Biham and Orr Dunkelman: "The SHAvite-3 Hash Function, Submission to NIST (Round 2)". Available: `http://csrc.nist.gov/groups/ST/hash/sha-3/Round2/documents/SHAvite-3_Round2.zip` (2010/05/03).

[7] Niels Ferguson, Stefan Lucks, Bruce Schneier, Doug Whiting, Mihir Bellare, Tadayoshi Kohno, Jon Callas, Jesse Walker: "The Skein Hash Function Family, Submission to NIST (Round 2)". Available: `http://csrc.nist.gov/groups/ST/hash/sha-3/Round2/documents/Skein_Round2.zip` (2010/05/03).