Practical consequences of the aberration of narrow-pipe hash designs from ideal random functions

Danilo Gligoroski[1] and Vlastimil Klima[2]

[1] Faculty of Information Technology, Mathematics and Electrical Engineering, Department of Telematics, Norwegian University of Science and Technology, Trondheim, Norway, e-mail: Danilo.Gligoroski@item.ntnu.no
[2] Independent Cryptologist - Consultant, Prague, Czech Republic, e-mail: v.klima@volny.cz

Abstract. In a recent note to the NIST hash-forum list, the following observation was presented: narrow-pipe hash functions differ significantly from ideal random functions $H : \{0,1\}^N \to \{0,1\}^n$ that map bit strings from a big domain where $N = n + m,\ m \geq n$ ($n = 256$ or $n = 512$). Namely, 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 narrow-pipe hash functions, for certain values of N (the values that are causing the last padded block that is 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 $y \in Y_\varnothing,\ H^{-1}(y) = \varnothing$.
In this paper we extend the same finding to SHA-2 and show consequences of this abberation when narrow-pipe hash functions are employed in HMAC and in two widely used protocols: 1. The pseudo-random function defined in SSL/TLS 1.2 and 2. The Password-based Key Derivation Function No.1, i.e, PBKDF1.

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.

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 a recent note to the NIST hash-forum list [3] it was shown that four of the SHA-3 [4] second round candidates: BLAKE [5], Hamsi [6], SHAvite-3 [7] and Skein [8] 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.

In this paper we extend that finding also to the current cryptographic hash standard SHA-2 [9] and we show what are the security consequences if those hash functions would be used in HMAC and in two widely used protocols: 1. The pseudo-random function defined in SSL/TLS 1.2 [10] and 2. The Password-based Key Derivation Function No.1, i.e, PBKDF1 as defined in PKCS#5 v1 [11].

2 Some basic mathematical facts for ideal random functions

We will discuss the properties of ideal random functions over finite and infinite domains.[3] More concretely we will pay our attention for:

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. *([3]) Let $\mathcal{F}_C$ be the family of all functions $C : X \to Y$ (where X and Y are two sets with equal cardinality) 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,*

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

□

Corollary 1. *([3]) If the function $C \in \mathcal{F}_C$ is chosen uniformly at random, then there exists a set $Y_{\varnothing}^C \subseteq Y$ such that for every $y \in Y_{\varnothing}^C$, $C^{-1}(y) = \varnothing$ and*

$$|Y_{\varnothing}^C| \approx e^{-1}|Y| \approx 0.36|Y|.$$

□

Proposition 2. *([3]) 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,*

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

□

[3] 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 $= 2^{128} - 1$.

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. *([3]) Let $\mathcal{F}_{C_\nu}$ be a countable family of functions $C_\nu : X \to Y$, $\nu \in \mathbb{N}$ and let $C : X \to Y$ be 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}. □

Proposition 3. *Let $C_1 : X \to Y$, $C_2 : X \to Y$ are two particular functions, chosen uniformly at random (where $X = Y = \{0,1\}^n$). If we define a function $C : X \to Y$ as a composition:*

$$C = C_1 \circ C_2 \tag{4}$$

then for every $y \in Y$ the probability P_2 that the set $C^{-1}(y)$ is empty is $P_2 = e^{-1+e^{-1}}$.

Proof. We can use the same technique used in the proof of Proposition 1 in [3] but extended to two domains (i.e, one intermediate domain Z) since we have a composition of two functions. Thus let us put the following notation:

$$C \equiv C_1 \circ C_2 : X \xrightarrow{C_1} Z \xrightarrow{C_2} Y$$

From Proposition 1 it follows that for every $z \in Z$ the probability that the set $C_1^{-1}(z)$ is empty is approximately e^{-1} i.e, the probability that z has a preimage is $(1 - Pr\{C_1^{-1}(z) = \varnothing\}) = (1 - e^{-1})$.

Now, for the probability that the set $C^{-1}(y)$ is empty (for every $y \in Y$) we have:

$$Pr\{C^{-1}(y) = \varnothing\} = \left(1 - \frac{1}{2^n}\right)^{2^n(1-Pr\{C_1^{-1}(y)=\varnothing\})} \approx e^{-1+e^{-1}}.$$

Lemma 2. *$C_1, C_2, \ldots C_k : X \to Y$ are k particular (not necessary different) functions, chosen uniformly at random (where $X = Y = \{0,1\}^n$). If we define a function $C : X \to Y$ as a composition:*

$$C = C_1 \circ C_2 \circ \ldots \circ C_k \tag{5}$$

then for every $y \in Y$ the probability P_k that the set $C^{-1}(y)$ is empty is approximately $P_k = e^{-1+P_{k-1}}$, where $P_1 = e^{-1}$.

Proof. The lemma can be proved by using mathematical induction for the value of k and the Proposition 3.

The Lemma 2 models the probability of some element in Y to have a preimage if we apply consecutively different random functions defined over the same narrow domain $\{0,1\}^n$. Is the sequence P_k convergent? If yes, what is the limit value and what is the speed of the convergence?

In this paper we will give answers on these questions, but we have to stress that the mathematical proofs for some of those answers will be given elsewhere.[4, 5]

Lemma 3. *Let $P_1 = e^{-1}$ and $P_k = e^{-1+P_{k-1}}$. Then the following limit holds:*

$$\lim_{i\to\infty} (\log_2(1 - P_{2^i}) + i - 1) = 0 \tag{6}$$

As a direct consequence of Lemma 3 is the following Corollary:

Corollary 2. *The entropy $E(C(X))$ of the set $C(X) = \{C(x) \mid x \in X\}$, where the function C is a composition of 2^i functions mapping the domain $\{0,1\}^n$ to itself, as defined in (5) is:*

$$E(C(X)) = n + \log_2(1 - P_{2^i}) \tag{7}$$

□

[4] In the initial version of this paper the Lemma 3 was given as a Conjecture, but in the mean time Zoran Šunić from the Department of Mathematics, Texas A&M University, USA has proven it for which we express him an acknowledgement.

[5] After reading our first version of the paper submited to the eprint archive, we got an email from Ernst Schulte-Geers from the German BSI for which we express him an acknowledgement, pointing out that in fact Lemma 3 was known long time ago from the paper of Flajolet and Odlyzko [13].

The last corollary can be interpreted in the following way: With every consecutive mapping of a narrow domain $\{0,1\}^n$ to itself by any random function defined on that domain, the volume of the resulting image is shrinking. The speed of the shrinking is exponentially slow i.e, for shrinking the original volume 2^n of $X = \{0,1\}^n$ to an image set with a volume of 2^{n-i+1} elements, we will need to define a composition of 2^i functions i.e,

$$C = C_1 \circ C_2 \circ \ldots \circ C_{2^i}.$$

3 The narrow-pipe nature of SHA-2 and four SHA-3 candidates

3.1 The case of SHA-2

Let us analyze the iterated procedure defined in SHA-256 (and the case for SHA-512 is similar)[9]. First, a message M is properly padded:

$$M \leftarrow M||1000\ldots000\langle l_{64}\rangle$$

where the 64-bit variable $\langle l_{64}\rangle$ is defined as the length of the original message M in bits. Then the padded message is parsed into N, 512-bit chunks:

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

The iterative procedure for hashing the message M then is defined as:

$h^0 = \text{IV}$
for $i = 0, \ldots, N-1$
  $h^{i+1} =$ **CompressSHA256**(h^i, m^i)
return h^N

where **CompressSHA256**() is the compression function for SHA-256.

Now, let us hash messages that are extracted from some pool of randomness with a size of 1024 bits. The padding procedure will make the final block that would be compressed by the **CompressSHA256**() to be always the same i.e, to be the following block of 512 bits:

$$\underbrace{1000\ldots00010000000000}_{512\ bits}$$

If we suppose that the compression function **CompressSHA256**() 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 $|Y_\varnothing| \approx 0.36 \times 2^{256}$ i.e,

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

On the other hand, for an ideal random function $W : \{0,1\}^{1024} \rightarrow \{0,1\}^{256}$ from Proposition 2 we have that

$$Pr\{W^{-1}(M) = \varnothing\} = e^{-2^{768}} \approx 0.$$

3.2 The case of the Second round SHA-3 candidates BLAKE, Hamsi, SHAvite-3 and Skein

In [3] it was shown that the second round candidates BLAKE, Hamsi, SHAvite-3 and Skein all manifest abberations from ideal random functions defined over wider domains. The basic method how this was shown was the fact that for certain lengths of the messages that are hashed, the final padding block does not contain any bits from the message, and thus acts as an independent random function defined over a narrow domain $X = \{0,1\}^n$ that is mapped to itself.

4 Practical consequences of the observed abberations of the narrow-pipe designs

We point out several concrete protocols that are widely used and where the observed abberations of narrow-pipe hash designs from the ideal random function will be amplified due to the iterative use of hash functions in those protocols.

4.1 Reduced entropy outputs from narrow-pipe hash functions

The first practical consequence is by direct application of the Lemma 2.

Let us consider the following scenario: We are using some hash function that gives us 256 bits of output, and we have a pool of randomness of a size of 2^{20} blocks (where the block size is the size of message blocks used in the compression function of that hash function). The pool is constantly updated by actions from the user and from the running operating system. We need random numbers obtained from that pool that will have preferably close to 256 bits of entropy.

If we use narrow-pipe hash design, then depending on the nature of the distribution of the entropy in the randomness pool, we can obtain outputs that can have outputs with entropy as low as 237 bits or outputs with entropy close to 256 bits.

More concretely, if the distribution of the entropy in the pool is somehow concentrated in the first block (or in the first few blocks), then from

the Lemma 2 we have that the entropy of the output will not be 256 bits but "just" slightly more than 237 bits. We say "just" because having 237 bits of entropy is really high enough value for any practical use, but it is much smaller than the requested value of 256 bits of entropy. In a case of more uniform distribution of the entropy in the whole pool of randomness, the narrow-pipe hash design will give us outputs with entropies close to 256 bits. The cases where due to different reasons (users habits, user laziness, regularity of actions in the operating system, to name some), the pool is feeded with randomness that is concentrated more on some specific blocks, the outputs will have entropy between 237 and 256 bits.

On the other hand, we want to emphasize, if in all this scenarios we use wide-pipe hash design, the outputs will always have close to 256 bits of entropy, regardless where the distribution of the entropy in the pool will be.

From this perspective, we can say that although the consequences can be just of theoretical interest, there are real and practical scenarios where the abberation of narrow-pipe hash design from ideal random functions can be amplified to some more significant and theoretically visible level.

4.2 Reduced entropy outputs from HMACs produced by narrow-pipe hash functions

HMAC [12] is one very popular scheme for computing MAC - Message Authentication Codes when a shared secret is used by the parties in the communication. We are interested in a possible loss of entropy in the HMAC construction if we use narrow-pipe hash constructions.

Proposition 4. *Let a message M be of a size of 256 bits and has a full entropy of 256 and let "secret" be shared secret of 256 bits. If in HMAC construction we use a narrow-pipe hash function that parses the hashed messages in 512 blocks, then $mac = HMAC(secret, M)$ has an entropy of 254.58 bits.*

Proof. Let we use the hash function SHA256 that has the compression function **CompressSHA256**(). From the definition of HMAC we have that

$$mac = HMAC(secret, M) = hash((secret \oplus opad)||hash((secret \oplus ipad)||M))$$

where $\oplus$ is the operation of bitwise xoring and $||$ is the operation of string concatenation.

Computing of *mac* will use four calls of the compression function **CompressSHA256**() in the following sequence:

1. $h_1 = \mathbf{CompressSHA256}(iv_{256}, (secret \oplus ipad)) \equiv C_1(iv_{256})$
2. $h_2 = \mathbf{CompressSHA256}(h_1, M||CONST256) \equiv C_2(h_1)$, where

$$CONST256 = \underbrace{1000\ldots000100000000}_{256\ bits}.$$

3. $h_3 = \mathbf{CompressSHA256}(iv_{256}, (secret \oplus opad)) \equiv C_3(iv_{256})$
4. $mac = h_4 = \mathbf{CompressSHA256}(h_3, h_2||CONST256) \equiv C_4(h_3)$

For a fixed secret key "*secret*" the value h_1 will be always the same and will be obtained with $C_1(iv_{256})$. The function C_2 depends from the message M that has a full entropy of 256 bits, thus C_2 is not one function but it represent a whole class of 2^{256} random functions mapping 256 bits to 256 bits. Thus, we can consider that any call of the function C_2 decreases the entropy of h_2 to $256 + \log_2(1 - P_1)$.

For the value h_3 we have a similar situation as for h_1. Similarly as $C_2()$, the function $C_4()$ is a class of random functions that depends of the value h_2. Since we have already determined that the entropy of h_2 is $256 + \log_2(1 - P_1)$, it follows that for computing the entropy of mac we can apply the Corollary 2 obtaining that entropy $E(mac)$ is

$$E(mac) = 256 + \log_2(1 - P_2),$$

where $P_1 = \frac{1}{e}$, and $P_2 = e^{-1+\frac{1}{e}}$ which gives us the value $E(mac) = 254.58$. □

What is the difference if we use a double-pipe hash function instead of narrow-pipe in Proposition 4? The first difference is off course the fact that the initialization variable in the compression function as well as the intermediate variables h_1, h_2, h_3 and h_4 are 512 bits long, and we will need final chopping. Then, under the assumption that the compression function acts as ideal random function mapping 512 bits to 512 bits, and having the entropy of the message M to be 256, we have that the entropy of h_2 is also 256 (not $256+\log_2(1-P_1)$). The same applies for the entropy of h_4 which will give us that the entropy of mac after the chopping will be 256 bits.

Proposition 5. *Let a message M be of a size of 512 bits and has a full entropy of 512 and let "secret" be shared secret of 256 bits. If in HMAC construction we use a narrow-pipe hash function that parses the hashed messages in 512 blocks, then $mac = HMAC(secret, M)$ has an entropy of 254.58 bits.*

Proof. Let we use the hash function SHA256 that has the compression function **CompressSHA256**(). From the definition of HMAC we have that

$$mac = HMAC(secret, M) = hash((secret \oplus opad)||hash((secret \oplus ipad)||M))$$

where $\oplus$ is the operation of bitwise xoring and $||$ is the operation of string concatenation.

Computing of mac will use five calls of the compression function **CompressSHA256**() in the following sequence:

1. $h_1 = \mathbf{CompressSHA256}(iv_{256}, (secret \oplus ipad)) \equiv C_1(iv_{256})$
2. $h_2 = \mathbf{CompressSHA256}(h_1, M) \equiv C_2(h_1)$
3. $h_3 = \mathbf{CompressSHA256}(h_2, CONST512) \equiv C_3(h_2)$, where
$$CONST512 = \underbrace{1000\ldots0001000000000}_{512\ bits}.$$
4. $h_4 = \mathbf{CompressSHA256}(iv_{256}, (secret \oplus opad)) \equiv C_4(iv_{256})$
5. $mac = h_5 = \mathbf{CompressSHA256}(h_4, h_3||CONST256) \equiv C_5(h_4)$, where
$$CONST256 = \underbrace{1000\ldots000100000000}_{256\ bits}.$$

Above, we consider the call of the function **CompressSHA256**(iv_{256}, $(secret \oplus ipad)$) as a call to an ideal random function $C_1 : \{0,1\}^{256} \rightarrow \{0,1\}^{256}$ that will map the 256-bit value iv_{256} to the 256-bit value h_1. The function C_2 is a specific one. Actually, since it depends from the message M that has a full entropy of 512 bits, C_2 is not one function but it represent a whole class of 2^{512} random functions mapping 256 bits to 256 bits. Thus, we can consider that there is no entropy loss for h_2 i.e, it has a full entropy of 256 bits.

For the value h_3 we start to consider the entropy loss again from the value 256. The call to the function C_3 will decrease the entropy of h_3 to $256 + \log_2(1 - P_1)$. For a fixed secret key "*secret*" the value h_4 will be always the same and will be mapped with $C_5(h_4)$ to the final value mac. Similarly as $C_2()$, the function $C_5()$ is a class of random functions that depends of the value h_3. Since we have already determined that the entropy of h_3 is $256 + \log_2(1 - P_1)$, it follows that for computing the entropy of mac we can apply the Corollary 2 obtaining that entropy $E(mac)$ is

$$E(mac) = 256 + \log_2(1 - P_2),$$

where $P_1 = \frac{1}{e}$, and $P_2 = e^{-1+\frac{1}{e}}$ which gives us the value $E(mac) = 254.58$. □

Again, if we are interested to know what will happen if we use a double-pipe hash function in the Proposition 5, we can say that the entropy of the 512-bit variable h_3 will start to decrease from the value 512 and will be $512+\log_2(1-P_1)$, and the entropy of h_5 will be $512+\log_2(1-P_2)$, that after the final chopping will give us a *mac* with full entropy of 256.

4.3 Loss of entropy in the pseudo-random function of SSL/TLS 1.2

SSL/TLS 1.2 is one very popular suit of cryptographic algorithms, tools and protocols defined in [10]. Its pseudo-random function PRF which is producing pseudo-random values based on a shared secret value "*secret*", a seed value "*seed*" (and by an optional variable called "*label*") is defined as follows:

$$PRF(secret, label, seed) = P_{<hash>}(secret, label \mid\mid seed), \tag{8}$$

where the function $P_{<hash>}(secret, seed)$ is defined as:

$$\begin{aligned} P_{<hash>}(secret, seed) = {} & HMAC_{<hash>}(secret, A(1) \mid\mid seed) \mid\mid \\ & HMAC_{<hash>}(secret, A(2) \mid\mid seed) \mid\mid \\ & HMAC_{<hash>}(secret, A(3) \mid\mid seed) \mid\mid \\ & \ldots \end{aligned} \tag{9}$$

and where $A(i)$ are defined as:

$$\begin{aligned} A(0) &= seed \\ A(i) &= HMAC_{<hash>}(secret, A(i-1)). \end{aligned} \tag{10}$$

Proposition 6. *Let "secret" be shared secret of 256 bits. The entropy $E(A(i))$ of the i-th value $A(i)$ as defined in the equation (10) for the hash function SHA-256 can be computed with the following expression:*

$$E(A(i)) = 256 + \log_2(1 - P_{2i}) \tag{11}$$

where the values P_{2i} are defined recursively in the Lemma 2.

Proof. We can use the same technique described in the previous subsection and in the proof of Proposition 4. Since we have two volume compressive calls of the compression function, and since the computation of $A(i)$ depends on the value of the previous value $A(i-1)$ in the computation of $A(i)$ we have $2i$ times shrinking of the entropy. □

As a direct consequence of the previous Proposition we have the following:

Corollary 3. *Let the size of "$A(i)$ || $seed$" be 512 bits, and let "$secret$" be shared secret of 256 bits. For the i-th part $PRF_i = HMAC_{SHA-256}(secret, A(i) \,||\, seed)$ as defined in the equation (9) the entropy $E(PRF_i)$ can be computed with the following expression:*

$$E(PRF_i) = E(PRF_i) = 256 + \log_2(1 - P_{2i+3}) \tag{12}$$

Proof. Computing of PRF_i will use five calls of the compression function **CompressSHA256**() in the following sequence:

1. $h_1 = \mathbf{CompressSHA256}(iv_{256}, (secret \oplus ipad)) \equiv C_1(iv_{256})$
2. $h_2 = \mathbf{CompressSHA256}(h_1, A(i) \,||\, seed) \equiv C_2(h_1)$
3. $h_3 = \mathbf{CompressSHA256}(h_2, CONST1024) \equiv C_3(h_2)$, where

$$CONST1024 = \underbrace{1000\ldots0010000000000}_{512\ bits}.$$

4. $h_4 = \mathbf{CompressSHA256}(iv_{256}, (secret \oplus opad)) \equiv C_4(iv_{256})$
5. $PRF_i = h_5 = \mathbf{CompressSHA256}(h_4, h_3||CONST256) \equiv C_5(h_4)$, where

$$CONST256 = \underbrace{1000\ldots000100000000}_{256\ bits}.$$

Similarly as in Proposition 5 we can see that the function C_2 is a specific one since it depends from $A(i)$ || $seed$. For a given and fixed $seed$, the entropy of "$A(i)$ || $seed$" is the entropy of $A(i)$ and from Proposition 6, it is $E(A(i)) = 256 + \log_2(1 - P_{2i})$ bits. From here it follows that the entropy of h_2 is $E(h_2) = 256 + \log_2(1 - P_{2i+1})$.

For the value h_3 we further have $E(h_2) = 256 + \log_2(1 - P_{2i+2})$. For a fixed secret key "$secret$" the value h_4 will be always the same and will be mapped with $C_5(h_4)$ to the final value PRF_i, with an entropy

$$E(PRF_i) = 256 + \log_2(1 - P_{2i+3}).$$

□

For illustration we can say that the entropy of $E(PRF_1) = 253.463$, but the entropy of $E(PRF_{60}) = 250.00$.

On the other hand, having in mind the discussions about the different attitude of double-pipe hash function in used HMACs, it is clear that with double-pipe hash designs we will not face this kind of entropy loss.

4.4 Loss of entropy in the PBKDF1

The Password-Based Key Derivation Function number 1 is defined in PKCS#5 v1 [11] and is frequently used in many software products that are generating keys (further used for different cryptographic operations) from passwords.

In its definition the following iterative process is used:

$T_1 = \text{Hash}(P \mid\mid S)$
for $i = 2$ to $Count$
$\quad T_{i+1} = \text{Hash}(T_i)$
return T_{Count}

where P is the password value and S is an 8 byte salt.

As a direct consequence of Lemma 2 and the Corollary 2 we have the following corollary:

Corollary 4. *If the hash function used in PBKDF1 is a hash function with a compression function that is mapping n-bits to n-bits, then the entropy $E(T_{Count})$ of the value T_{Count} can be computed with the following expression:*

$$E(T_{Count}) = n + \log_2(1 - P_{Count}) \tag{13}$$

where the values P_{Count} are defined recursively in the Lemma 2. □

What is interesting, is that in different standards and programmers manuals the recommended values of $Count$ are in the range from 2^{10} to 2^{24}. That means that the loss of entropy in the final value T_{Count} will be from 10 to 24 bits if we use narrow-pipe hash designs, and there will be no entropy loss if we use wide-pipe hash design.

4.5 SHA-2 and narrow-pipe SHA-3 candidates would suffer from the same successful attack exploiting the narrow-pipe abberation

There is one more inconvenience with narrow-pipe hash designs that directly breaks one of the NIST requirements for SHA-3 hash competition [4]. Namely, one of the NIST requirement is: *"NIST also desires that the SHA-3 hash functions will be designed so that a possibly successful attack on the SHA-2 hash functions is unlikely to be applicable to SHA-3."*

Now, from all previously stated in this paper it is clear that if an attack is launched exploiting narrow-pipe weakness of SHA-2 hash functions, then that attack can be directly used also against narrow-pipe SHA-3 candidates.

5 Conclusions and future cryptanalysis directions

We have shown that SHA-2 and the 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.

Further, as an interesting research direction we point to investigations of the stability of the limit in the equation (6). Namely, for the ideal random functions the Corollary 2 says that we need 2^i applications of the functions in order to decrease the entropy of the final image for $i-1$ bits. However, our initial experiments show that if we work with some concrete compression function, then the exact value of the number $e \approx 2.7182818...$ has to be replaced by some other concrete value $e \pm \epsilon$. And then, the speed of the entropy loss can increase dramatically faster than with the case of an ideal random function.

We have shown also several other consequences of using these functions as PRFs or KDFs or MACs (or HMACs). Namely, the outputs from those functions differ significantly from outputs of ideal random functions and have less entropy than it would be expected.

Acknowledgement

We would like to thank Jean-Philippe Aumasson (from the team of BLAKE hash function), and Orr Dunkelman (from the team of SHAvite-3 hash function) for their great comments, and precise remarks that have improved the text significantly. We would like also to thank Zoran Šunić from the Department of Mathematics, Texas A&M University, USA, for his proof of Lemma 3, as well as Ernst Schulte-Geers from the German BSI, pointing out that in fact Lemma 3 was known long time ago from the paper of Flajolet and Odlyzko [13].

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. D. Gligoroski: "Narrow-pipe SHA-3 candidates differ significantly from ideal random functions defined over big domains", NIST hash-forum mailing list, 7 May 2010.

4. 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).
5. 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).
6. 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).
7. 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).
8. 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).
9. NIST FIPS PUB 180-2, ""Secure Hash Standard", National Institute of Standards and Technology, U.S. Department of Commerce, August 2002.
10. T. Dierks, E. Rescorla: "The Transport Layer Security (TLS) Protocol Version 1.2," RFC 5246, August 2008.
11. RSA Laboratories. PKCS #5 v2.1: "Password-Based Cryptography Standard", 5 October 2006.
12. H. Krawczyk, M. Bellare and R. Canetti: "HMAC: Keyed-Hashing for Message Authentication", RFC 2104, February 1997.
13. P. Flajolet and A. M. Odlyzko: "Random Mapping Statistics", EUROCRYPT (1989), pp. 329–354