Haraka – Efficient Short-Input Hashing for Post-Quantum Applications

Stefan Kölbl[1], Martin M. Lauridsen[1], Florian Mendel[2], and Christian Rechberger[1,2]

[1] DTU Compute, Technical University of Denmark, Denmark
[2] IAIK, Graz University of Technology, Austria
{stek,mmeh}@dtu.dk
{christian.rechberger,florian.mendel}@iaik.tugraz.at

Abstract. Many efficient cryptographic hash function design strategies have been explored recently, not least because of the SHA-3 competition. Almost exclusively these design are geared towards good performance for long inputs. However, various use cases exist where *performance on short inputs matters more*. An example is HMAC, and such functions also constituting the bottleneck of various hash-based signature schemes like SPHINCS, or XMSS which is currently under standardization. Secure functions specifically designed for such applications are scarce. In this paper, we fill this gap by proposing two short-input hash functions (or rather simply compression functions) exploiting instructions on modern CPUs that support the AES. To our knowledge these proposals are the fastest on modern high-end CPUs, reaching throughputs *below one cycle per hashed byte* even for short inputs while still having a *very low latency* of no more than 60 cycles. Under the hood, this results comes with several innovations.
First, we study whether the number of rounds for said functions can be reduced if collision resistance is not expected, but *only second-preimage resistance*. The conclusions is: only a little.
Second, since their inception AES-like designs allow for supportive security arguments by means of counting and bounding the number of active S-boxes. However, this ignores powerful attack vectors using truncated differentials, of which rebound attacks are a popular example. With our design, we develop for the first time a general tool-based method to include arguments against attack vectors using truncated differentials.

Keywords: Cryptographic hash functions, second-preimage resistance, AES-NI, hash-based signatures, post-quantum

1 Introduction

Cryptographic hash functions are often constructed with collision resistance in mind. Consider e.g. the SHA-3 competition, which involved a large part of the research community, where collision resistance was one of the main requirements. Sometimes, cryptographic functions are designed with collision resistance as the main or only requirement, see e.g. VSH [12].

This is however in contrast to a sizable and perhaps even growing set of applications that do require cryptographic hashing, but explicitly do not require collision resistance. Universal one-way hash functions (UOWHF) [4] are, in principle, candidate functions, but they will not suffice for many applications.

Consider as an example the proof for the HMAC construction. In its first version from the 90s it does require collision resistance from its hash function [3], but in later versions the collision resistance requirement is dropped in favor of milder requirements [2].

Another example are hash-based signature schemes originally introduced by Lamport [29]. Modern versions like XMSS [10], which is currently submitted as a draft to the IETF and features short signatures sizes, and the state-less scheme SPHINCS [7], are getting more and more attention as they are a robust candidate for quantum-resistant signature schemes, i.e. believed to be secure in the presence of hypothetical quantum computers. One of the main advantages of such schemes are that their security reduces to the security properties of the hash function(s) used.

All of the schemes mentioned require many calls to a hash function, but only process comparably short inputs. For instance in SPHINCS-256, some 500.000 calls to two hash functions are needed to reach a post-quantum security level of 128 bits. One of those functions compresses a 512-bit string to a 256-bit string and is used in a Merkle-tree construction, while the other maps a 256-bit string to a 256-bit string.

Secure short-input keyed hash functions also found applications in protecting against *hash flooding* denial of service attacks. This has been addressed with the SipHash [1] family, but the security requirements are much lower for this setting.

All the examples above share the fact that they do not require collision resistance from their underlying hash function(s), and also only process short inputs[3]. However, almost all hash functions designs are mostly geared towards good performance on long messages and, as we will show, perform rather poorly on short inputs.

Contributions. In this paper we explicitly consider second-preimage and preimage resistance as sole design goals for cryptographic hash functions, and pay attention on the short-input performance. We also aim to shed light on the following question: How much faster can a hash function become if collision resistance is dropped from the list of requirements? To that end we limit ourselves to one particular design strategy that is fairly well understood and scalable: AES-like designs.

On the practical side we propose two concrete compression function constructions and reach performances better than 1 cycle per byte (cpb) on various Intel architectures. Competitive designs are considerable slower than that, and even those speeds also only reached for long messages. Our proposals share strong similarities with the permutation AESQ that is used in the CAESAR candidate PAEQ [9].

[3] For HMAC, one of the two calls to the hash function used is always for a short input.

On the theoretical side, there are several contributions that come along with this new proposal: Firstly, we study if the number of rounds for said proposal can be reduced if collision resistance is not expected, but *only second-preimage resistance*. The conclusions is that only one round (5 instead of 6) can be saved.

Secondly, we describe new ways to bound the applicability of attacks. Traditionally, resistance of key-less constructions like cryptographic hash functions against collision attacks is almost solely based on arguments that are also found for keyed constructions like block ciphers. Two possibilities are:

1. A bound on the probability of the best differential trail, denoted p, is compared with the amount of degrees of freedom f. Assuming "ideal message modification" it is then argued that all available degrees of freedom are spent magically, i.e. the resulting probability is much higher, namely $p_2 = p \cdot 2^f$. The number of rounds is then chosen to make sure that p_2 is below some security requirement.
2. Again start with a bound on the probability of the best differential trail denoted by p. Choose a number of rounds r_1 to make sure that p is low enough for the required security level, i.e. $\log_2(1/p)$. Then add r_2 number of rounds for which are bypassed, based on experience and/or estimates, by so-called message modification technique. The resulting number of rounds $r = r_1 + r_2$ is then expected to help carry over the security expectations stemming from r_1 rounds in the keyed setting to r rounds in the key-less setting.

Examples can be found in various SHA-3 candidate submission like Grøstl [16], ECHO [5], Luffa [13], or the later proposed hash function constructions PHOTON [18] or SPN-Hash [11]. There are however various problems with these approaches. In particular, they do not consider truncated differential trails, and as such do not cover rebound attacks, and sometimes would require too many rounds to satisfy, especially with option (1) from above. Thus, approach (1) is too conservative and at the same time ignores (one of) the most power full attack vectors. And approach (2) is based on assumptions about message modification ability of an adversary (essentially solving systems of equations that get harder and harder with more rounds) while at the same time still ignoring (one of) the most power full attack vectors.

To somewhat remedy this situation we propose a way to include truncated differentials in the arguments of type (2) from above. Using this as security arguments for collision attacks is already novel, and we extend this method to also cover our new second-preimage attack vector.

Finally we remark at this point that both implementations and parts of the code used for the security analysis of Haraka are publicly available at [27].

2 Specification of Haraka

Haraka exists in two variants denoted Haraka-512/256 and Haraka-256/256 with signatures

$$\begin{aligned} &\text{Haraka-512/256} : \mathbb{F}_2^{512} \to \mathbb{F}_2^{256} \quad \text{and} \\ &\text{Haraka-256/256} : \mathbb{F}_2^{256} \to \mathbb{F}_2^{256}. \end{aligned} \tag{1}$$

For both variants we claim 256-bit (second)-preimage resistance respectively 128-bit in the presence of quantum computers, but we make no claims about other non-random properties.

The main components are two permutations denoted π_{512} and π_{256} on 512 bits and 256 bits, respectively. Both Haraka-512/256 and Haraka-256/256 rely on the well-known Davies-Meyer construction using a permutation with a feed-forward (applying the XOR operation) of the input. As such, they are defined as

$$\begin{aligned} &\text{Haraka-512/256}(x) = \mathsf{trunc}(\pi_{512}(x) \oplus x) \quad \text{and} \\ &\text{Haraka-256/256}(x) = \pi_{256}(x) \oplus x, \end{aligned} \tag{2}$$

where $\mathsf{trunc} : \mathbb{F}_2^{512} \to \mathbb{F}_2^{256}$ is a particular truncation function (described below).

2.1 Specification of π_{512} and π_{256}

In the following, we give our specification of the permutations used in Haraka. In Section 3 we give our security analysis of the constructions and, based on this, motivate our design choices in Section 4.3.

The constructions of π_{512} and π_{256} are *iterated*, thus applying a *round function* several times to obtain the full permutation. The permutations π_{512} and π_{256} operate on *states* which have the same size as respective inputs. Due to the similarity of the permutations, much of their description is common to both. In general, we let b denote the number of blocks of the state, so for π_{512} we have $b = 4$ while for π_{256} we have $b = 2$.

Denote the total number of rounds by T and denote by R_t the round with index $t = 0, \ldots, T-1$. The state *before* applying R_t is denoted S^t, and thus S^0 is the initial state. As both π_{512} and π_{256} use the AES round function, states are arranged in matrices of bytes, and we use subscripts to denote the column index, starting from column zero being the leftmost one. The state size is $4 \times 4b$ bytes, so 4×16 for π_{512} and 4×8 for π_{256}. When we talk about a *block*, we refer to a 16-byte string consisting of columns $x_{4i} \| \cdots \| x_{4i+3}$ for $i = 0, \ldots, b-1$.

When a stream of bytes is loaded into the state, the order is column first, such that the first byte of the input stream is in the first row of the first column, while the last byte of the stream is in the last row of the last column.

Let aes denote the parallel application of m AES rounds to each of the b blocks of the state. As such, for $t = 0, \ldots, T-1$, the round function for π_{512} is $R_t = \mathsf{mix}_{512} \circ \mathsf{aes}$ while for π_{256} it is $R_t = \mathsf{mix}_{256} \circ \mathsf{aes}$. Thus, in both cases, a single round consists of m rounds of the AES applied to each block of the state, followed by a linear mixing function. Round constants are injected via the **aes** operations (see below). The total number of rounds $T = 5$ while using $m = 2$ AES rounds for both Haraka-512/256 and Haraka-256/256.

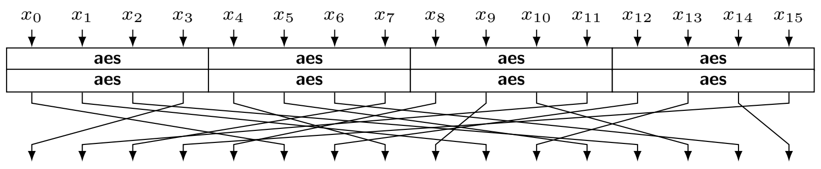
One of the differences between π_{512} and π_{256} are the specifics of the linear mixing used. In both cases, the mixing itself is comprised of simply permuting the state columns. For π_{512}, the sixteen columns of the state are permuted such that each output block contains precisely one column from each of the $b = 4$ input blocks. For π_{256} on the other hand we have $b = 2$ so we obtain the most even distribution of the columns by mapping two columns from each of the $b = 2$ input blocks to each of the $b = 2$ output blocks. More specifically, letting $x_0 \| \cdots \| x_{15}$ denote the columns for a state of π_{512}, the columns are permuted by $\textbf{mix}_{512}$ as

$$x_0 \| \cdots \| x_{15} \mapsto x_3 \| x_{11} \| x_7 \| x_{15} \| x_8 \| x_0 \| x_{12} \| x_4 \| x_9 \| x_1 \| x_{13} \| x_5 \| x_2 \| x_{10} \| x_6 \| x_{14}. \tag{3}$$

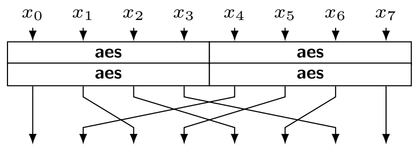
Likewise for π_{256} the eight columns denoted $x_0 \| \cdots \| x_7$ are permuted by $\textbf{mix}_{256}$ as

$$x_0 \| \cdots \| x_7 \mapsto x_0 \| x_4 \| x_1 \| x_5 \| x_2 \| x_6 \| x_3 \| x_7. \tag{4}$$

The round functions for both permutations are depicted in Figure 1.


x_0 x_1 x_2 x_3 x_4 x_5 x_6 x_7 x_8 x_9 x_{10} x_{11} x_{12} x_{13} x_{14} x_{15}
aes
aes

(a) For π_{512}



(b) For π_{256}

Fig. 1: Depictions of round functions R_t for π_{512} (a) and for π_{256} (b). Each x_i denotes a column of 4 bytes of the state.

Round Constants. For each AES call we use different round constants [4] via the round key addition. The constants are derived using a similar approach as in the CAESAR candidate Prøst [24]. Let p_i be the least significant bit of the ith digit after the decimal point of π, then the round constants are defined as

$$RC_j = p_{128} || \ldots || p_2 || p_1 \qquad \forall j = 0 \ldots 39. \tag{5}$$

[4] Note that in an earlier version the constants had strong symmetries which allowed more efficient attacks by exploiting those [23].

The AES layer aes_i uses $\{RC_{4i}, RC_{4i+1}, RC_{4i+2}, RC_{4i+3}\}$ in the case of π_{512} respectively $\{RC_{2i}, RC_{2i+1}\}$ for π_{256}. The constants are also given in Table 9 in the Appendix.

Truncation Function. Let $x \in \mathbb{F}_2^{512}$ be some input to the truncation function trunc. Then $\mathsf{trunc}(x)$, which is used in Haraka-512/256, is obtained as concatenating two columns from each block: The least significant two from the first two blocks; the two most significant columns from the last two blocks. As such

$$\mathsf{trunc}(x_0 \| \cdots \| x_{15}) = x_2 \| x_3 \| x_6 \| x_7 \| x_8 \| x_9 \| x_{12} \| x_{13}. \tag{6}$$

3 Security against Truncated Differential Attacks

The three most commonly defined security requirements for a cryptographic hash functions are

- **Preimage resistance:** Given an output y it should be computationally infeasible to find any input x such that $y = H(x)$,
- **Second-preimage resistance:** Given $x, y = H(x)$ it should be computationally infeasible to find any $x' \neq x$ such that $y = H(x')$, and
- **Collision resistance:** Finding two distinct inputs x, x' such that $H(x) = H(x')$ should be computationally infeasible.

For any ideal hash function generic attacks exist, which are able to find a (second-)preimage with a complexity of 2^n and collisions with a complexity of $2^{n/2}$, where n is the output size of the hash functions in bits. Quantum computers can improve upon this by using Grover's algorithm [17] to further reduce the complexity of finding a (second-)preimage to $2^{n/2}$. It is also known that this is the optimal bound for quantum computing.

In the following sections we discuss common attack vectors which will aid in choosing appropriate parameters for Haraka to achieve the desired security properties. We focus on the second-preimage resistance, since the main applications of Haraka do not require collision resistance.

3.1 Preliminaries

Differential cryptanalysis is one of the most powerful tools in evaluating the security of cryptographic hash functions. It is also a very natural attack vector as both collision and second-preimage resistance require the attacker to efficiently find two distinct inputs yielding the same output.

Definition 1. *A* differential trail *Q is a sequence of differences*

$$\alpha_0 \xrightarrow{R_0} \alpha_1 \xrightarrow{R_1} \cdots \xrightarrow{R_{T-1}} \alpha_T \tag{7}$$

in the states for the application of the function on two distinct inputs.

Definition 2. *The* differential probability *of a differential trail Q is defined as*

$$\mathrm{DP}(Q) = \Pr(\alpha_0 \to \alpha_1 \to \ldots \to \alpha_T) = \prod_{t=0}^{T-1} \Pr(\alpha_t \to \alpha_{t+1}) \tag{8}$$

and gives the probability, taken over random choices of the inputs, that the pair follows the differential trail. The last equality holds if we assume independent rounds.

The AES round function uses the SubBytes, ShiftRows and MixColumns operations, which we also denote as SB, SR and MC, respectively. For our further analysis we are also interested in how truncated differentials [26] propagate through MixColumns. One property of MixColumns is that the branch number is 5, i.e. if we have one active byte at the input we will always get four active bytes at the output. In general if an input column contains a active bytes to MixColumns, then the probability of having b active bytes in the corresponding output column, where $a + b \geq 5$, can be approximated with $2^{(b-4)8}$.

Differential Trails. One way to estimate $\mathrm{DP}(Q)$ for the best trail is to count the minimum number of active S-boxes. As the maximum differential probability for the AES S-box is 2^{-6} this allows to give an upper bound on $\mathrm{DP}(Q)$. While the number of active S-boxes gives a good estimate for the costs of an attack in the block cipher setting, this is only partially true for cryptographic hash functions. Consider a pair of inputs $(x, x \oplus \alpha)$ as input to a non-linear function, like the AES S-box, then $S(x \oplus K) \oplus S(x \oplus \alpha \oplus K) = \beta$ holds only with a certain probability if the key K is unknown. This can be very useful in the block cipher settings, where it gives a bound on the probability of the best differential trail.

In the case of hash functions there is no secret key and an attacker has full control over the input bits. This allows him to choose the pair $(x, x \oplus \alpha)$ such that $S(x) \oplus S(x \oplus \alpha) = \beta$ holds with probability 1. The limit of this approach is only restricted by the number of free and independent values, referred to as *degrees of freedom*. This means that the probability of a differential trail can be very low and contain many active S-boxes, but if the conditions are easy to fulfill and the attacker has enough degrees of freedom an attack can be very efficient.

A popular technique to count the number of active S-boxes for AES-based designs is based on *mixed integer linear programming* (MILP) [31,34]. The basic idea is to express the restrictions on the differences given by the round transformations as linear equations and generate a optimization problem which can be solved with any MILP optimizer, e.g. Gurobi [19] or CPLEX [21]. We use this technique later to find the minimum number of active S-boxes for Haraka, which aids us in choosing our parameters.

3.2 Capabilities of the Attacker

One of the main difficulties in the design of hash functions is to estimate the security margin one expects against a powerful attacker. While bounds on the probability of differential trails can be useful in the block cipher setting, they have little meaning for hash functions. There is no secret input and the attacker can freely choose the messages. This degrees of freedom can be used to solve conditions imposed by a differential trail and lead to surprisingly efficient attacks.

This was partially addressed in the design of Fugue [20] and SPN-hash [11]. The former assumes that an attacker can improve the probability of a differential trail by using the degrees of freedom directly, i.e. if one has f degrees of freedom the probability can be improved by 2^f. SPN-hash assumes the attacker can bypass r_2 rounds by estimates from existing attacks and the total number of rounds is then given by $r = r_1 + r_2$, where r_1 is chosen such that the probability of the best differential is low enough for the required security level. A mayor drawback of this approaches is that they do not resemble the capabilities of an attacker in practice, which can either lead to too conservative estimates while also ignoring important attack vectors.

The most powerful collision attacks on AES-based hash functions, like the rebound attack [30], use truncated differentials combined with a clever use of the degrees of freedom to reduce the attack complexity. Arguing security against these type of attacks is a difficult task, as one has to estimate the limits of an attacker to use the available degrees of freedom in a smart way to reduce the attack complexity. Additionally, in the second-preimage scenario the attacker has much less control as the actual values of the state are fixed and the conditions are instead solved by carefully choosing the differential trails. In the following we propose a new method to better bound the capabilities of an attacker in practice under reasonable assumptions.

Truncated Differentials. While the MILP model to count the number of active S-boxes already uses truncated differentials, it does not cover the costs of propagation of those. When an attacker tries to utilize a truncated differential the transitions through MixColumns are probabilistic and, if not controlled by the attacker, will determine the attack complexity similar to the outbound phase in the rebound attack.

Independent of the number of rounds, the best we can achieve is a security level of 256 bits, as an attacker can always use a (fully active) truncated differential with probability ≈ 1 and the probability that this gives a valid second-preimage is 2^{-256}.

Utilizing Degrees of Freedom. The previous model still ignores the fact that a powerful attacker can utilize the available degrees of freedom to reduce the attack complexity. To take this into account we assume the attacker is able to use all degrees of freedom in an *optimal way*, i.e. the attacker has an algorithm to solve any condition in constant time, as long as there are enough independent degrees of freedom left.

Without any further restrictions we can not achieve any level of security in this model, as the attacker can always use a truncated differential which is active in all bytes and has a probability of 1 and then use the degrees of freedom to guarantee that $f(x) \oplus f(x \oplus \alpha) = 0$. However, it is very unlikely that an attacker can utilize the degrees of freedom unrestricted over many rounds as, after already two rounds each byte of the state depends on all the others in AES-like designs.

We suggest a more restrictive model of the attacker, in which he is still given all the previous capabilities but can only solve the conditions for q consecutive rounds of the cipher. This means, the attacker chooses a state S^k and then is allowed to solve any conditions for $S^{k-q}, \ldots S^{k+q}$ in constant time, as long as there are still degrees of freedom available. The remaining conditions which can not be solved form our security level. We can formulate this as a MILP problem with the goal to find the lowest attack complexity over all possible states S^k (for more details and the application to Haraka see Section 4.1).

This model for truncated differential attacks resembles how collision attacks on cryptographic hash functions *actually* work in practice. The attacker can control how the differences propagate over a part of the state and tries to minimize the conditions in the remaining rounds [30,35]. The currently best known attacks on AES-based hash functions utilize the degrees of freedom for up to three (AES) rounds to reduce the complexity of an attack [33,22]. These results can not be carried over directly to our construction as we compose our state of four individual AES states.

However, in the collision setting the attacker can choose both the values and differences freely, while the second-preimage scenario is much more restrictive and less degrees of freedom are available. We suggest $q = 2$, allowing our idealized attacker to cover a generous 4 rounds with the degrees of freedom to have a comfortable security margin.

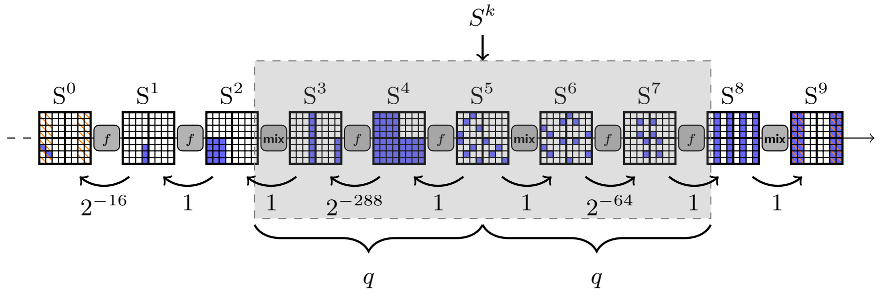


Fig. 2: Truncated model utilizing degrees of freedom for $T = 3, m = 2, q = 3$. For finding a collision the attacker would have full control over the middle rounds. As there are only 352 conditions the attack costs for an idealized attacker would be 2^{16}. The four AES states A, B, C, D are ordered clockwise starting at the top left as A, C, D, B.

4 Analysis of Haraka

In the following we give the security claims for Haraka and the security analysis which lead to the proposed parameters.

Security Claims. We claim second-preimage resistance of 256-bits for Haraka. As will been seen later in the paper, for only one additional round (a performance penalty of around 20%) we claim 128-bit of collision resistance. We make no claims about near-collision or other generalizations of this property and distinguishers of the underlying permutation, because such properties do not seem to be needed in applications like HMAC or hash-based signature schemes. Overall, this leads to a conjectured post-quantum security level of 128-bits against both collision and second-preimage attacks.

Non-randomness that might slightly speed-up second-preimage attacks is not excluded by our models and bounds, but we conjecture this to be negligible. To support our conjecture, consider as an example the slight speed-up of second-preimage attacks [14,15] on the SHA-3 candidate Hamsi [28] which use a very strong non-random property of the compression function. No such strong property seem likely to exist for our proposals.

4.1 Second-Preimage Resistance

Generic Attacks. As the output size $n = 256$, a generic attack exists with a complexity of 2^{256} resp. 2^{128} on a quantum computer.

For iterative hash functions, a generic attack exists which improves upon the naïve brute force approach [25]. However, this attacks requires long messages and are not applicable to our construction.

Basic Differential Second-Preimage Attack for Weak Messages. For finding a second-preimage the attacker can use a differential trail Q leading to a collision, i.e. $f(x \oplus \alpha) = y$. However, as the values of the state are fixed by the output Y, all differentials trails hold either with probability 1 or 0. For a random message the probability that an attacker succeeds is bounded by $\mathrm{DP}(Q)$ and if Q does not give a second-preimage for y then the attacker has to try a different trail $Q' \neq Q$.

Counting the number of active S-boxes gives a bound on the maximum value of $\mathrm{DP}(Q)$ and can give some insights on the security. We consider both the number of active S-boxes for the permutation and when the permutation is used in the DM-mode. As some of the output is truncated, this can potentially reduce the number of active S-boxes and has to be taken into account.

For Haraka-512/256 the best differential trails have a probability of $\mathrm{DP}(Q) = 2^{-780}$ respectively 2^{-804} for trails leading to a collision when used in DM-mode. Correspondingly for Haraka-256/256, those probabilities are 2^{-480} and 2^{-630} respectively. For the number of active S-boxes for Haraka-512/256 and Haraka-256/256 see Table 1. Note that this corresponds to previous work that studied second-preimage attacks for MD4 [37] and SHA-1 [32].

Truncated Model. We used the model presented in Section 3.2 to find the optimal number of rounds for our constructions. We denote the input column j to $\mathsf{MixColumns}$ (resp. $\mathsf{SubBytes}$) in round r as MC_j^r (SB_j^r). We remark that in the

Table 1: Lower bound on the number of active S-boxes in a differential trail for the permutations used in Haraka, for the permutation when used in DM mode and for trails leading to a collision when used in DM mode. Table 10 in the Appendix gives the numbers for a wider choice of parameters.

	Permutation	DM-mode	DM-mode (coll.)
Haraka-256/256	80	80	105
Haraka-512/256	130	128	134

following, consider the number of rounds T and the number of AES rounds per round m as variables.

We define the costs for an attacker to follow a truncated differential, starting at state S^k as

$$C_{\text{Trunc}} = \sum_{r=0}^{T\cdot m} \sum_{j=0}^{4b} C^r_{\mathsf{MC}_j} \tag{9}$$

where

$$\forall r : 0 \le r < k, \forall j : 0 \le j < 4b \quad : \quad C^r_{\mathsf{MC}_j} \ge (4 - \sum_{i=0}^{3} \mathsf{MC}^r_{i+4j}) \cdot 8$$

$$\forall r : k \le r \le T \cdot m, \forall j : 0 \le j < 4b \quad : \quad C^r_{\mathsf{MC}_j} \ge (4 - \sum_{i=0}^{3} \mathsf{SB}^r_{i+4j}) \cdot 8$$

An additional requirement is that the input and output difference in the non-truncated part of the state are equal to get a valid second-preimage, i.e. $x \oplus \alpha = \Delta\pi_{512}(x \oplus \alpha)$ which we denote as $C_{\text{Collision}}$. The optimization goal is

$$\textbf{minimize:} \quad C_{\text{Collision}} + C_{\text{Trunc}}. \tag{10}$$

The requirements for Haraka are that each attack under this model costs at least 2^{256}. We use the previous model to determine the security level of our construction for different number of rounds T and m. For every parameter set we use the MILP model to find the lowest attack costs by searching over all possible starting states S^k. The results can be found in Table 2.

If we don't allow the attacker to utilize any degrees of freedom $T = 4, m = 2$ would be sufficient for Haraka-512/256 resp. $T = 2, m = 2$ for Haraka-256/256. (see Table 2). However, as discussed in the previous section this approach would be very optimistic. Taking into account the assumptions we make on the capabilities of an attacker utilizing the degrees of freedom at least 5 rounds are required (see Table 3). The solving time increases quickly with the number of rounds and for the standard parameters ($T = 5$, $m = 2$, $q = 2$) it takes around 17 minutes[5] to find the lower bound for an attack for all possible starting points S^k.

[5] Using Gurobi 6.5.0 (linux64), Intel(R) Core(TM) i7-4770S CPU @ 3.10GHz, 16GB RAM

Table 2: Bounds on the best attack in our truncated setting without utilizing degrees of freedom over multiple rounds.

(a) Security for π_{512}

T	m = 1	2	3	4	5
1	0	32	48	64	64
2	32	128	96	96	96
3	48	192	176	192	192
4	112	256	256	256	256
5	128	256	256	256	256
6	208	256	256	256	256
7	224	256	256	256	256

(b) Security for π_{256}

T	m = 1	2	3	4	5
1	0	0	0	0	128
2	0	256	176	192	192
3	184	256	240	256	256
4	176	256	256	256	256
5	256	256	256	256	256
6	240	256	256	256	256
7	256	256	256	256	256

Table 3: Security bounds on the best attack in our truncated setting utilizing additional degrees of freedom over q rounds for π_{512} and π_{256} using $m = 2$. Non-bold entries do not obtain the generic bounds and in the case of collision resistance values above 128 would be outperformed by generic attacks.

(a) Second-preimage π_{512}

T	1	2	3	4	5	6
$q = 1$	0	96	144	**256**	**256**	**256**
$q = 2$	0	0	96	128	**256**	**256**
$q = 3$	0	0	0	96	128	**256**

(b) Collision π_{512}

T	1	2	3	4	5	6
$q = 1$	0	48	136	**176**	**256**	**256**
$q = 2$	0	0	40	96	**168**	**256**
$q = 3$	0	0	0	32	96	**160**

(c) Second-preimage π_{256}

T	1	2	3	4	5	6
$q = 1$	0	176	192	**256**	**256**	**256**
$q = 2$	0	128	128	192	**256**	**256**
$q = 3$	0	0	128	128	192	**256**

(d) Collision π_{256}

T	1	2	3	4	5	6
$q = 1$	0	**168**	**176**	**240**	**256**	**256**
$q = 2$	0	64	112	**160**	**256**	**256**
$q = 3$	0	0	64	112	**176**	**256**

In Figure 2 we give an example how this attack model works in practice. The attacker starts in this case at S^5 and can control $q = 3$ rounds in both directions. When finding a collision the attacker has control over the full state, therefore he has enough degrees of freedom available to fulfill the conditions for the transitions through MixColumns. The only remaining part is the transition in the first round which happens with a probability of 2^{-16}.

Other Preimage and Second-preimage Attacks. Relevant are meet-in-the-middle (MITM) attacks that are a popular attack vector for preimage attacks and in turn also suitable to find second-preimages. Perhaps the most relevant literature pointer here is an attack on the 5-round reduced output transformation of Grøstl [36]. We did not find a way to extend attacks, even in relaxed settings, beyond $T = 4$ rounds.

4.2 Collision Resistance

While we do not require collision resistance, we would still like to discuss the security level of our construction with respect to this criteria in the following. Similar to our arguments for second-preimage security we can apply our truncated model for finding collisions. The best collision attacks on AES-based hash functions are based on the rebound attack and covered by our model. However, for finding a collision an attacker can freely choose the complete internal state and not only the differences, which translates to more degrees of freedom. Therefore the expected security level is lower for the same number of rounds (see Table 3).

Nonetheless, the best generic attack also has a lower complexity of 2^{128}, compared to the second-preimage case, which might suggest that one only requires 2^{128} in our truncated model. However, it is likely that the more relaxed collision setting allows to exploit this after using up all degrees of freedom. Consequently, we suggest to also aim for a security level of 2^{256} in our truncated model, which requires adding one round for Haraka-512/256.

4.3 Design Choices

In the following we interpret our security analysis which lead to the proposed parameters and design choices. We remind again that T denotes the number of rounds of either π_{512} or π_{256}, and m denotes the number of AES rounds applied to each of the b blocks in each round.

Round Parameters T and m. One of the first questions which arise is how the number of AES rounds and frequency of mixing the individual states influences the security bounds. From our analysis there is a strong indication that $m = 2$ is an optimal choice (see Table 1), as it gives the best trade-off between number of active S-boxes and the total number of required AES rounds Tmb. We propose $T = 5$, as this gives the required security parameters in our truncated model even when assuming a very powerful attacker controlling more rounds than the best known attacks are capable of.

Mixing Layers. For the mixing layer, a variety of choices were considered. The main criteria here were that the layer should be efficiently implementable (see Section 5.3), while still contributing to a highly secure permutation. Other potential candidates for the mixing layer are discussed in Appendix C. With respect to our criteria, for most choices of T and m, using the proposed $\textsf{mix}_{512}$ and $\textsf{mix}_{256}$ give a significant higher number of active S-boxes compared to other approaches.

Truncation Pattern for Haraka-512/256. There are many possible choices for the truncation pattern for Haraka-512/256. In our analysis we consider truncation patterns which truncate row-wise or column-wise, as these are most efficient to implement, due to the way words are stored in memory. For example, a single column makes up 4 adjacent bytes in memory, thus allowing for more efficient access. A row can likewise be accessed by first transposing the block. The pattern we chose is taking the two least significant columns of the first two states and the two most significant columns of the last two states. This compared favorably to row-wise patterns or choosing the same two columns from each state.

5 Implementation Aspects and Performance

The compression functions Haraka-512/256 and Haraka-256/256 have been designed with particular target platforms in mind. Specifically, we consider architectures with hardware acceleration for the AES. To that end, we assume the existence of an instruction pipeline that can execute a single round of the AES with a latency of $L_{\textsf{aes}}$ cycles and an inverse throughput of $T_{\textsf{aes}}^{-1}$ instructions per cycle. Table 4 gives the latencies and inverse throughputs for a single round of the AES on our target platforms. We remark that our Haswell test machine has an i7-4600M CPU at 2.90GHz; the Skylake machine has an i7-6700 CPU at 3.40GHz. We furthermore expect Haraka to be efficiently implementable on ARMv8 due to the support of AES instructions. We remark that the Turbo Boost technology has been switched off for all our performance measurements.

Table 4: Latency and inverse throughput for AES instructions on target platforms

Architecture	$L_{\textsf{aes}}$	$T_{\textsf{aes}}^{-1}$
Haswell	7	1
Skylake	4	1

Naturally, when encrypting a single block with the AES, one must wait $L_{\textsf{aes}}$ cycles each time the block is encrypted for one round. However, if the inverse throughput $T_{\textsf{aes}}^{-1}$ is low compared to $L_{\textsf{aes}}$, and if additional *independent* data blocks are available for processing, one can use this data independency to better utilize the AES pipeline. Thus, in theory, if using $k = L_{\textsf{aes}} \cdot T_{\textsf{aes}}^{-1}$ independent blocks for the AES, one can encrypt each of those blocks for a single round in just $(k-1) \cdot T_{\textsf{aes}}^{-1} + L_{\textsf{aes}}$ cycles, while m rounds of the AES can be completed for all k blocks in just $(k-1) \cdot T_{\textsf{aes}}^{-1} + L_{\textsf{aes}} \cdot m$ cycles, as illustrated in Figure 3.

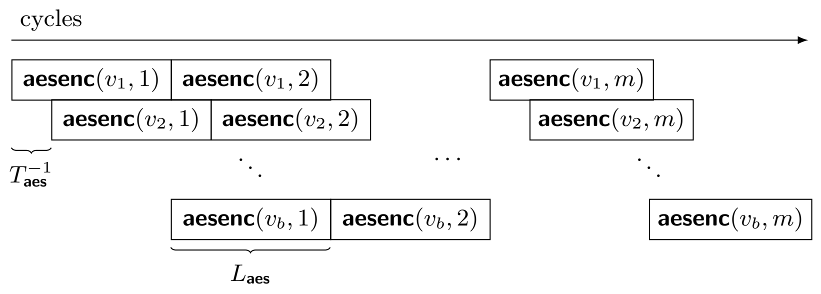

cycles
$\mathbf{aesenc}(v_1, 1)$
$\mathbf{aesenc}(v_1, 2)$
$\mathbf{aesenc}(v_1, m)$
$\mathbf{aesenc}(v_2, 1)$
$\mathbf{aesenc}(v_2, 2)$
$\mathbf{aesenc}(v_2, m)$
T_{aes}^{-1}
$\mathbf{aesenc}(v_b, 1)$
$\mathbf{aesenc}(v_b, 2)$
$\mathbf{aesenc}(v_b, m)$
L_{aes}

Fig. 3: Pipelined AES instructions. A box $\mathbf{aesenc}(v, i)$ denotes the application of the ith AES round to a block v.

5.1 General Construction Considerations

With the above observations in mind, and with the numbers of Table 4, it was clear from the beginning that our states for Haraka-512/256 and Haraka-256/256 should use $b > 1$, because otherwise using the AES instruction on the state would lead to non-optimal pipeline utilization. In Appendix C, we discuss considerations regarding the mode of operation for the permutations used in our compression function designs, and how we ended with the choice of using a permutation in a Davies-Meyer style operation, but where for Haraka-512/256 the output is truncated to obtain the 2 : 1 compression ratio.

5.2 Multiple Inputs

As described above, the theoretically optimal choice of state blocks would equal $L_{\mathsf{aes}} \cdot T_{\mathsf{aes}}^{-1}$. However, as detailed, the Haraka variants use varying number of blocks. To that end, we consider for both Haraka-512/256 and Haraka-256/256 the parallel application of the corresponding function to *multiple inputs*, assuming that such are available for processing. For example, if $k = L_{\mathsf{aes}} \cdot T_{\mathsf{aes}}^{-1} = 7$, with a state size of $b = 4$ blocks, one could process two *independent* inputs x and x' in parallel, thus artificially extending the state to $b = 8$ blocks, allowing better pipeline utilization. We denote the number of parallel inputs processed by P. For each of our constructions and target platforms, there will be an optimal choice of P which allows good AES pipeline utilization while, at the same time, keeping the full context in low-level cache.

5.3 Implementation of Linear Mixing

Consider the case where $P = 1$, i.e. when considering a single input. Even if the number of blocks in the state is less than $L_{\mathsf{aes}} \cdot T_{\mathsf{aes}}^{-1}$, a number of the instructions used for the linear mixing can be hidden after the **aes** operation. For example, while the instruction to encrypt the mth AES round of R_t is still being executed for one or more blocks, while other blocks have already finished, instructions pertaining to the mixing of the finished blocks can be executed while the AES instructions for the remaining blocks are allowed to finish. To that end, more so than otherwise, choosing instructions for the linear mixing layer with low latency and high throughput is important.

For the implementation of $\textbf{mix}_{512}$ and $\textbf{mix}_{256}$, we make use of the punpckhdq and punpckldq instructions. On both Haswell and Skylake, those instructions have a latency of 1 clock cycle and an inverse throughput of 1 instruction/cycle. The $\textbf{mix}_{512}$ $\textbf{mix}_{256}$ approaches have the property that each output block contains columns from each of the input blocks. This is not the case for the other approaches to linear mixing considered (see Appendix C), and indeed our analysis show that our approach yields better security properties.

In the case of Haraka-512/256 where the state has $b = 4$ blocks, $\textbf{mix}_{512}$ uses eight instructions in the mixing layer. In the case of Haraka-256/256, where the state size is $b = 2$ blocks, the implementation of $\textbf{mix}_{256}$ can be made with just one application of each of punpckhdq and punpckldq.

Table 5: Performance of Haraka-512/256 and Haraka-256/256 (in cpb) on Haswell and Skylake as a function of the number of rounds T. In all cases, the number of AES rounds per round is fixed to $m = 2$. The numbers are taken as the minimum over choices of P in the range $P = 1, \ldots, 16$.

	Haraka-512/256		Haraka-256/256	
T	Haswell	Skylake	Haswell	Skylake
1	0.27	0.16	0.21	0.13
2	0.59	0.31	0.44	0.25
3	0.86	0.43	0.66	0.38
4	1.16	0.58	0.89	0.50
5	**1.42**	**0.71**	**1.10**	**0.63**
6	1.69	0.83	1.30	0.75
7	1.96	0.97	1.51	0.88
8	2.22	1.11	1.71	1.00

5.4 Discussion of Performance

In the following, we discuss the performance results of the Haraka construction, both with respect to varying parameters for the construction itself, but also in the light of other similar constructions of different kinds.

Table 5 gives performance figures for Haraka-512/256 and Haraka-256/256, using $m = 2$ but with a varying number of rounds, on both the Haswell and Skylake micro-architectures. As is evident, with $T = 5$, we obtain a performance as high as 0.71 cpb for Haraka-512/256 on Skylake, while on Haswell we obtain 1.42 cpb. For Haraka-256/256, the corresponding numbers with $T = 5$, as is our parameter choice, are 0.63 cpb and 1.10 cpb, respectively.

It is interesting to compare against the corresponding functions H and F from the SPHINCS-256 construction [7], which have identical functional signatures and similar design criteria. Here, AVX2 implementations utilizing 8-way parallelization

Table 6: Performance comparison for various primitives on the Haswell and Skylake platforms. The primitives are divided in three categories: AES-NI-based hash function implementations; Others (including hash functions, stream ciphers and PRFs); and Compression functions. For the latter, some implementations are directly derived from the code of the corresponding hash function but reducing it to a compression function taking a 64-byte input. Numbers for entries marked by a † are from eBACS [6] (other numbers are obtained by running SUPERCOP). For Haraka performance numbers, we use $P = 1$ for a fairer comparison. For the double-block constructions marked (IV), we mean variants where the chaining input is used as additional message input.

		Haswell		Skylake	
Type	Primitive	64-byte	4-KiB	64-byte	4-KiB
AES-NI	ECHO-256	20.38	4.78	18.30	4.26
	Grøstl-256	32.22	9.73	28.92	8.50
	Fugue-256	58.19	14.86	53.38	14.26
Other	ChaCha12	3.69	0.80	3.47	0.79
	Salsa20†	6.62	1.48	6.38	1.35
	SipHash-2-4	2.12	1.42	2.16	1.41
	SipHash-4-8	3.88	2.77	3.97	2.83
	BLAKE2S	5.30	5.45	4.77	4.92
	BLAKE2B	10.88	5.11	7.92	3.47
	Skein-512-256	13.59	5.89	13.08	5.50
	BLAKE-256	16.67	7.74	15.19	6.93
	SHA-256	28.75	12.75	19.47	7.80
	Keccak[$c = 512$]	24.28	10.51	20.31	9.22
	JH-256	28.64	14.05	28.20	13.93
	LANE-256	89.41	29.96	70.28	23.40
	Whirlpool	68.72	35.91	50.81	27.39
Compression	**Haraka-256/256**	**1.29**	–	**0.72**	–
	Haraka-512/256	**1.78**	–	**0.97**	–
	SPHINCS-256-F	11.76	–	11.22	–
	SPHINCS-256-H	11.58	–	10.97	–
	Hirose (IV)	8.38	–	6.47	–
	AbreastDM (IV)	13.45	–	12.97	–
	TandemDM (IV)	16.33	–	12.91	–
	Hirose	16.64	–	13.05	–
	Grøstl-256	24.20	–	19.64	–
	AbreastDM	26.77	–	25.73	–
	TandemDM	32.44	–	25.69	–
	Fugue-256	53.23	–	49.72	–
	SHA-256	14.97	–	12.56	–
	Keccak[$c = 512$]	27.94	–	19.67	–
	Whirlpool	29.56	–	21.28	–
	LANE-256	30.77	–	23.66	–

(i.e. using $P = 8$) lead to a performance of 1.63 cpb for their H function and 1.64 cpb for their F function on Haswell. By employing the availability of AVX-512 on Skylake, it is reasonable to assume that on this platform, the SPHINCS-256 functions would have their performance doubled. Meanwhile, we remark that even under this assumption, in both the cases of Haswell and Skylake, our functions perform favorably in comparison to those of SPHINCS-256.

Table 7: Number of calls to the AES round function, and number of calls to the AES key schedule assistant function, for AES-NI-based primitives.

Primitive	Key length	# **aesenc**	# **aeskeygenassist**
Haraka-512/256	–	40	0
Haraka-256/256	–	20	0
ECHO-256	–	256	0
Grøstl-256	–	240	0
Fugue-256	–	198	0
Hirose	256	112	52
Hirose (IV)	256	56	26
TandemDM	256	112	104
TandemDM (IV)	256	56	52
AbreastDM	256	112	104
AbreastDM (IV)	256	56	52

In some applications, it is likely that several inputs will *not* be available for processing in parallel. To that end, it is interesting to compare the performance for Haraka using $P = 1$ to the corresponding functions from SPHINCS-256. In this case, from Table 6 we see that Haraka-256/256 performs very well with 1.29 cpb and 0.72 cpb on Haswell and Skylake, respectievly, while the numbers for Haraka-512/256 are 1.78 cpb on Haswell and 0.97 cpb on Skylake. From benchmarking the corresponding SPHINCS-256 functions on the same machines, using AVX2 implementations and $P = 1$, we obtain a performance of 11.58 cpb on Haswell and 10.97 cpb on Skylake for their H function, and respectively 11.76 and 11.22 cpb for their F function on Haswell and Skylake respectively. Thus, when several inputs are not available to draw on for parallelization, our Haraka-512/256 and Haraka-256/256 constructions perform at least 6.5 times better than those from SPHINCS-256 on our target platforms.

In Table 6 we compare the performance not only with the SPHINCS-256 functions, but also a wide selection of other functions that are to an extent similar to our Haraka construction. As an example, we compare against a wide range of *generic* hash functions including some with implementations based on AES-NI, as well as all SHA-3 finalists. We also compare against ChaCha12 which is also used in SPHINCS-256, as well as the SipHash PRF construction. Furthermore, we compare against a range of other compression function constructions including the Hirose, AbreastDM and TandemDM so-called double-block constructions which were implemented using AES-256 as the underlying block cipher. In some cases, generic hash functions were converted directly to compression functions by simply modifying the existing code to strip away the overhead associated with supporting arbitrary input sizes. We remark that while generic hash functions are designed to accept inputs of arbitrary lengths, they obtain their stable performance only for inputs much longer than 64 bytes, due to their overhead associated with e.g. initialization, as is also evident from the table. To that end, we give their performance for both the short input of 64 bytes (thus matching the input size of Haraka-512/256), as well as a long input of 4 KiB. We note that even comparing against the performance for long messages of 4 KiB, the performance compares favorably to the Haraka constructions. We remark that for all entries in Table 6 that are based on AES, we give in Table 7 a count on the number of calls to the AES round function as well as to the instruction for performing one round of the AES key schedule.

5.5 Performance in SPHINCS

While the previous performance figures provide a good comparison between the functions itself, the actual performance figures relevant for a hash-based signatures scheme are the costs for key-generation, signing and verifying a signature. The total costs for this operations are difficult to derive by only looking at the performance of the short-input hash function.

We therefore modified the optimized AVX implementation of SPHINCS given in [7], by replacing all calls to SPHINCS-256-F and SPHINCS-256-H by Haraka-256/256 resp. Haraka-512/256. Parallel calls to these functions are processed to the same extend, using 8 calls at the same time, and no further optimizations have been applied. As can be seen in Table 8, the current performance gains by using Haraka are between $\times 1.5$ to $\times 2.8$, depending on the platform and operation.

Table 8: Comparison of the AVX implementation of SPHINCS with our implementation using Haraka. All numbers are given as the total number of cycles required and measured using SUPERCOP.

	Haswell		Skylake	
	SPHINCS-256	Haraka	SPHINCS-256	Haraka
Key Generation	3 295 808	2 060 864	2 839 018	1 426 138
Signing	52 249 518	34 938 076	43 517 538	23 312 354
Verify	1 495 416	695 222	1 291 980	452 066

6 Conclusion and Remarks on Future Work

Together with in-depth implementation considerations of modern CPU design, we presented the seemingly fastest proposal for compression/short-input hashing on such platforms, with a performance of less than 1 cpb on a Skylake desktop CPU, both with and without parallelization across multiple inputs. Despite exploring a larger design-space, the design ended up having strong similarities with the permutation AESQ that is used in the CAESAR candidate PAEQ [9]. Our implementations for Haraka, as well as code for security analysis, is available to check out at [27].

We remark that our design is optimized for short inputs, and long-message performance is out of scope for this paper. Nevertheless, when our compression function is put into one of the well-understood domain extension methods like the Sponge construction [8], the resulting long-message speed would still be below 2 cycles per byte, and hence would also be very competitive.

In contrast to competing designs that were the fastest so far, we can give arguments in favor of its security against important classes of attacks that goes beyond statements such as: "Nobody seems to be able to break more rounds". As a novelty, we also consider attacks using truncated differentials that are bounded, and thus for the first time also go beyond the argument using bounds on the number of active S-boxes. This, of course, does not rule out attacks outside of the models that we consider, and hence more cryptanalysis is needed to establish more trust in the proposal. Particularly, MITM-style attacks will be an important attack vector to consider, as no good bounding mechanisms are available yet.

Returning to our initial question: How much faster can a hash function become if collision resistance is dropped from the list of requirements? In our proposal we drop from 6 rounds to 5 rounds and still retain security against second-preimage attacks. We can conclude that the performance gains are rather limited for the class of hash function design strategies that we consider, namely AES-like designs. This particularly holds when aiming at pre-quantum security levels that are higher than those for collision resistance, namely 256 bits rather than 128 bits. The reason why aiming at higher security levels makes sense is that there is evidence that (at least for generic attacks) the post-quantum security levels will in both cases be 128 bits. Of course, this argument does not consider non-generic attacks that use capabilities of hypothetical quantum computers, and we leave investigations in this direction as future work.

Acknowledgments

The authors would like to thank Andreas Hülsing and Peter Schwabe for their discussions on the XMSS and SPHINCS hash-based signature schemes, in particular in relation to the applicability of Haraka in those designs. We also want to thank Shay Gueron and Daniel J. Bernstein for insightful discussions on efficient use of the AES-NI instructions. Additionally, we want to thank Jérémy Jean for pointing out a problem with the choice of constants in an earlier version of this paper. Lastly, we would also like to thank the anonymous reviewers who helped improve the quality of this paper

References

1. Aumasson, J., Bernstein, D.J.: Siphash: A fast short-input PRF. In: Galbraith, S.D., Nandi, M. (eds.) Progress in Cryptology - INDOCRYPT. Lecture Notes in Computer Science, vol. 7668, pp. 489–508. Springer (2012)

2. Bellare, M.: New Proofs for NMAC and HMAC: Security without Collision Resistance. J. Cryptology 28(4), 844–878 (2015)
3. Bellare, M., Canetti, R., Krawczyk, H.: Keying Hash Functions for Message Authentication. In: CRYPTO '96. pp. 1–15
4. Bellare, M., Rogaway, P.: Collision-Resistant Hashing: Towards Making UOWHFs Practical. In: CRYPTO '97. pp. 470–484 (1997)
5. Benadjila, R., Billet, O., Gilbert, H., Macario-Rat, G., Peyrin, T., Robshaw, M., Seurin, Y.: SHA-3 Proposal: ECHO. Submission to NIST (updated) (2009), `http://crypto.rd.francetelecom.com/echo/doc/echo_description_1-5.pdf`
6. Bernstein, D.J., (editors), T.L.: eBACS: ECRYPT Benchmarking of Cryptographic Systems. `http://bench.cr.yp.to`, accessed 19 November 2015
7. Bernstein, D.J., Hopwood, D., Hülsing, A., Lange, T., Niederhagen, R., Papachristodoulou, L., Schneider, M., Schwabe, P., Wilcox-O'Hearn, Z.: SPHINCS: practical stateless hash-based signatures. In: EUROCRYPT 2015. pp. 368–397
8. Bertoni, G., Daemen, J., Peeters, M., Assche, G.V.: On the indifferentiability of the sponge construction. In: Advances in Cryptology - EUROCRYPT 2008. pp. 181–197 (2008)
9. Biryukov, A., Khovratovich, D.: PAEQ. Submission to the CAESAR competition (2014), `http://competitions.cr.yp.to/round1/paeqv1.pdf`
10. Buchmann, J.A., Dahmen, E., Hülsing, A.: XMSS - A Practical Forward Secure Signature Scheme Based on Minimal Security Assumptions. In: Post-Quantum Cryptography - PQCrypto 2011. pp. 117–129 (2011)
11. Choy, J., Yap, H., Khoo, K., Guo, J., Peyrin, T., Poschmann, A., Tan, C.H.: SPN-Hash: Improving the Provable Resistance against Differential Collision Attacks. In: AFRICACRYPT 2012. pp. 270–286
12. Contini, S., Lenstra, A.K., Steinfeld, R.: VSH, an Efficient and Provable Collision-Resistant Hash Function. In: EUROCRYPT 2006. pp. 165–182 (2006)
13. De Canniere, C., Sato, H., Watanabe, D.: Hash Function Luffa: Supporting Document. Submission to NIST (Round 1) (2008), `http://ehash.iaik.tugraz.at/uploads/f/fe/Luffa_SupportingDocument.pdf`
14. Dinur, I., Shamir, A.: An Improved Algebraic Attack on Hamsi-256. In: Fast Software Encryption, FSE 2011. pp. 88–106
15. Fuhr, T.: Finding Second Preimages of Short Messages for Hamsi-256. In: ASIACRYPT 2010. pp. 20–37 (2010)
16. Gauravaram, P., Knudsen, L.R., Matusiewicz, K., Mendel, F., Rechberger, C., Schläffer, M., Thomsen, S.S.: Grøstl – a SHA-3 candidate. Submission to NIST (Round 3) (2011), `http://www.groestl.info/Groestl.pdf`
17. Grover, L.K.: A Fast Quantum Mechanical Algorithm for Database Search. In: Miller, G.L. (ed.) ACM Symposium on the Theory of Computing. pp. 212–219. ACM (1996)
18. Guo, J., Peyrin, T., Poschmann, A.: The PHOTON Family of Lightweight Hash Functions. In: CRYPTO 2011. pp. 222–239
19. Gurobi Optimization, Inc.: Gurobi Optimizer Reference Manual (2015), `http://www.gurobi.com`
20. Halevi, S., Hall, W.E., Jutla, C.S.: The Hash Function Fugue. Submission to NIST (updated) (2009), `http://domino.research.ibm.com/comm/research_projects.nsf/pages/fugue.index.html/$FILE/fugue_09.pdf`
21. IBM: ILOG CPLEX Optimizer (2015), `http://www-01.ibm.com/software/commerce/optimization/cplex-optimizer/`
22. Jean, J., Naya-Plasencia, M., Peyrin, T.: Improved Cryptanalysis of AES-like Permutations. J. Cryptology 27(4), 772–798 (2014)

23. Jean, J.: Cryptanalysis of haraka. Cryptology ePrint Archive, Report 2016/396 (2016), http://eprint.iacr.org/
24. Kavun, E.B., Lauridsen, M.M., Leander, G., Rechberger, C., Schwabe, P., Yalçın, T.: Prøst. Submission to the CAESAR competition: https://competitions.cr.yp.to/caesar-submissions.html
25. Kelsey, J., Schneier, B.: Second Preimages on n-Bit Hash Functions for Much Less than 2^n Work. In: EUROCRYPT 2005. pp. 474–490
26. Knudsen, L.R.: Truncated and Higher Order Differentials. In: Fast Software Encryption, FSE 1994. pp. 196–211
27. Kölbl, S., Lauridsen, M.M., Mendel, F., Rechberger, C.: Haraka code repository. https://github.com/kste/haraka
28. Özgül Küçük: The Hash Function Hamsi. Submission to NIST (updated) (2009), http://www.cosic.esat.kuleuven.be/publications/article-1203.pdf
29. Lamport, L.: Constructing digital signatures from a one way function. Technical Report SRI-CSL-98, SRI International Computer Science Laboratory (1979)
30. Mendel, F., Rechberger, C., Schläffer, M., Thomsen, S.S.: The rebound attack: Cryptanalysis of reduced Whirlpool and Grøstl. In: Fast Software Encryption, FSE 2009. pp. 260–276 (2009)
31. Mouha, N., Wang, Q., Gu, D., Preneel, B.: Differential and Linear Cryptanalysis Using Mixed-Integer Linear Programming. In: Inscrypt 2011. pp. 57–76
32. Rechberger, C.: Second-Preimage Analysis of Reduced SHA-1. In: ACISP 2010. pp. 104–116
33. Sasaki, Y., Li, Y., Wang, L., Sakiyama, K., Ohta, K.: Non-full-active Super-Sbox Analysis: Applications to ECHO and Grøstl. In: ASIACRYPT 2010. pp. 38–55 (2010)
34. Sun, S., Hu, L., Wang, P., Qiao, K., Ma, X., Song, L.: Automatic Security Evaluation and (Related-key) Differential Characteristic Search: Application to SIMON, PRESENT, LBlock, DES(L) and Other Bit-Oriented Block Ciphers. In: ASIACRYPT 2014. pp. 158–178
35. Wang, X., Yin, Y.L., Yu, H.: Finding Collisions in the Full SHA-1. In: CRYPTO 2005. pp. 17–36
36. Wu, S., Feng, D., Wu, W., Guo, J., Dong, L., Zou, J.: (Pseudo) Preimage Attack on Round-Reduced Grøstl Hash Function and Others. In: Fast Software Encryption, FSE 2012. pp. 127–145
37. Yu, H., Wang, G., Zhang, G., Wang, X.: The Second-Preimage Attack on MD4. In: CANS 2005. pp. 1–12

A Round Constants

Table 9: Round constants used in π_{512} and π_{256}.

RC_0	`0684704ce620c00ab2c5fef075817b9d`	RC_{20}	`d3bf9238225886eb6cbab958e51071b4`
RC_1	`8b66b4e188f3a06b640f6ba42f08f717`	RC_{21}	`db863ce5aef0c677933dfddd24e1128d`
RC_2	`3402de2d53f28498cf029d609f029114`	RC_{22}	`bb606268ffeba09c83e48de3cb2212b1`
RC_3	`0ed6eae62e7b4f08bbf3bcaffd5b4f79`	RC_{23}	`734bd3dce2e4d19c2db91a4ec72bf77d`
RC_4	`cbcfb0cb4872448b79eecd1cbe397044`	RC_{24}	`43bb47c361301b434b1415c42cb3924e`
RC_5	`7eeacdee6e9032b78d5335ed2b8a057b`	RC_{25}	`dba775a8e707eff603b231dd16eb6899`
RC_6	`67c28f435e2e7cd0e2412761da4fef1b`	RC_{26}	`6df3614b3c7559778e5e23027eca472c`
RC_7	`2924d9b0afcacc07675ffde21fc70b3b`	RC_{27}	`cda75a17d6de7d776d1be5b9b88617f9`
RC_8	`ab4d63f1e6867fe9ecdb8fcab9d465ee`	RC_{28}	`ec6b43f06ba8e9aa9d6c069da946ee5d`
RC_9	`1c30bf84d4b7cd645b2a404fad037e33`	RC_{29}	`cb1e6950f957332ba25311593bf327c1`
RC_{10}	`b2cc0bb9941723bf69028b2e8df69800`	RC_{30}	`2cee0c7500da619ce4ed0353600ed0d9`
RC_{11}	`fa0478a6de6f55724aaa9ec85c9d2d8a`	RC_{31}	`f0b1a5a196e90cab80bbbabc63a4a350`
RC_{12}	`dfb49f2b6b772a120efa4f2e29129fd4`	RC_{32}	`ae3db1025e962988ab0dde30938dca39`
RC_{13}	`1ea10344f449a23632d611aebb6a12ee`	RC_{33}	`17bb8f38d554a40b8814f3a82e75b442`
RC_{14}	`af0449884b0500845f9600c99ca8eca6`	RC_{34}	`34bb8a5b5f427fd7aeb6b779360a16f6`
RC_{15}	`21025ed89d199c4f78a2c7e327e593ec`	RC_{35}	`26f65241cbe5543843ce5918ffbaafde`
RC_{16}	`bf3aaaf8a759c9b7b9282ecd82d40173`	RC_{36}	`4ce99a54b9f3026aa2ca9cf7839ec978`
RC_{17}	`6260700d6186b01737f2efd910307d6b`	RC_{37}	`ae51a51a1bdff7be40c06e2822901235`
RC_{18}	`5aca45c22130044381c29153f6fc9ac6`	RC_{38}	`a0c1613cba7ed22bc173bc0f48a659cf`
RC_{19}	`9223973c226b68bb2caf92e836d1943a`	RC_{39}	`756acc03022882884ad6bdfde9c59da1`

B Active S-boxes

Table 10: Lower bound on the number of active S-boxes in a differential trail for the permutation and for the permutation when used in our mode for π_{512} ((a), (b), (c)) and for π_{256} ((d), (e), (f)). The cell color indicates the number of active S-boxes per total number of AES rounds (more transparent means fewer active).

(a) π_{512} DM-permutation.

	m				
T	1	2	3	4	5
1	1	5	9	25	26
2	5	25	45	50	55
3	9	45	66	75	84
4	25	80	90	100	125
5	41	130	114	125	154
6	60	150	138	150	195
7	64	170	162	175	224

(b) π_{512} permutation used in DM-mode.

	m				
T	1	2	3	4	5
1	0	3	7	17	25
2	3	17	37	46	53
3	7	37	58	71	82
4	17	72	82	96	123
5	33	128	106	121	152
6	52	142	130	146	193
7	60	162	154	171	222

(c) π_{512} permutation used in DM-mode leading to collision.

	m				
T	1	2	3	4	5
1	0	9	13	17	25
2	12	34	37	46	58
3	18	76	60	71	91
4	32	93	84	96	128
5	39	134	108	121	161
6	52	159	132	146	198
7	60	198	156	171	231

(d) π_{256} permutation.

	m				
T	1	2	3	4	5
1	1	5	9	25	26
2	5	25	40	50	55
3	9	35	59	75	84
4	25	60	80	100	125
5	34	80	101	125	153
6	45	100	122	150	190
7	52	110	143	175	221

(e) π_{256} permutation used in DM-mode.

	m				
T	1	2	3	4	5
1	1	5	9	25	26
2	5	25	40	50	55
3	9	35	59	75	84
4	25	60	80	100	125
5	34	80	101	125	153
6	45	100	122	150	190
7	52	110	143	175	221

(f) π_{256} permutation used in DM-mode leading to collision.

	m				
T	1	2	3	4	5
1	13	30	21	25	34
2	20	50	42	50	65
3	38	65	63	75	99
4	35	75	84	100	130
5	56	105	105	125	164
6	55	125	126	150	195
7	73	140	147	175	229

C Considerations Regarding Modes of Operation and Linear Mixing

When designing the general constructions for the compression functions, we initially had three approaches in mind:

1. Davies-Meyer construction with a block cipher (referred to as dm),
2. Davies-Meyer construction with a permutation (referred to as dmperm), and
3. Sponge construction (referred to as sponge).

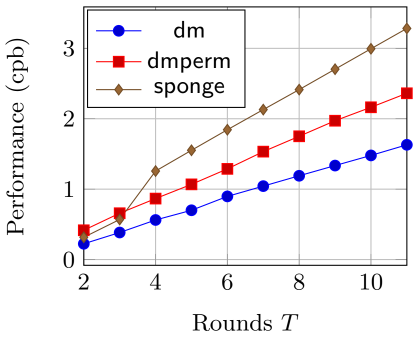
For the first construction, we used a state of two blocks initialized to zero. As part of the round function R_t, we would apply two parallel calls the AES as part of the **aes** operation. The actual bits of the message would be taken into the state over several rounds via a simple message expansion procedure. While the block cipher approach led to a small context size, the simplicity of the message expansion implied the possibility for the attacker to control differences injected even after many rounds, thus obtaining collisions by difference cancellation. While this can potentially be mitigated by a more complex message expansion, this would in turn lead to harder analysis and slower implementations.

In order to avoid the negative consequences on security from a too simple message expansion, and to performance from a too complex message expansion, we opted to abandon the block cipher-based approach of (1) in favor of a permutation-based approach. In particular, we load the full message into the state of the permutation from the beginning. As such, the state size for Haraka-512/256 must be at least 64 bytes, while that of Haraka-256/256 must be at least 32 bytes, or, equivalently $b = 4$ and $b = 2$ blocks, respectively. With this, we considered two general approaches, namely (2) and (3) above. Firstly, one approach is to use a Davies-Meyer construction where the message is loaded into the state which has the size of the domain in bits. This is the approach we landed on, and that described in Section 2 above. Finally, with a Sponge-based approach, one would choose the state size to be *larger* than the size of the domain. The state is initialized to some constant, e.g. all zeroes. The message is XORed into the most significant $|M|$ bits of the state, and a permutation is applied. The output is now taken as e.g. the most significant 256 bits in the case of both Haraka-512/256 and Haraka-256/256.

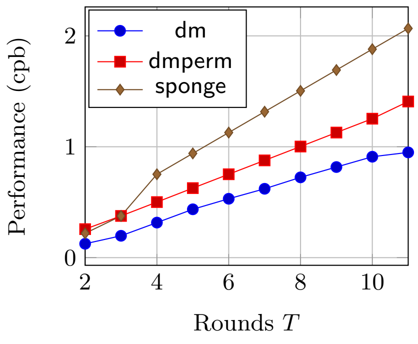
While the dm approach above was found to lead to significantly poorer security margins, in comparison to the dmperm and sponge approaches, we nevertheless implemented all three approaches in C.

For the sponge approach, we used a state consisting of 6 blocks, or, equivalently, 96 bytes. For dm, we used a state of 2 blocks, initialized to zero. The message expansion consisted of shuffling message bits and XORing them to other message bits, so, in other words, a simple linear expansion. In all cases, the permutation applied in each round had the form of **aes** (consisting of m rounds of the AES applied in parallel to each block of the state) followed by a linear mixing. Here, we focus on a fixed mixing layer (in particular using the **blend** mixing detailed below) while, in Section 5.3, we describe considerations regarding different approaches to the linear mixing.

In our consideration here, the mixing layer is implemented by using the **blend** (or pblendw) instruction which is available in Intel CPUs supporting SSE 4.1. The **blend** instruction itself takes in two block operands and an 8-bit mask w. Let $y = \mathbf{blend}_w(a, b)$ be the blend operation on operands a and b using mask w. Then the ith least significant 16-bit word of y is determined as the corresponding word of either a or b, depending on the value of the ith bit of w. As such, **blend** gives us essentially a way to mix two blocks without permuting the byte positions. The mixing using **blend** is now defined as using $\mathbf{blend}_w$ on block i with block $i + 1$ modulo the number of blocks of the state. Fixing $m = 2$, i.e. using two AES rounds per round, Figure 4 details the performance using the three general construction approaches dm, dmperm and sponge, described above. The numbers are taken as the minimum over choices of P in the range $P = 1, \ldots, 16$. Note, that the optimal choice for a particular value of P may not be constant across choices of the number of rounds T. Evidently, the dm approach has the best overall performance. The sponge approach is significantly slower than the dmperm approach when $T > 3$. To that end, and combined with the observation regarding the security properties of the dm approach, this led to the overall choice of the dmperm construction used for both Haraka-512/256 and Haraka-256/256.



(a) Haswell



(b) Skylake

Fig. 4: Performance using $m = 2$ for each of the three general Haraka-512/256 constructions considered

For the linear mixing layer, we considered several possible approaches:

1. The $\mathbf{mix}_{512}$ and $\mathbf{mix}_{256}$ approaches described in Section 2, using the punpckhdq and punpckldq instructions;
2. The **blend** approach, as described above, using the pblendw instruction; and
3. Using a combination of a block-wise byte shuffle and XOR (denoted **shuffle-xor**) with the following state block, i.e. where block i updated with a byte shuffle and XORed with block $i + 1$ modulo the number of blocks, to obtain the updated block. This approach uses the pshufb and pxor instructions.

The effect of each of this operations applied to the state of π_{512} can be seen in Figure 5. On both the Haswell and Skylake microarchitectures, the instructions used for those three approaches all have a latency of one clock cycle, while the

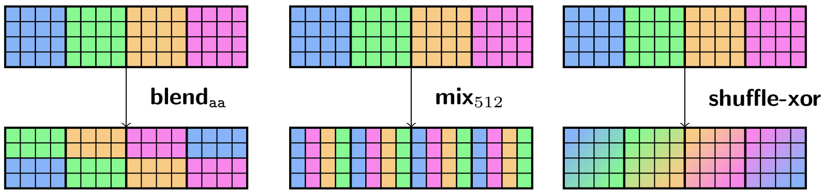


Fig. 5: Effect of applying one round of the mixing layers on the state of π_{512}.

inverse throughput varies from e.g. 0.33 instructions/cycle for the XOR operation to 1 instruction/cycle for the `punpckhdq` and `punpckldq` instructions.

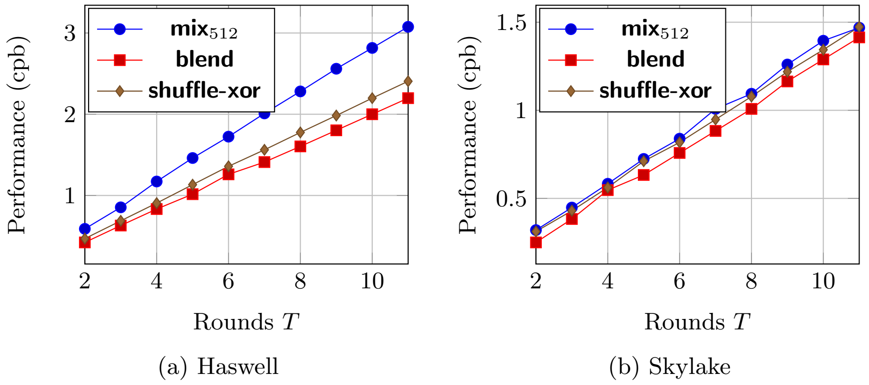


Fig. 6: Performance of Haraka-512/256 using $m = 2$ for each of the three approaches to linear mixing considered

Figure 6 gives a performance comparison of the three approaches to the linear mixing layer. As shown, with the exception of the **mix**$_{512}$ operation on Haswell, all other approaches have comparable performance for both Haswell and Skylake. Concludingly, it makes sense to choose the approach yielding the best security properties, namely the **mix**$_{512}$ and **mix**$_{256}$ operations.