Haraka v2 – Efficient Short-Input Hashing for Post-Quantum Applications

Stefan Kölbl[1], Martin M. Lauridsen[2], Florian Mendel[3], and Christian Rechberger[1,3]

[1] DTU Compute, Technical University of Denmark, Denmark
[2] InfoSec Global Ltd., Switzerland
[3] IAIK, Graz University of Technology, Austria
stek@dtu.dk
martin.lauridsen@infosecglobal.com
{christian.rechberger,florian.mendel}@iaik.tugraz.at

Abstract. Recently, many efficient cryptographic hash function design strategies have been explored, not least because of the SHA-3 competition. These designs are, almost exclusively, geared towards high performance on long inputs. However, various applications exist where the *performance on short (fixed length) inputs matters more*. Such hash functions are the bottleneck in hash-based signature schemes like SPHINCS or XMSS, which is currently under standardization. Secure functions specifically designed for such applications are scarce. We attend to this gap by proposing two short-input hash functions (or rather simply compression functions). By utilizing AES instructions on modern CPUs, our proposals are the fastest on such platforms, reaching throughputs *below one cycle per hashed byte* even for short inputs, while still having a *very low latency* of less than 60 cycles. Under the hood, this results comes with several innovations. First, we study whether the number of rounds for our hash functions can be reduced, if only second-preimage resistance (and not collision resistance) is required. The conclusion 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, including the powerful rebound attacks. We develop 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 commonly 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 [CLS06]. This is in contrast to a sizable and growing set of applications, that utilize cryptographic hashing, but explicitly *do not require collision resistance*. Consider as an example the proof for the HMAC construction, which initially required collision resistance from its hash function [BCK96], but in later versions the collision resistance requirement was dropped in favor of milder requirements [Bel15]. Universal one-way hash functions (UOWHF) [BR97] are, in principle, candidate functions, but they will not suffice for many applications.

Another example, which brings us to the main use-case of this paper, are hash-based signature schemes originally introduced by Lamport [Lam79]. Recent schemes include XMSS [BDH11], which is currently submitted as a draft standard to the IETF and which features short signatures sizes, and the state-less scheme SPHINCS [BHH$^+$15]. A recent version of the former, XMSS-T [HRS16], attains additional security against multi-target preimage attacks on the underlying hash function. Arguably, such designs are the most mature candidates for signature schemes offering post-quantum security, i.e. they are believed to be secure in the presence of hypothetical quantum computers, as their security reduces *solely* to the security properties of the hash function(s) used, thus relying on minimal assumptions.

The hash-based signature schemes mentioned require many calls to a hash function, but only process short inputs. For instance in SPHINCS-256, about 500000 calls to two hash functions are needed to reach a post-quantum security level of 128 bits. One of those functions (denoted H) compresses a 512-bit string to a 256-bit string and is used in a Merkle-tree construction, while the other (denoted F) maps a 256-bit string to a 256-bit string.

The applications share the absence of collision resistance from the requirements imposed on the underlying hash function(s), and further they process only short inputs[4]. However, nearly all cryptographic hash functions are geared towards high performance on long messages and, as we will show, perform rather poorly on short inputs.

1.1 Contributions

Motivated by the applications described above, we explicitly consider preimage- and second-preimage resistance as the sole security goals for cryptographic hash functions, particularly dropping collision resistance, and furthermore target high performance on short (fixed length) inputs. We limit ourselves to one particular design strategy, which is fairly well understood and scalable: AES-like designs. This enables both strong security arguments, while also allowing excellent performance on widespread platforms offering AES-specific instructions, such as modern Intel and AMD CPUs, as well as the ARMv8 architecture.

Concretely we propose Haraka v2, two secure (in the above sense) short-input hash functions achieving a performance better than 1 cycle per byte (cpb) and a latency of only 60 cycles, on various Intel architectures. As we show in Section 5.3, competitive designs are somewhat slower than that. Our proposals share strong similarities with the permutation AESQ that is used in the CAESAR candidate PAEQ [BK14]. We perform benchmarks of the SPHINCS-256 hash-based signature, using Haraka v2 as the underlying hash functions, and show performance speed-ups of $\times 1.50$ to $\times 2.86$. As hash-based signature schemes such as SPHINCS are already practical, and in the case of XMSS in the process of standardization, this shows that Haraka v2 can significantly contribute to speeding up such schemes.

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

On the theoretical side, our proposal also carries with it several contributions. Firstly, we study if the number of rounds for Haraka v2 can be reduced if only second-preimage resistance, and *not collision resistance*, is required. The conclusion is that only one round (5 rounds instead of 6) can be dropped. Secondly, and as a point we like to elaborate at this point already, we describe new ways to bound the applicability of attacks. Traditionally, resistance against differential attacks (which are important for collision- and second-preimage attacks) of key-less constructions, such as cryptographic hash functions, is almost solely based on arguments that are also found for keyed constructions such as block ciphers. Common approaches include (1) using a bound on the best differential trail and comparing it with the available degrees of freedom, or (2) assuming a number of rounds *controlled* by the attacker, and use a bound on the best differential trail for the *uncontrolled* rounds as a security margin. Such arguments have been used for various SHA-3 candidates like Grøstl [GKM+11], ECHO [BBG+09], Luffa [DSW08], and the more recent hash function PHOTON [GPP11]. One problem of these approaches is that they do not consider truncated differentials, and as such do not cover rebound attacks.

Arguments against rebound attacks are of course still possible and can be found in the literature, also for the aforementioned designs. Often this involves designing concrete rebound attacks along with arguments for why improving them is unlikely. Alternatively, designers make assumptions akin to (2) about the *controlled* rounds that are simply "for free", and then focus on bounding the effect of the *uncontrolled* rounds. Perhaps the most notable arguments in that direction are for the design of SPN-Hash [CYK+12], which uses approach (2), but provides bounds for the *uncontrolled* rounds using differentials and not solely single trails. However, the *controlled* rounds are still treated as a black box.

To improve the situation, we propose a way to model an idealized attacker who has capabilities which resemble cryptanalytic techniques such as the rebound attack. We take into account how the complexity of an attack can be reduced in the controlled rounds, like in the inbound phase of the rebound attack, by using the available degrees of freedom to fulfill conditions in a truncated differential. This allows us better security arguments by not having to treat parts of the hash function as a black box, and we can take into account also the fact that there are less degrees of freedom available in a second-preimage attack. Overall, this gives us a better understanding of the required number of rounds for Haraka v2 to resist these types of attacks.

Finally, we remark at this point that both implementations of our proposals, including test vectors, the SPHINCS code for benchmarking and the code used to generate the MILP models for the security analysis of Haraka v2, are publicly available [KLMR16].

1.2 Related Work

Several proposals have been submitted to the SHA-3 competition that aim to take advantage of AES instructions in modern CPUs. Among them are Grøstl [GKM+11], ECHO [BBG+09], Fugue [HHJ09], LANE [Ind08]. Many of them are geared towards performance on long messages, and often show severe performance degradation for short messages. The CAESAR competition for authenticated encryption schemes saw many proposals, including AEGIS [WP14], PAEQ [BK14] and Tiaoxin [Nik14], based on utilizing AES instructions. Recently, two designs for permutations based on the AES round function have been proposed. Jean and Nikolic [JN16] studied AES-based designs for MACs and authenticated encryption, however not for hashing applications. Gueron and Mouha [GM16] propose Simpira v2, a family of permutations based on Feistel networks.

1.3 Recent Developments in Short-Input Hashing

Haraka v1 was originally presented to a larger group of cryptographers in November 2015 [Pro], with the explicit goal of providing fast hashing on short inputs, the main application being speeding up hash-based signature schemes. Simpira, mentioned above, started circulating a few months thereafter, with one of the three main applications mentioned also being hash-based signature schemes [GM16, Section 7]. Haraka v1 was broken by Jean [Jea16] (see also Section 4.2), and in this paper Haraka v2 is presented, which differs from Haraka v1 in the choice of round constants. Simpira (the former version) was broken in two different ways [DEM16, Røn16] and Simpira v2 addresses the identified problems. Concrete performance numbers for Simpira v2 in modern hash-based signature schemes are not available yet, but our benchmarks in Section 5.3 suggest that Simpira v2 is slower.

Recently, KangarooTwelve, a variant of Keccak with a reduced number of rounds, was proposed, aimed at improved hashing speed. However, its performance is still geared towards long inputs. Furthermore, improvements on SHA-256 implementations are being discussed in the community. In Section 5.3 we discuss briefly recent performance figures on Skylake for SHA-256 as well as comparison with KangarooTwelve.

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 [AB12] family, but the security requirements are much lower for this setting.

2 Specification of Haraka v2

Haraka v2 exists in two variants denoted Haraka-512 v2 and Haraka-256 v2 with signatures

$$\begin{aligned} &\text{Haraka-512 v2} : \mathbb{F}_2^{512} \to \mathbb{F}_2^{256} \quad \text{and} \\ &\text{Haraka-256 v2} : \mathbb{F}_2^{256} \to \mathbb{F}_2^{256}. \end{aligned} \tag{1}$$

For both variants, we claim 256 bits of security (respectively 128 bits in the presence of quantum computers) against (second)-preimage resistance, but we make *no further 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 v2 and Haraka-256 v2 employ the well-known Davies-Meyer (DM) construction using a permutation with a feed-forward (applying the XOR operation) of the input. As such, they are defined as

$$\begin{aligned} &\text{Haraka-512 v2}(x) = \mathsf{trunc}(\pi_{512}(x) \oplus x) \quad \text{and} \\ &\text{Haraka-256 v2}(x) = \pi_{256}(x) \oplus x, \end{aligned} \tag{2}$$

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 v2. In Section 3, we give our security analysis of the constructions and, based on this, motivate our design choices in Section 4.4.

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. When we talk about a *block*, we refer to a 16-byte string consisting of four columns denoted $x_{4i} \| \cdots \| x_{4i+3}$ for $i = 0, \ldots, b-1$. In general, we let b denote the number of 128-bit blocks of the state, so for π_{512} we have $b = 4$ while for π_{256} we have $b = 2$. The state arrangement is given in Figure 1.

Haraka-256 v2 state

$x_{3,0}$	$x_{3,1}$	$x_{3,2}$	$x_{3,3}$	$x_{3,4}$	$x_{3,5}$	$x_{3,6}$	$x_{3,7}$	$x_{3,8}$	$x_{3,9}$	$x_{3,10}$	$x_{3,11}$	$x_{3,12}$	$x_{3,13}$	$x_{3,14}$	$x_{3,15}$
$x_{2,0}$	$x_{2,1}$	$x_{2,2}$	$x_{2,3}$	$x_{2,4}$	$x_{2,5}$	$x_{2,6}$	$x_{2,7}$	$x_{2,8}$	$x_{2,9}$	$x_{2,10}$	$x_{2,11}$	$x_{2,12}$	$x_{2,13}$	$x_{2,14}$	$x_{2,15}$
$x_{1,0}$	$x_{1,1}$	$x_{1,2}$	$x_{1,3}$	$x_{1,4}$	$x_{1,5}$	$x_{1,6}$	$x_{1,7}$	$x_{1,8}$	$x_{1,9}$	$x_{1,10}$	$x_{1,11}$	$x_{1,12}$	$x_{1,13}$	$x_{1,14}$	$x_{1,15}$
$x_{0,0}$	$x_{0,1}$	$x_{0,2}$	$x_{0,3}$	$x_{0,4}$	$x_{0,5}$	$x_{0,6}$	$x_{0,7}$	$x_{0,8}$	$x_{0,9}$	$x_{0,10}$	$x_{0,11}$	$x_{0,12}$	$x_{0,13}$	$x_{0,14}$	$x_{0,15}$

Haraka-512 v2 state

Fig. 1: State for Haraka-512 v2 and Haraka-256 v2 (not including the shaded area). The box $x_{i,j}$ denotes the ith byte in the jth column of the state.

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 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 $R_t = \textbf{mix}_{512} \circ \textbf{aes}$ while for π_{256} it is $R_t = \textbf{mix}_{256} \circ \textbf{aes}$. 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 number of rounds is $T = 5$ while using $m = 2$ AES rounds for both Haraka-512 v2 and Haraka-256 v2 (totaling 10 AES rounds).

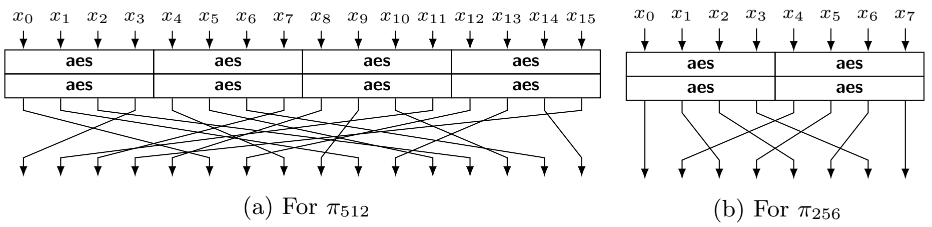
The main difference between π_{512} and π_{256} is the linear mixing used. In both cases, the mixing itself is comprised of simply permuting the state columns. For π_{512}, the sixteen columns of the state are permuted such that each output block contains precisely one column from each of the $b = 4$ 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. Letting $x_0 \| \cdots \| x_{15}$ denote the columns for a state of π_{512}, the columns are permuted by 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 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 2.



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
aes
aes
aes
aes
aes
aes
(a) For π_{512}
x_0 x_1 x_2 x_3 x_4 x_5 x_6 x_7
aes
aes
aes
aes
(b) For π_{256}

Fig. 2: 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 via the round key addition in order to destroy the symmetries between the individual states (see Section 4.2). The constants are derived using a similar approach as in the CAESAR candidate Prøst [KLL+14]. 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

$$RC_j = p_{128j+128}||\ldots||p_{128j+2}||p_{128j+1} \qquad \forall j = 0\ldots 39. \tag{5}$$

The AES layer aes_i uses round constants $(RC_{4i}, RC_{4i+1}, RC_{4i+2}, RC_{4i+3})$ in the case of π_{512}, respectively (RC_{2i}, RC_{2i+1}) for π_{256}. The constants are also given in Appendix A. The use of π is an application of a nothing-up-my-sleeve number; another choice of a known constant would be an equally qualified.

Truncation Function Let $x \in \mathbb{F}_2^{512}$. Then $\mathsf{trunc}(x)$, which is used in Haraka v2, is obtained as concatenating two columns from each block: The least significant two from the first two blocks, and 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 Requirements

The three most commonly defined security requirements for a cryptographic hash function H 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.

Generic attacks, which can find a (second-)preimage with a complexity of 2^n and collisions with a complexity of $2^{n/2}$, exist for any hash function, where n is the digest size in bits. Quantum computers can improve upon this by using Grover's algorithm [Gro96] 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. Brassard, Høyer and Tapp's method [BHT98] suggests an algorithm finding collisions in $2^{n/3}$ steps, however the actual costs are not lower compared to methods based on classical computers [Ber09].

In the following sections, we discuss common attack vectors which will aid in choosing appropriate parameters for Haraka v2 to achieve the desired security properties. As described, we focus on second-preimage resistance, as the main applications of Haraka v2 do not require collision resistance.

3.1 Preliminaries

Differential cryptanalysis is a powerful tools for 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 S^t, 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 (i.e. the differences match). The last equality holds if we assume independent rounds.

The AES round function uses the SubBytes, ShiftRows and MixColumns operations (denoted SB, SR and MC for short). For our further analysis we will be interested in how truncated differentials [Knu94] propagate through MixColumns. The branch number of MixColumns is 5, so if an input column to MixColumns contains a active bytes, then the probability of having b active bytes in the corresponding output column, where $a + b \geq 5$ and $1 \leq a, b \leq 4$, can be approximated by $2^{(b-4)8}$.

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. 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) [MWGP11, SHW$^+$14]. The basic idea is to express the restrictions on the trail, given by the round transformations, as linear equations, and generate a optimization problem which can be solved with any MILP optimizer, e.g. Gurobi [Inc16] or CPLEX [IBM16]. We use this technique later to find the minimum number of active S-boxes for Haraka v2, which aids us in an informed choice of parameters.

3.2 Capabilities of an Attacker

One of the main difficulties in the design of hash functions is to estimate the security margin one expects against a powerful attacker. As described, bounding the probability of trails can be useful for block ciphers but are of limited use for hash functions, as there is no secret input. Degrees of freedom can be used, to some extent, to solve many conditions on the trail and lead to surprisingly efficient attacks. This was partially addressed in the design of Fugue [HHJ09] and SPN-hash [CYK$^+$12]. 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, estimated based on existing attacks, and the total number of rounds is 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 major drawback of this approach is that they do not resemble the capabilities of an attacker in practice, which can lead to too conservative estimates while also ignoring important attack vectors.

The most powerful collision attacks on AES-based hash functions, such as the rebound attack [MRST09], use truncated differentials combined with a clever use of the degrees of freedom to reduce the attack complexity. Arguing security against this 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. 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 trails. In the following, we propose a new method to better bound the capabilities of an attacker in practice under reasonable assumptions.

Truncated Differentials A MILP model to count the number of active S-boxes inherently uses truncated differentials (at the byte level), as it considers active bytes but not the difference values, but it does not cover the costs of their propagation. 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.

An attacker can always use a (fully active) truncated differential with probability ≈ 1 (as a fully active state will very likely remain fully active after MixColumns), which gives a valid second-preimage if the input difference is equal to the output difference. This happens with a probability of 2^{-256}, hence the security can be at most 256 bits for this attack vector.

Utilizing Degrees of Freedom The previous approaches still ignore 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 having a probability of 1 and then use the degrees of freedom to guarantee the condition $f(x) \oplus f(x \oplus \alpha) = 0$. In general the state size is at least as big as the output size, hence the attacker will have enough degrees of freedom to solve these conditions.

However, it is very unlikely that an attacker can utilize the degrees of freedom in this way without further restrictions. In the case of AES, already after two rounds we get full diffusion, i.e. every byte of the output depends on all bytes of the input. In general solving a condition like $f(x) \oplus f(x \oplus \alpha) = 0$ then corresponds to solving a system of non-linear equations over $\mathbb{F}_{2^8}$ which is an NP-hard problem.

The model we propose is more restrictive and reflects the capabilities of an attacker in practice. The attacker is still allowed to solve conditions for *free* using the degrees of freedom, but can only do so for q consecutive rounds of the primitive. This means, the attacker chooses a state S^k and then is allowed to solve any conditions for states $S^{k-q}, \dots, S^{k+q}$ in *constant time*, as long as there are still degrees of freedom available. The remaining conditions which can not be solved make up the security level. We can formulate this as a MILP problem with the goal of finding the lowest attack complexity over all possible states S^k (for more details and the application to Haraka v2 see Section 4.2).

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 [MRST09, WYY05]. 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 [SLW$^+$10, JNP14]. These results can not be carried over directly to our construction, as we compose our state of four individual (respectively two) AES states. Very recent work on AESQ [BMS16] found that 4 AES rounds can be covered in an inbound phase, albeit at a high cost.

Therefore, we use both $q = 2$ for the collision and second-preimage case, allowing our idealized attacker to cover 4 rounds with the degrees of freedom to have a comfortable security margin.

4 Analysis of Haraka v2

In the following we give the security claims for Haraka v2 and the security analysis which lead to the proposed parameters.

4.1 Security Claims

We claim second-preimage resistance of 256 bits for Haraka v2 against classical computers. As will been seen later in the paper, for only one additional round (a performance penalty of around 20%) we claim 128 bits of collision resistance. We make no claims against near-collisions or other generalizations of this property, nor against distinguishers of the underlying permutation, because such properties do not seem to be needed in applications like hash-based signature schemes [BDH11, BHH+15]. 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 [DS11, Fuh10] on the SHA-3 candidate Hamsi [Kü09] which uses a very strong non-random property of the compression function. No such strong property seems likely to exist for our proposals.

4.2 Second-Preimage Resistance

For an output size of $n = 256$ the best generic attacks have a complexity of 2^{256} respectively 2^{128} on a classical- respectively quantum computer. For iterative hash functions, a generic attack exists which improves upon the naïve brute force approach for finding second preimages [KS05]. However, this attack requires long messages and is therefore not applicable to our construction.

Differential Second-Preimage Attack for Weak Messages For finding a second-preimage the attacker can use a differential trail Q leading to a collision, that means $f(x \oplus \alpha) = y$. However, as the values of the state are fixed by the output y, all differential trails hold 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 yield a second-preimage for y, then the attacker must try another trail $Q' \neq Q$ or another message.

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 itself, as well as when the permutation is used in the DM mode. As some of the output is truncated for Haraka-512 v2, this can potentially reduce the number of active S-boxes and has to be taken into account. For Haraka-512 v2 the best differential trail has a probability of $\mathrm{DP}(Q) = 2^{-780}$, while the best trail leading to a collision has probability $\mathrm{DP}(Q) = 2^{-804}$ when used in DM mode. Similarly, for Haraka-256 v2, those

Table 1: Lower bound on the number of active S-boxes in a differential trail for the permutations used in Haraka v2, for the permutation when used in DM mode and for trails leading to a collision when used in DM mode. Appendix C gives the numbers for a wider choice of parameters.

	Permutation	DM-mode	DM-mode (coll.)
Haraka-256 v2	80	80	105
Haraka-512 v2	130	128	134

probabilities are 2^{-480} and 2^{-630}, respectively. For the number of active S-boxes for Haraka-512 v2 and Haraka-256 v2 see Table 1. Note that this corresponds to previous work that studied second-preimage attacks for MD4 [YWZW05] and SHA-1 [Rec10].

Truncated Differentials We can use the approach from Section 3.2 to bound the costs of finding a second-preimage for an idealized attacker in order to determine the number of rounds for Haraka v2. To find a second-preimage the attacker needs to first find a truncated differential leading to a collision and then determine the trail with the available degrees of freedom. However, as the state is fixed by the initial message the degrees of freedom are limited to the choice of differences for each active byte in the truncated trail.

We denote the input column j to $\mathsf{MixColumns}$ (resp. $\mathsf{SubBytes}$) in round t as MC_j^t (resp. SB_j^t) and consider the number of rounds T and the number of AES steps per round m, as variables. We define the cost for an attacker to fulfill the conditions of a truncated differential, starting at state S^k, as

$$C_{\text{trunc}} = \sum_{t=0}^{T\cdot m-1} \sum_{j=0}^{4b-1} C_{\mathsf{MC}_j}^t \tag{9}$$

where the costs in the forward direction are given by decision variables $C_{\mathsf{MC}_j}^t$ satisfying

$$\forall t : k \le t \le T\cdot m, \forall j : 0 \le j < 4b : \qquad C_{\mathsf{MC}_j}^t \ge \left(4 - \sum_{i=0}^{3} \mathsf{SB}_{i,j}^t\right)\cdot 8 \tag{10}$$

and in the backwards direction by

$$\forall t : 0 \le t < k, \forall j : 0 \le j < 4b : \qquad C_{\mathsf{MC}_j}^t \ge \left(4 - \sum_{i=0}^{3} \mathsf{MC}_{i,j}^t\right)\cdot 8 \tag{11}$$

where $\mathsf{SB}_{i,j}^t$ resp. $\mathsf{MC}_{i,j}^t$ is 1 if the byte is active and 0 otherwise. Note that here $C_{\mathsf{MC}_j}^t$ corresponds to the $\log_2$ complexity for the transitions through $\mathsf{MixColumns}$ (resp. the inverse $\mathsf{MixColumns}$) to satisfy the truncated differential trail.

An additional requirement is, that the input and output difference are equal, in order to get a valid second-preimage, that means $\mathsf{trunc}(x \oplus \alpha) = \mathsf{trunc}(\Delta\pi_{512}(x \oplus \alpha))$. The complexity depends on the number of active bytes at the input which are not truncated

$$C_{\text{collision}} = \sum_{j\in\mathcal{I}_c} \sum_{i=0}^{3} \mathsf{SB}_{i,j}^0 \cdot 8 \tag{12}$$

where $\mathcal{I}_c$ is the set of column indices which are not truncated at the output. The optimization goal for the MILP problem is then given by

$$\textbf{minimize:} \quad C_{\text{collision}} + C_{\text{trunc}}. \tag{13}$$

The requirements for Haraka v2 are that each attack in this model costs at least 2^{256} to have a good security margin. We applied this model to explore how the security level evolves for different choices of 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 are given in Table 2. The time to solve the MILP problem 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.

Degrees of Freedom. The previous scenario does not yet take into account the capabilities of an attacker utilizing the available degrees of freedom. For the second-preimage scenario the attacker can freely choose the differences in one of the states S^k to reduce the costs of the attack for q rounds in both directions

$$\mathcal{D} = \sum_{j=0}^{4b-1} \sum_{i=0}^{3} S^k \cdot 8. \tag{14}$$

The costs for an attack are then given by the number of conditions which can be reduced in the controlled rounds $\mathcal{R} = \{r \mid k - q \leq r < k + q \wedge 0 \leq r < T \cdot m\}$ by using degrees of freedom

$$C_{\text{reducible}} \geq \sum_{t \in \mathcal{R}} \sum_{j=0}^{4b-1} C^t_{\mathsf{MC}_j} - \mathcal{D} \quad \text{and} \quad C_{\text{reducible}} \geq 0, \tag{15}$$

and the number of conditions which can not be controlled by the attacker

$$C_{\text{trunc}} = \sum_{t \in \mathbb{Z}_{T \cdot m} \setminus \mathcal{R}} \sum_{j=0}^{4b-1} C^t_{\mathsf{MC}_j}. \tag{16}$$

The goal is now to find the minimal attack costs by solving this MILP model

$$\textbf{minimize:} \quad C_{\text{collision}} + C_{\text{trunc}} + C_{\text{reducible}}. \tag{17}$$

If we do not allow the attacker to utilize any degrees of freedom, the parameters $T = 4$ and $m = 2$ would be sufficient for Haraka-512 v2, and parameters $T = 2$ and $m = 2$ would suffice for Haraka-256 v2 (see Table 2). However, as discussed in Section 3.2, this approach would be too optimistic (from our perspective). Taking into account the assumptions we make on the capabilities of an attacker utilizing the degrees of freedom, at least $T = 5$ rounds are required (see Table 3) for the best attack to require at least 2^{256} steps.

In Figure 3, we give an example to illustrate how this attack model works for a collision attack. The attacker starts in this case at S^4 and can control $q = 2$ rounds in both directions. When searching for 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 $\mathsf{MixColumns}$ in the controlled rounds. The only remaining part is the transition in the first round which happens with a probability of 2^{-16}.

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

Table 2: Complexity bounds ($\log_2$) of the best attack in our truncated setting, over multiple rounds, without utilizing degrees of freedom

(a) Security for π_{512}

	m				
T	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}

	m				
T	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: Complexity bounds ($\log_2$) of the the best attack in our truncated setting, utilizing additional degrees of freedom over $2q$ rounds for π_{512} and π_{256}, with $m = 2$ fixed. Entries which are bold are not better then the generic attacks.

(a) Second-preimage for 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 for 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 for 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 for 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**

Meet-in-the-middle Attacks A powerful technique for finding preimages are meet-in-the-middle techniques and they have been applied to various AES-based hash functions, for instance Whirlpool [Sas11] and Grøstl [WFW$^+$12]. The basic attack principle is to split the function into two sub-functions, such that a part of the message only affects the first function and another part of the message the second function. These sub-functions are referred to as chunks (of rounds) and bytes which have influence on only one of these functions are called *neutral* bytes. The limiting constraint of this attack is the number of rounds we can independently propagate our message through these chunks.

We are interested in finding out the highest number of rounds of Haraka-256 v2 and Haraka-512 v2 that can be attacked. In this case, the strategy is to have a single neutral byte in the forward and backward chunk. We can check for all possible positions of two

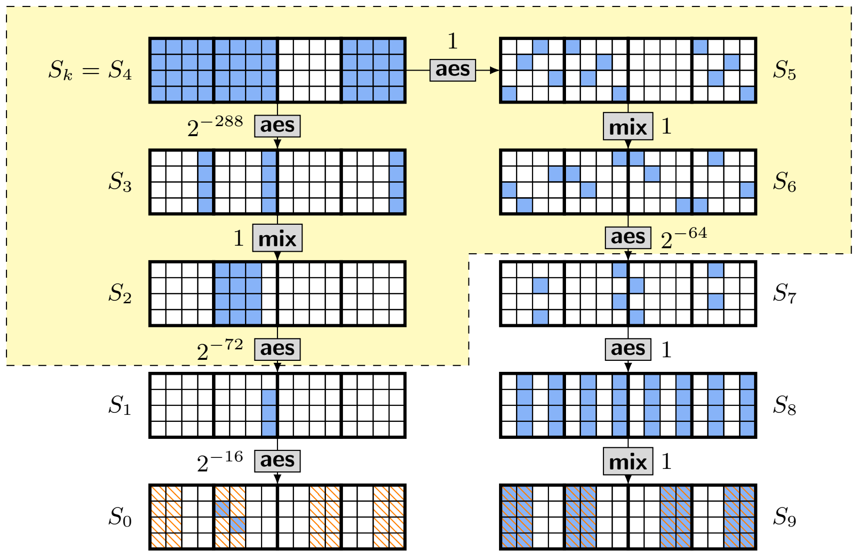


Fig. 3: Truncated model utilizing degrees of freedom for $T = 3, m = 2$ and $q = 2$. An active byte is marked as 🟦; a byte which is removed due to trunc is marked with ▨; boxes $\boxed{f}$ denotes a function f mapping one state to the next, and the number next to it gives the transition probability. For finding a collision, the attacker would have full control over the middle rounds, marked in the highlighted area. As there are only 53 conditions on bytes which all can be fulfilled with the available degrees of freedom the attack costs for an idealized attacker would be 2^{16}.

unknown bytes after how many rounds we still are able to find a match, meaning that the state is not unknown in all bytes. Starting at the beginning of a Haraka v2 round in the forward direction we can still compute the value of 16 bytes after $\mathsf{SR} \circ \textbf{mix} \circ \mathsf{MC} \circ \mathsf{SR} \circ \mathsf{MC} \circ \mathsf{SR}$. In the backwards direction we can also compute 16 bytes after $\mathsf{SR}^{-1} \circ \mathsf{MC}^{-1} \circ \textbf{mix}^{-1} \circ \mathsf{SR}^{-1} \circ \mathsf{MC}^{-1} \circ \mathsf{SR}^{-1} \circ \mathsf{MC}^{-1} \circ \textbf{mix}^{-1}$. In total this covers 3 rounds of Haraka v2. In an attack we can choose a different starting point, but the total number of rounds which can be covered stays the same. The initial structure technique [SA09] allows us to further extend the separation of the two chunks by 1 round.

We can use this now to mount an attack on 3.5 rounds of Haraka-256 v2, following the procedure given in [Sas11] (see Figure 4):

1. Randomly select values for the constant bytes ■ in AC^4.
2. For all 2^8 possible choices for $\mathsf{AC}^4_{*,0}$ 🟦 which keep $\mathsf{MC}^4_{0,0}, \mathsf{MC}^4_{1,0}, \mathsf{MC}^4_{2,0}$ constant, compute forward to obtain the state in MC^0 and store the result in a table T.
3. For all 2^8 possible choices for $\mathsf{MC}^5_{*,4}$ 🟥 which keep $\mathsf{AC}^5_{0,4}, \mathsf{AC}^5_{1,4}, \mathsf{AC}^5_{2,4}$ constant, compute backward to obtain the state in AC^0.
4. Check if there is an entry in T that matches with AC^0 through MixColumns. If so check whether the remaining bytes also match, otherwise repeat from step 2 (or step 1 if necessary).

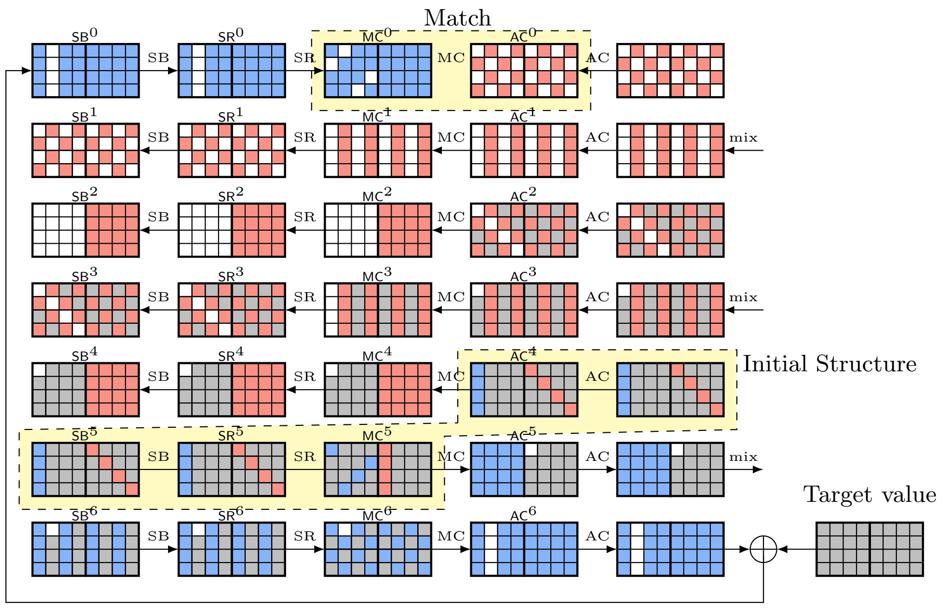


Fig. 4: Meet-in-the-middle attack on 3.5 rounds of Haraka-256 v2. All □ are unknown, ▨ are constant, 🟥 neutral bytes backward and 🟦 neutral bytes forward.

Matching. We can check whether the states MC^0 and AC^0 can be matched through $\mathsf{MixColumns}$ in the following way. Lets consider the first column, which gives us the following two equations

$$\mathsf{AC}^0_{2,0} = \mathsf{MC}^0_{3,0} \oplus 2 \cdot \mathsf{MC}^0_{2,0} \oplus 3 \cdot \mathsf{MC}^0_{1,0} \oplus \mathsf{MC}^0_{0,0} \tag{18}$$

$$\mathsf{AC}^0_{0,0} = 3 \cdot \mathsf{MC}^0_{3,0} \oplus \mathsf{MC}^0_{2,0} \oplus \mathsf{MC}^0_{1,0} \oplus 2 \cdot \mathsf{MC}^0_{0,0} \tag{19}$$

As we know the values for $\mathsf{MC}^0_{3,0}, \mathsf{MC}^0_{1,0}, \mathsf{MC}^0_{0,0}$ we can simplify this to

$$\mathsf{AC}^0_{2,0} \oplus C_0 = 2 \cdot \mathsf{MC}^0_{2,0} \quad \text{and} \quad \mathsf{AC}^0_{0,0} \oplus C_1 = \mathsf{MC}^0_{2,0}. \tag{20}$$

We can use this now to check whether we can fulfill:

$$\mathsf{AC}^0_{2,0} \oplus C_0 = 2 \cdot (\mathsf{AC}^0_{0,0} \oplus C_1) \tag{21}$$

$$\mathsf{AC}^0_{2,0} \oplus 2 \cdot \mathsf{AC}^0_{0,0} = C_0 \oplus 2 \cdot C_1, \tag{22}$$

where the right side can be computed in step (2) and the left side in step (3) of our attack.

Complexity. Computing the table T and 2^8 values for AC^0 costs 2^8 3.5-round Haraka-256 v2 evaluations and requires $2^8 \cdot 8$ bytes of memory, as we only need to store 1 byte of information for each column. The success probability for the match is 2^{-32} for the left half of the state and 2^{-64} for the right half. Hence, on average $2^8 \cdot 2^8 \cdot 2^{-96} = 2^{-80}$ candidates will remain in Step 4. There are still $12 + 8$ byte conditions which have to be satisfied, therefore if we repeat step 1-4 2^{240} times we expect to find $2^{240} \cdot 2^{-80} \cdot 2^{-20 \cdot 8} = 1$ solution. The overall complexity is $2^{240} \cdot 2^{8} = 2^{248}$ evaluations of Haraka v2 to find a preimage.

We were not able to extend the attack to 4 rounds, as we would have only two bytes in each column of MC^0 and AC^0. In this case we can not filter out solutions in the matching step. For Haraka-512 v2 we can attack 4 rounds in a very similar way (see Section D).

Attack on Haraka v1 by Jean An attack by Jean [Jea16] on a previous version of Haraka v2, denoted Haraka v1, has been published. In this section we explain how the attack was possible, and why it is not applicable to Haraka v2 presented in this paper. In [LSWD04] it was shown that if the two halves of an AES state are equal, then applying a keyless AES round function preserves this property. In Haraka v1, round constants exhibited strong symmetries in the sense that i) the same constant was used for each block, and ii) the same constant was used for each column in each block. The observation by Jean is, that when using the property of [LSWD04] together with the weak constants and the fact that the mixing layer of Haraka v2 permute the columns, one can construct an efficient structural distinguisher that allows for collisions and preimages.

In Haraka v2 presented in this paper, this structural property has been dealt with by destroying properties (i) and (ii) above, particularly by using round constants based on the digits of π. We refer to Section 2 for the details. We remark that the new choice of constants *do not affect* the performance of Haraka v2. As the attack was structural, and feasible purely due to the round constants, we believe the number of rounds for Haraka v2, which is based on the truncated model (see Section 4.2), is still well-founded and provides long-term security.

4.3 Collision Resistance

While we explicitly do not require collision resistance for Haraka v2, we still discuss the security level 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 these are covered by our model. However, for finding a collision, an attacker can freely choose the complete internal state and not only the differences. This translates to more degrees of freedom. Therefore, the expected security level is lower for the same number of rounds (see Table 3).

The best generic attack 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 security model. However, this would still indicate some non-ideal property, and it is likely that the more relaxed collision setting allows to exploit this after using up all degrees of freedom. Consequently, we opt to also aim for a security level of 2^{256} in our truncated security model, which requires adding one round for Haraka-512 v2.

4.4 Design Choices

In the following, we interpret the security analysis of Section 4.2 which led to the proposed parameters and design choices. We recall 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.

Mode As described, we use the DM mode for our permutations to define Haraka v2. Other modes were considered, including a sponge-based construction and a block cipher in DM mode. The choice to use a permutation in DM mode is motivated both by performance and security considerations. We refer to Appendix E for the details.

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. Our analysis of Table 2 gives a strong indication that $m = 2$ is an optimal choice, as it gives the best trade-off between number of active S-boxes and the total number of required AES rounds Tmb. The number of rounds is chosen as $T = 5$, as this gives the required security parameters in the truncated model, even when assuming a 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. Our main criteria were that the layer should be efficiently implementable (see Section 5.2) on our target platforms, while still contributing to a highly secure permutation. Other potential candidates for the mixing layer are discussed in Appendix E. With respect to our criteria, for most choices of T and m, using the proposed $\textsf{mix}_{512}$ and $\textsf{mix}_{256}$ give a significantly higher number of active S-boxes, compared to other approaches discussed in Appendix E.

Truncation Pattern for Haraka-512 v2 There are many possible choices for the truncation pattern for Haraka-512 v2. 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. The pattern we chose is taking the two least significant columns of the first two blocks and the two most significant columns of the last two states. We found that this approach compared favorably, with respect to the number of active S-boxes, to row-wise patterns or patterns choosing the same two columns from each state.

5 Implementation Aspects and Performance

As mentioned, Haraka v2 is designed *solely for use on platforms with AES hardware support.* To that end, we assume the existence of a hardware instruction pipeline, which can execute a single round of the AES with an instruction denoted **aes**, with a *latency* of $L_{\textsf{aes}}$ cycles and an *inverse throughput* of $T_{\textsf{aes}}^{-1}$ instructions per cycle (given for our target architectures in Table 4). 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 v2 to be efficiently implementable on ARMv8 due to its support of AES instructions. We remark that the Turbo Boost technology has been switched off for all our performance measurements.

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

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

Architecture	$L_{\textsf{aes}}$ [cycles]	$T_{\textsf{aes}}^{-1}$ [instructions/cycle]
Haswell	7	1
Skylake	4	1

$k = L_{\textsf{aes}} \cdot T_{\textsf{aes}}^{-1}$ 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 5. As such, with Haraka v2 using several AES blocks, the pipeline is better utilized.

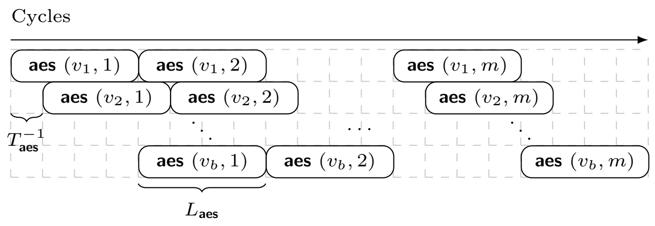


Fig. 5: Pipelined AES instructions. A box **aes** (v, i) denotes the application of the ith AES round to a block v.

5.1 Multiple Inputs

As described above, the theoretically optimal choice of state blocks, performance wise, would be $b = L_{\textsf{aes}} \cdot T_{\textsf{aes}}^{-1}$. However, Haraka v2 uses a varying number of blocks. To that end, we consider for both Haraka-512 v2 and Haraka-256 v2 the parallel application of the corresponding function to *multiple inputs*, assuming that such are available for processing. For example, if $k = L_{\textsf{aes}} \cdot T_{\textsf{aes}}^{-1} = 7$, 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 in this manner 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.

5.2 Implementation of Linear Mixing

Consider the case where $P = 1$, i.e. when using a single input. Even if the number of blocks in the state is less than $L_{\textsf{aes}} \cdot T_{\textsf{aes}}^{-1}$, 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 second AES round of a Haraka v2 round 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 mix_{512} and 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. In the case of Haraka-512 v2, where the state has $b = 4$ blocks, mix_{512} uses eight instructions in the mixing layer, while for Haraka-256 v2 we require just one call to each of the instructions.

5.3 Haraka v2 Performance and Discussion

In the following, we present the performance of Haraka v2 when implemented on the Haswell and Skylake platforms, and discuss their performance in relation to other primitives which would be other potential candidates for our target applications.

Table 5: Benchmarks for various primitives on the Haswell and Skylake platforms. We give the implementation type as well as state size, block size and output size. Implementations marked † are taken from SUPERCOP (see eBACS [Be]); the rest are written by us. For selected primitives, we give the performance using a varying number of independent inputs processed in parallel, $P \in \{1, 4, 8\}$.

			Sizes (bits)				
	Primitive	Implementation	State	Block	Output	Haswell	Skylake
$P = 1$	**Haraka-256 v2**	AES-NI	256	256	256	**1.25**	**0.72**
	Haraka-512 v2	AES-NI	512	512	256	**1.75**	**0.97**
	AESQ (from PAEQ)	AES-NI†	512	512	512	3.75	2.19
	Simpirav2[$b = 2$]	AES-NI	256	256	256	1.91	1.09
	Simpirav2[$b = 4$]	AES-NI	512	512	512	4.5	2.12
	SPHINCS-256-H	AVX2†	512	256	256	11.16	10.92
	SPHINCS-256-F	AVX2†	512	256	256	11.31	11.12
$P = 4$	**Haraka-256 v2**	AES-NI	1024	256	1024	1.12	0.63
	Haraka-512 v2	AES-NI	2048	512	1024	1.38	0.72
	Simpirav2[$b = 2$]	AES-NI	2048	512	1024	1.31	0.94
	Simpirav2[$b = 4$]	AES-NI	2048	512	1024	1.02	0.94
$P = 8$	**Haraka-256 v2**	AES-NI	2048	256	2048	1.14	0.66
	Haraka-512 v2	AES-NI	4096	512	2048	1.43	0.92
	Simpirav2[$b = 2$]	AES-NI	2048	256	2048	0.96	0.94
	Simpirav2[$b = 4$]	AES-NI	4096	512	4096	0.94	0.94
	SPHINCS-256-H	AVX2†	4096	256	2048	1.99	1.62
	SPHINCS-256-F	AVX2†	4096	256	2048	2.11	1.71

First of all, it is interesting to compare Haraka v2 to SPHINCS-256-H and SPHINCS-256-F from the SPHINCS-256 construction [BHH+15], which have identical functional signatures and similar design criteria to Haraka-512 v2 and Haraka-256 v2 respectively. If we first consider the performance using 8-way parallelization (i.e. using $P = 8$), we see from Table 5 that the SPHINCS functions have a performance of 1.62 cpb on Skylake for the H function and 1.71 cpb for the F function. These implementations do not utilize AVX-512 (employing 512-bit registers), so it is reasonable to assume their performance could be doubled under such circumstance. However, we note that even under this assumption, in both the cases of Haswell and Skylake, Haraka v2 performs favorably in comparison to those of SPHINCS-256.

In some applications, including some of the function calls in hash-based signatures, several calls to the short-input hash function *can not be parallelized*. To that end, it is of interest to compare the performance for Haraka v2, using $P = 1$, to the corresponding functions from SPHINCS-256. In this case, from the first part of Table 5, we see that Haraka-256 v2 performs very well with 1.25 cpb and 0.72 cpb on Haswell and Skylake, respectively, while the numbers for Haraka-512 v2 are 1.75 cpb on Haswell and 0.97 cpb on Skylake. From benchmarking the corresponding SPHINCS-256 functions on the same machines using $P = 1$, we obtain a performance of 11.16 cpb on Haswell and 10.92 cpb on Skylake for their H function, and respectively 11.31 and 11.12 cpb for their F function on Haswell and Skylake respectively. Thus, when the hash function calls in hash-based signatures can not be parallelized, Haraka v2 performs between 6.5 times and 15 times better on our tested platforms.

In Table 5, we compare the performance of Haraka v2 not only with the SPHINCS-256 functions, but also other designs which exhibit similar block-, input- and output sizes as Haraka v2. We comment on their benchmarks in the following.

With AESQ and Haraka v2 having very similar designs, the former having 20 AES rounds per block compared to Haraka v2 with 10, it is reasonable that AESQ is about twice as slow as Haraka-512 v2. The remaining margin can be explained by AESQ employing an evolving round key update rather than tabularized constants like Haraka v2.

Another close competitor is Simpira v2, which we implemented and benchmarked using $b = 2$ and $b = 4$, i.e. with two and four AES blocks respectively, thereby matching the sizes of the Haraka-256 v2 and Haraka-512 v2. Simpira v2 uses 15 AES rounds per block. Despite this being less than AESQ, it is slower when $P = 1$, because only one AES round (for $b = 2$) or two AES rounds (for $b = 4$) can be computed in parallel due to the Feistel structure. This is confirmed when we consider the performance of Simpira v2 with $P = 4$ and $P = 8$. For $b = 2$, parallelizing over $P = 4$ inputs brings the performance up to 2.37 cpb on Haswell and 1.62 cpb on Skylake. For $b = 4$, the performance is boosted up to 2.03 cpb and 1.17 cpb for $P = 4$ on Haswell and Skylake respectively. With $P = 8$, the effect of parallelization brings its performance up to 1.56 cpb for Haswell and 1.35 cpb for Skylake. We remark that in the Simpira v2 paper [GM16], the authors give their own benchmarks using $P = 4$ independent inputs. They report performance slightly better than ours, at 0.95 cpb for $b = 2$ and 0.94 cpb for $b = 4$ measured on a Skylake machine. However, as no source code was provided, we wrote our own optimized implementation.

From Table 5, we see that when multiple independent inputs are available for processing, the gap between the performance of Haraka v2 and of the Simpira v2 and SPHINCS functions diminishes. This makes sense as essentially processing multiple inputs gives a source of independence to draw on, allowing to parallelize instruction calls which would not otherwise be possible. As such, the throughput becomes more a question of the total number of instructions needed to obtain the desired security level, and less about the interplay of these instructions. With Haraka v2 still performing favorably, there are a couple of interesting observations. First, Haraka v2 performs better with $P = 4$ than with $P = 8$. This is simply due to the number of 128-bit registers available; with $P = 8$ more overhead occurs due to otherwise unnecessary read/write operations. Simpira v2 with its fewer parallel AES round applications in general need to process more independent inputs to achieve its optimal performance, as is evident from Table 5. Second, the best performance obtained overall is 0.63 cpb on Skylake with Haraka-256 v2 for $P = 4$. This matches very well with the theoretical maximum of $(20 \cdot 4)/(32 \cdot 4) = 0.625$ cycles per processed byte.

We considered also comparing against the recent KangarooTwelve extendable output hash from the Keccak team [BDP$^+$]. It uses 12 rounds of the Keccak permutation in a sponge construction, employing tree hashing when possible. However, for a short input of only 64 bytes, only one permutation call is needed and no parallelization can be made. The authors state a latency of ≈ 530 cycles in this case, yielding 8.28 cpb for a 512-bit input and half that performance for a 256-bit input, even when using Skylake-optimized implementations. In the Simpira v2 paper, the authors compare against an optimized SHA-256 implementation which is parallelized for $P = 4$ "long" inputs, claiming a performance of 2.35 cpb. For short inputs, and also considering the $P = 1$ cases, the performance would be reduced to a fraction of that, excluding also SHA-256 as a competitor. However, as no source code was provided, we were not able to verify against our own benchmarks. With this said, generic hash functions generally obtain their best performance for longer input, and for most such functions their poor performance on short inputs come as no surprise.

5.4 Performance of SPHINCS using Haraka v2

While the previous performance figures provide a good comparison between the functions themselves, the actual performance figures relevant for a hash-based signature scheme are the costs for key generation, signing and verifying a signature. The total costs for these operations are difficult to derive by only looking at the performance of the short-input hash function. For that reason, we modified the optimized AVX implementation of SPHINCS given in [BHH$^+$15], by replacing all calls to SPHINCS-256-F and SPHINCS-256-H by our implementations of Haraka-256 v2 and Haraka-512 v2 respectively. Parallel calls to these functions are processed to the same extent, using $P = 8$ calls at the same time, and no further optimizations have been applied. As can be seen in Table 6, the current performance gains by using Haraka v2 are between a factor of 1.50 to 2.86, depending on the platform and operation.

6 Conclusion and Remarks on Future Work

Together with in-depth implementation considerations on CPUs offering AES hardware acceleration, we presented the seemingly fastest proposal for compression/short-input hashing on our target platforms, with a performance of less than 1 cpb on a Skylake desktop CPU, both with and without parallelization across multiple inputs. As a concrete application of Haraka v2, we show that by using it inside the SPHINCS-256 hash-based signature, we can speed up the key generation, signing and verification operations by factors $\times 1.99$, $\times 1.87$ and $\times 2.86$, respectively, on Intel's Skylake architecture.

Table 6: Comparison of the AVX implementation of SPHINCS-256 with our implementation using Haraka v2. All numbers are given as the total number of cycles required, and are measured using SUPERCOP. The speed-up factor of operations are given in parentheses.

	Haswell		Skylake	
	SPHINCS-256	Haraka v2	SPHINCS-256	Haraka v2
Key generation	3,295,808	2,060,864 ($\times$1.60)	2,839,018	1,426,138 ($\times$1.99)
Signing	52,249,518	34,938,076 ($\times$1.50)	43,517,538	23,312,354 ($\times$1.87)
Verification	1,495,416	695,222 ($\times$2.15)	1,291,980	452,066 ($\times$2.86)

Despite having explored a larger design-space, Haraka v2 ended up having strong similarities with the AESQ permutation, used in the CAESAR candidate PAEQ [BK14]. All implementations for Haraka v2, including code for security analysis and for SPHINCS using Haraka v2, are publicly available [KLMR16].

We cover the important differential- and meet-in-the-middle attack vectors in our security analysis. We also give security arguments for Haraka v2 against various classes of attacks, without having to treat a large part of the hash function as a black box, as is the usual approach. This, of course, does not rule out attacks outside of the models that we consider. Hence, as for all other cryptographic primitives, more cryptanalysis is useful to establish more trust in the proposal.

Returning to the question: How much faster can a hash function become, if collision resistance is dropped from the list of requirements? With Haraka v2, we drop from $T = 6$ rounds to $T = 5$ rounds and still retain security against second-preimage attacks. We conclude that the performance gains are limited for the class of strategies considered, namely AES-like designs. This particularly holds when aiming at pre-quantum security levels higher than those for collision resistance, namely 256 bits rather than 128 bits. Aiming at higher security levels make sense, as there is evidence (at least for generic attacks), that the post-quantum security level will be 128 bits in both cases. 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.

References

AB12. Jean-Philippe Aumasson and Daniel J. Bernstein. SipHash: A Fast Short-Input PRF. In *Progress in Cryptology - INDOCRYPT*, pages 489–508, 2012.

BBG$^+$09. Ryad Benadjila, Olivier Billet, Henri Gilbert, Gilles Macario-Rat, Thomas Peyrin, Matt Robshaw, and Yannick Seurin. SHA-3 Proposal: ECHO. Submission to NIST (updated), 2009.

BCK96. Mihir Bellare, Ran Canetti, and Hugo Krawczyk. Keying Hash Functions for Message Authentication. In *CRYPTO '96*, pages 1–15, 1996.

BDH11. Johannes A. Buchmann, Erik Dahmen, and Andreas Hülsing. XMSS - A Practical Forward Secure Signature Scheme Based on Minimal Security Assumptions. In *Post-Quantum Cryptography - PQCrypto 2011*, pages 117–129, 2011.

BDP$^+$. Guido Bertoni, Joan Daemen, Michaël Peeters, Gilles Van Assche, and Ronny Van Keer. KangarooTwelve: fast hashing based on Keccak-p.

Be. Daniel J. Bernstein and Tanja Lange (editors). eBACS: ECRYPT Benchmarking of Cryptographic Systems. `http://bench.cr.yp.to`, accessed 19 November 2015.

Bel15. Mihir Bellare. New Proofs for NMAC and HMAC: Security without Collision Resistance. *J. Cryptology*, 28(4):844–878, 2015.

Ber09. Daniel J. Bernstein. Cost analysis of hash collisions: Will quantum computers make SHARCS obsolete? `https://cr.yp.to/hash/collisioncost-20090823.pdf`, 2009.

BHH$^+$15. Daniel J. Bernstein, Daira Hopwood, Andreas Hülsing, Tanja Lange, Ruben Niederhagen, Louiza Papachristodoulou, Michael Schneider, Peter Schwabe, and Zooko Wilcox-O'Hearn. SPHINCS: Practical Stateless Hash-Based Signatures. In *EUROCRYPT 2015*, pages 368–397, 2015.

BHT98. Gilles Brassard, Peter Høyer, and Alain Tapp. Quantum Cryptanalysis of Hash and Claw-Free Functions. In *LATIN '98: Theoretical Informatics*, pages 163–169, 1998.

BK14. Alex Biryukov and Dmitry Khovratovich. PAEQ. Submission to the CAESAR competition, 2014. `https://competitions.cr.yp.to/round1/paeqv1.pdf`.

BMS16. Nasour Bagheri, Florian Mendel, and Yu Sasaki. Improved Rebound Attacks on AESQ: Core Permutation of CAESAR Candidate PAEQ. In *Information Security and Privacy, ACISP 2016*, 2016.

BR97. Mihir Bellare and Phillip Rogaway. Collision-Resistant Hashing: Towards Making UOWHFs Practical. In *CRYPTO '97*, pages 470–484, 1997.

CLS06. Scott Contini, Arjen K. Lenstra, and Ron Steinfeld. VSH, an Efficient and Provable Collision-Resistant Hash Function. In *EUROCRYPT 2006*, pages 165–182, 2006.

CYK$^+$12. Jiali Choy, Huihui Yap, Khoongming Khoo, Jian Guo, Thomas Peyrin, Axel Poschmann, and Chik How Tan. SPN-Hash: Improving the Provable Resistance against Differential Collision Attacks. In *AFRICACRYPT 2012*, pages 270–286, 2012.

DEM16. Christoph Dobraunig, Maria Eichlseder, and Florian Mendel. Cryptanalysis of simpira v1. Cryptology ePrint Archive, Report 2016/244, 2016. `http://eprint.iacr.org/2016/244`.

DS11. Itai Dinur and Adi Shamir. An Improved Algebraic Attack on Hamsi-256. In *Fast Software Encryption, FSE 2011*, pages 88–106, 2011.

DSW08. Christophe De Canniere, Hisayoshi Sato, and Dai Watanabe. Hash Function Luffa: Supporting Document. Submission to NIST (Round 1), 2008.

Fuh10. Thomas Fuhr. Finding Second Preimages of Short Messages for Hamsi-256. In *ASIACRYPT 2010*, pages 20–37, 2010.

GKM$^+$11. Praveen Gauravaram, Lars R. Knudsen, Krystian Matusiewicz, Florian Mendel, Christian Rechberger, Martin Schläffer, and Søren S. Thomsen. Grøstl – a SHA-3 candidate. Submission to NIST (Round 3), 2011.

GM16. Shay Gueron and Nicky Mouha. Simpira v2: A Family of Efficient Permutations Using the AES Round Function. Cryptology ePrint Archive, Report 2016/122, 2016.

GPP11. Jian Guo, Thomas Peyrin, and Axel Poschmann. The PHOTON Family of Lightweight Hash Functions. In *CRYPTO 2011*, pages 222–239, 2011.

Gro96. Lov K. Grover. A Fast Quantum Mechanical Algorithm for Database Search. In *ACM Symposium on the Theory of Computing*, pages 212–219, 1996.

HHJ09. Shai Halevi, William E. Hall, and Charanjit S. Jutla. The Hash Function Fugue. Submission to NIST (updated), 2009.

HRS16. Andreas Hülsing, Joost Rijneveld, and Fang Song. Mitigating Multi-target Attacks in Hash-Based Signatures. In *Public-Key Cryptography - PKC 2016*, pages 387–416, 2016.

IBM16. IBM. ILOG CPLEX Optimizer, 2016.

Inc16. Gurobi Optimization Inc. Gurobi Optimizer Reference Manual, 2016.

Ind08. Sebastiaan Indesteege. The LANE hash function. Submission to NIST, 2008.

Jea16. Jérémy Jean. Cryptanalysis of Haraka. Cryptology ePrint Archive, Report 2016/396, 2016.

JN16. Jérémy Jean and Ivica Nikolic. Efficient Design Strategies Based on the AES Round Function. In *Fast Software Encryption, FSE 2016*, 2016.

JNP14. Jérémy Jean, María Naya-Plasencia, and Thomas Peyrin. Improved Cryptanalysis of AES-like Permutations. *J. Cryptology*, 27(4):772–798, 2014.

KLL$^+$14. Elif Bilge Kavun, Martin M. Lauridsen, Gregor Leander, Christian Rechberger, Peter Schwabe, and Tolga Yalçın. Prøst. Submission to the CAESAR competition, 2014. `https://competitions.cr.yp.to/round1/proestv1.pdf`.

KLMR16. Stefan Kölbl, Martin M. Lauridsen, Florian Mendel, and Christian Rechberger. Haraka code repository. `https://github.com/kste/haraka`, 2016.

Knu94. Lars R. Knudsen. Truncated and Higher Order Differentials. In *Fast Software Encryption, FSE 1994*, pages 196–211, 1994.

KS05. John Kelsey and Bruce Schneier. Second Preimages on n-Bit Hash Functions for Much Less than 2^n Work. In *EUROCRYPT 2005*, pages 474–490, 2005.

Kü09. Özgül Küçük. The Hash Function Hamsi. Submission to NIST (updated), 2009.

Lam79. Leslie Lamport. Constructing digital signatures from a one way function. Technical Report SRI-CSL-98, SRI International Computer Science Laboratory, 1979.

LSWD04. Tri Van Le, Rüdiger Sparr, Ralph Wernsdorf, and Yvo Desmedt. Complementation-Like and Cyclic Properties of AES Round Functions. In *Advanced Encryption Standard - AES, 4th International Conference, AES 2004*, pages 128–141, 2004.

MRST09. Florian Mendel, Christian Rechberger, Martin Schläffer, and Søren S. Thomsen. The Rebound Attack: Cryptanalysis of Reduced Whirlpool and Grøstl. In *Fast Software Encryption, FSE 2009*, pages 260–276, 2009.

MWGP11. Nicky Mouha, Qingju Wang, Dawu Gu, and Bart Preneel. Differential and Linear Cryptanalysis Using Mixed-Integer Linear Programming. In *Inscrypt 2011*, pages 57–76, 2011.

Nik14. Ivica Nikolić. Tiaoxin. Submission to the CAESAR competition, 2014. `https://competitions.cr.yp.to/round1/paeqv1.pdf`.

Pro. PQCRYPTO EU Project. Paris workshop, November 2015.

Rec10. Christian Rechberger. Second-Preimage Analysis of Reduced SHA-1. In *ACISP 2010*, pages 104–116, 2010.

Røn16. Sondre Rønjom. Invariant subspaces in simpira. *IACR Cryptology ePrint Archive*, 2016:248, 2016.

SA09. Yu Sasaki and Kazumaro Aoki. Finding Preimages in Full MD5 Faster Than Exhaustive Search. In *Advances in Cryptology - EUROCRYPT 2009*, pages 134–152, 2009.

Sas11. Yu Sasaki. Meet-in-the-Middle Preimage Attacks on AES Hashing Modes and an Application to Whirlpool. In *Fast Software Encryption - 18th International Workshop, FSE 2011*, pages 378–396, 2011.

SHW$^+$14. Siwei Sun, Lei Hu, Peng Wang, Kexin Qiao, Xiaoshuang Ma, and Ling Song. 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*, pages 158–178, 2014.

SLW$^+$10. Yu Sasaki, Yang Li, Lei Wang, Kazuo Sakiyama, and Kazuo Ohta. Non-full-active Super-Sbox Analysis: Applications to ECHO and Grøstl. In *ASIACRYPT 2010*, pages 38–55, 2010.

WFW$^+$12. Shuang Wu, Dengguo Feng, Wenling Wu, Jian Guo, Le Dong, and Jian Zou. (Pseudo) Preimage Attack on Round-Reduced Grøstl Hash Function and Others. In *Fast Software Encryption, FSE 2012*, pages 127–145, 2012.

WP14. Hongjun Wu and Bart Preneel. AEGIS. Submission to the CAESAR competition, 2014. `https://competitions.cr.yp.to/round1/aegisv1.pdf`.

WYY05. Xiaoyun Wang, Yiqun Lisa Yin, and Hongbo Yu. Finding Collisions in the Full SHA-1. In *CRYPTO 2005*, pages 17–36, 2005.

YWZW05. Hongbo Yu, Gaoli Wang, Guoyan Zhang, and Xiaoyun Wang. The Second-Preimage Attack on MD4. In *CANS 2005*, pages 1–12, 2005.

A Round Constants

Table 7: 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 Test Vectors for Haraka v2

Haraka-512 v2

```
Input:  00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f
        10 11 12 13 14 15 16 17 18 19 1a 1b 1c 1d 1e 1f
        20 21 22 23 24 25 26 27 28 29 2a 2b 2c 2d 2e 2f
        30 31 32 33 34 35 36 37 38 39 3a 3b 3c 3d 3e 3f
Output: be 7f 72 3b 4e 80 a9 98 13 b2 92 28 7f 30 6f 62
        5a 6d 57 33 1c ae 5f 34 dd 92 77 b0 94 5b e2 aa
```

Haraka-256 v2

```
Input:  00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f
        10 11 12 13 14 15 16 17 18 19 1a 1b 1c 1d 1e 1f
Output: 80 27 cc b8 79 49 77 4b 78 d0 54 5f b7 2b f7 0c
        69 5c 2a 09 23 cb d4 7b ba 11 59 ef bf 2b 2c 1c
```

C Active S-boxes

Table 8: 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

D Meet-in-the-middle attack on Haraka-512 v2

For Haraka-512 v2 we can attack 4 rounds in the following way (see Figure 6):

1. Randomly select values for the constant bytes in AC^4 ⬜.
2. For all 2^8 possible choices for $\mathsf{AC}^4_{*,0}$ which keep $\mathsf{MC}^4_{0,0}, \mathsf{MC}^4_{1,0}, \mathsf{MC}^4_{2,0}$ constant 🟦, compute forward to obtain the state in MC^0 and store the result in a table T.
3. For all 2^8 possible choices for $\mathsf{MC}^5_{*,4}$ which keep $\mathsf{AC}^5_{0,4}, \mathsf{AC}^5_{1,4}, \mathsf{AC}^5_{2,4}$ constant 🟥, compute backward to obtain the state in AC^0.
4. Check if there is an entry in T that matches with AC^0 through $\mathsf{MixColumns}$. If so check whether the remaining bytes also match, otherwise repeat from step 2 (or step 1 if necessary).

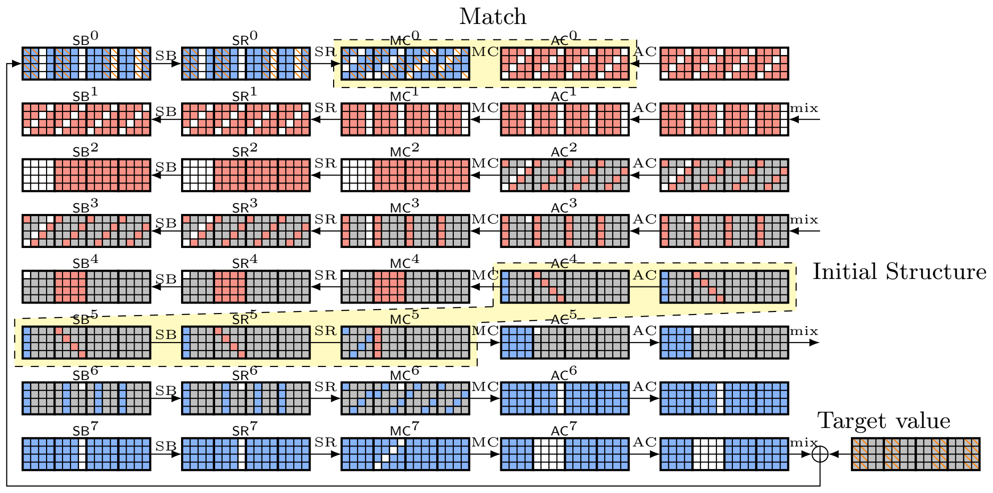


Fig. 6: Meet-in-the-middle attack on 4 rounds of Haraka-512 v2. All □ are unknown, ⬜ are constant, 🟥 neutral bytes backward, 🟦 neutral bytes forward and ▨ are the bytes truncated at the output.

Complexity. Computing the table T and 2^8 values for AC^0 costs 2^8 4-round Haraka-512 v2 evaluations and requires $2^8 \cdot 16$ bytes of memory. The success probability for the match is 2^{-32} for each state. Hence, on average $2^8 \cdot 2^8 \cdot 2^{-128} = 2^{-112}$ candidates will remain in Step 4. There are still 12 byte conditions which have to be satisfied in each state, which can be reduced to 4 byte conditions by using the fact that we can freely choose those bytes which are truncated at the output. Therefore, if we repeat step 1-4 2^{240} times we expect to find $2^{240} \cdot 2^{-112} \cdot 2^{-4 \cdot 4 \cdot 8} = 1$ solution. The overall complexity is $2^{240} \cdot 2^8 = 2^{248}$ evaluations of Haraka-512 v2 to find a preimage.

E 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.

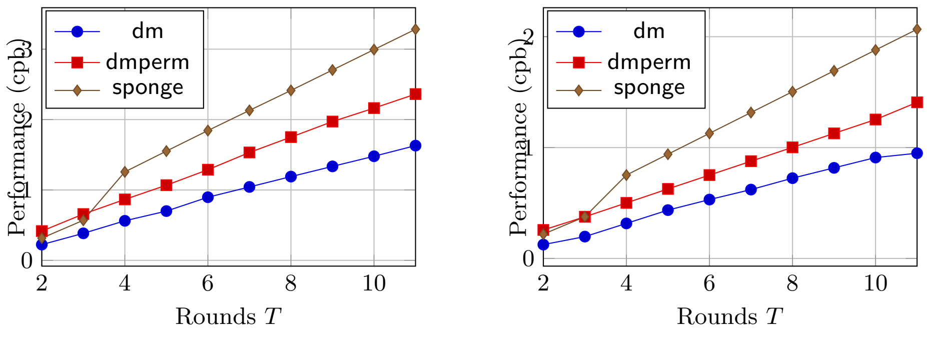
In order to avoid the negative consequences on security from a too simple message expansion, and to performance from a too complex message expansion, we opted to abandon the block cipher-based approach of (1) in favor of a permutation-based approach. In particular, we load the full message into the state of the permutation from the beginning. As such, the state size for Haraka-512 v2 must be at least 64 bytes, while that of Haraka-256 v2 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 v2 and Haraka-256 v2.

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 5.2, 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 $y = \textbf{blend}_w(a, b)$ 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 $\textbf{blend}_w$ 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 7 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 v2 and Haraka-256 v2.



(a) Haswell (b) Skylake

Fig. 7: Performance using $m = 2$ for each of the three general Haraka-512 v2 constructions considered

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

1. The $\textbf{mix}_{512}$ and $\textbf{mix}_{256}$ 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 8. On both the Haswell and Skylake microarchitectures, the instructions used for those three approaches all have a latency of one clock cycle, while the inverse throughput varies from e.g. 0.33 instructions/cycle for the XOR operation to 1 instruction/cycle for the punpckhdq and punpckldq instructions.

Figure 9 gives a performance comparison of the three approaches to the linear mixing layer. As shown, with the exception of the $\textbf{mix}_{512}$ operation on Haswell, all other approaches

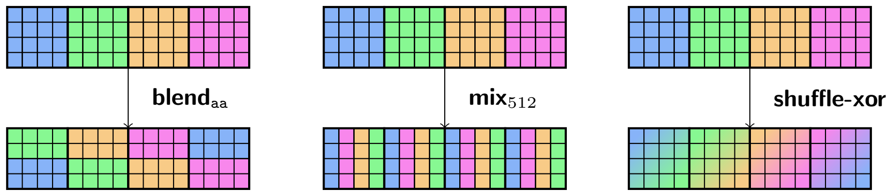
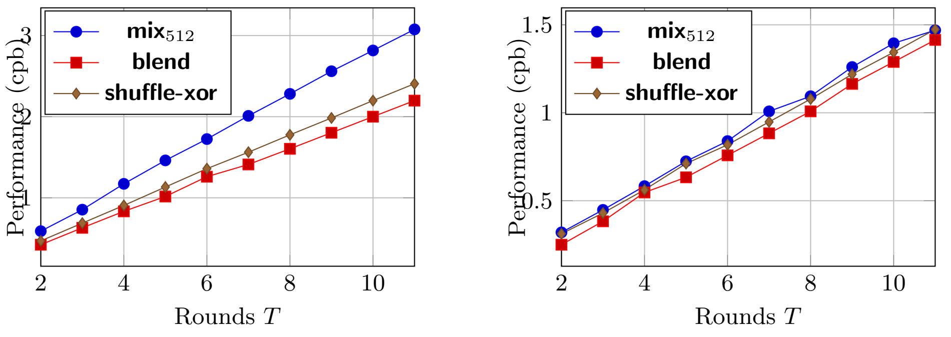


Fig. 8: Effect of applying one round of the mixing layers on the state of π_{512}.



(a) Haswell (b) Skylake

Fig. 9: Performance of Haraka-512 v2 using $m = 2$ for each of the three approaches to linear mixing considered

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.