Attacking the Knudsen-Preneel Compression Functions*

Onur Özen[1†] and Thomas Shrimpton[2‡] and Martijn Stam[1]

[1] EPFL IC IIF LACAL, Station 14, CH-1015 Lausanne, Switzerland
{onur.ozen, martijn.stam}@epfl.ch
[2] Dept. of Computer Science, Portland State University, Room 120, Forth Avenue Building, 1900 SW 4th Avenue, Portland OR 97201 USA
teshrim@cs.pdx.edu

Abstract. Knudsen and Preneel (Asiacrypt'96 and Crypto'97) introduced a hash function design in which a linear error-correcting code is used to build a wide-pipe compression function from underlying blockciphers operating in Davies-Meyer mode. Their main design goal was to deliver compression functions with collision resistance up to, and even beyond, the block size of the underlying blockciphers. In this paper, we (re)analyse the preimage resistance of the Knudsen-Preneel compression functions in the setting of public random functions.

We give a new preimage attack that is based on two observations. First, by using the right kind of queries it is possible to mount a non-adaptive preimage attack that is *optimal* in terms of query complexity. Second, by exploiting the *dual code* the subsequent problem of reconstructing a preimage from the queries can be rephrased as a problem related to the generalized birthday problem. As a consequence, the time complexity of our attack is intimately tied to the minimum distance of the dual code.

Our new attack consistently beats the one given by Knudsen and Preneel (in one case our preimage attack even beats their *collision* attack) and demonstrates that the gap between their claimed collision resistance and the actual preimage resistance is surprisingly small. Moreover, our new attack falsifies their (conjectured) preimage resistance security bound and shows that intuitive bounds based on the number of 'active' components can be treacherous.

Complementing our attack is a formal analysis of the query complexity (both lower and upper bounds) of preimage-finding attacks. This analysis shows that for many concrete codes the time complexity of our attack is optimal.

1 Introduction

Cryptographic hash functions remain one of the most used cryptographic primitives, and the design of provably secure hash functions (relative to various security notions) is an active area of research. From an appropriate perspective, most hash function designs can be viewed as the Merkle-Damgård iteration of a blockcipher-based compression function (where a single permutation can be regarded as a degenerate or fixed key blockcipher). The classical PGV blockcipher-based compression functions [22] have an output size matching the blocksize n of the underlying blockcipher. Yet even for the optimally secure ones [1, 29], the (time) complexity of collision- and preimage-finding attacks is at most $2^{n/2}$, resp. 2^n; when $n = 128$ (e.g. AES) the resulting bounds have been deemed unacceptable for current practice (cf. [9] for a recent survey of recommended cryptographic key sizes).

This mismatch between desired output sizes for blockciphers versus hash functions was recognized early on (dating back to Yuval [33]), and blockcipher-based compression functions that output more than n bits have been proposed. This output expansion is typically achieved by calling the blockcipher multiple times and then combining the resulting blockcipher outputs in some clever way. In the 1990s many so-called double-length constructions (where $2n$ bits are output) were put forth, but large classes of these were subsequently broken. Only recently have a few double-length constructions been supported by formal security proofs; see e.g. [7, 8, 10, 19, 21] for an overview. In any case, the standard approach in designing wider-output compression functions has been to fix a target output size (and often a target number of blockcipher calls as well) and then to try to build a compression function that is optimally collision-resistant for that size.

* The work described in this paper has been supported in part by the European Commission through the ICT programme under contract ICT-2007-216676 ECRYPT II.
† Supported by a grant of the Swiss National Science Foundation, 200021-122162.
‡ Supported by NSF grants CNS-0627752 and CNS-0845610.

Table 1. Knudsen-Preneel constructions (cf. [13, Table V]) based on $2n$-to-n bit primitive (PuRF or single-key blockcipher). Non-MDS parameters in italic.

Code	$sn + mn \to sn$	Preimage Resistance			
		Query Complexity	Our Attack Time	KP-Conj. Time	KP-Attack Time
$[r, k, d]_{2^e}$	$2kn \to rn$	$2^{rn/k}$	Sec. 6, 7	$2^{(d-1)n}$	Thm. 7
$[5, 3, 3]_4$	$(5+1)n \to 5n$	$2^{5n/3}$	$2^{5n/3}$	2^{2n}	2^{2n}
$[\mathit{8}, \mathit{5}, \mathit{3}]_4$	$(8+2)n \to 8n$	$2^{8n/5}$	$2^{8n/5}$	2^{2n}	2^{3n}
$[\mathit{12}, \mathit{9}, \mathit{3}]_4$	$(12+6)n \to 12n$	$2^{4n/3}$	$2^{4n/3}$	2^{2n}	2^{3n}
$[\mathit{9}, \mathit{5}, \mathit{4}]_4$	$(9+1)n \to 9n$	$2^{9n/5}$	$2^{11n/5}$	2^{3n}	2^{4n}
$[\mathit{16}, \mathit{12}, \mathit{4}]_4$	$(16+8)n \to 16n$	$2^{4n/3}$	$2^{7n/3}$	2^{3n}	2^{4n}
$[6, 4, 3]_{16}$	$(6+2)n \to 6n$	$2^{3n/2}$	$2^{3n/2}$	2^{2n}	2^{2n}
$[8, 6, 3]_{16}$	$(8+4)n \to 8n$	$2^{4n/3}$	$2^{4n/3}$	2^{2n}	2^{2n}
$[12, 10, 3]_{16}$	$(12+8)n \to 12n$	$2^{6n/5}$	$2^{6n/5}$	2^{2n}	2^{2n}
$[9, 6, 4]_{16}$	$(9+3)n \to 9n$	$2^{3n/2}$	2^{2n}	2^{3n}	2^{3n}
$[16, 13, 4]_{16}$	$(16+10)n \to 16n$	$2^{16n/13}$	2^{2n}	2^{3n}	2^{3n}

In three papers [11,12,13], Knudsen and Preneel adopted a different approach, namely to let the output size and (relatedly) the number of blockcipher calls vary as needed in order to guarantee a particular security target. Specifically, given r independent ideal compression functions $f_1, \ldots, f_r$, each mapping cn-bits to n bits, they create a new 'bigger' compression function outputting rn bits.[3] The $f_1, \ldots, f_r$ are run in parallel, and each of their inputs is some linear combination of the blocks of message and chaining variable that are to be processed. The rn-bit output of their construction is the concatenation of the outputs of these parallel calls.

The elegance of the KP construction is in *how* the inputs to $f_1, \ldots, f_r$ are computed: a generator matrix of an $[r, k, d]$ error-correcting code over $\mathbb{F}_{2^c}$ determines how the ck input blocks of the 'big' compression function are xor'ed together to form the inputs to the underlying r functions. (In a generalization they consider the f_i as mapping from bcn' to bn' bits instead and use a code over $\mathbb{F}_{2^{bc}}$.) The deliberate effect of this design is that when two inputs to the 'big' compression function differ, the corresponding inputs for the underlying functions will differ for at least d functions. In particular, when using a systematic generator, a change in the systematic part of the input results in at least $d-1$ so-called active functions in the non-systematic part. Intuitively this means that a preimage-finding (resp. collision-finding) attack must at least find a preimage (resp. collision) for the $d-1$ active functions in parallel.

Under a broad—but *prima facie* not unreasonable—assumption related to the complexity of finding collisions in parallel compression functions, Knudsen and Preneel show that any attack needs time at least $2^{(d-1)n/2}$ to find a collision in their construction. Thus for a code with minimum distance $d = 3$, one obtains a 2^n collision-resistance bound. For preimage resistance, Knudsen and Preneel *conjecture* that attacks will require at least $2^{(d-1)n}$ time. They also give preimage- and collision-finding attacks that, curiously, are mostly independent of the minimum distance. For preimage resistance the attacks of Knudsen and Preneel meet their conjectured bound (at least for MDS codes). For collision-resistance the story is different, as for many of the codes considered there is a significant gap between the actual complexity of their attacks and their $2^{(d-1)n/2}$ bound. Watanabe [31] has subsequently shown a collision attack that is more efficient than the one given by Knudsen and Preneel for many of their parameter sets. He was even able to show that their collision resistance bound is wrong for certain parameters (e.g. for $3 < d \leq k$).

This paper offers a new security analysis of the KP construction when the underlying compression functions are modeled as public random functions (PuRFs). In the process we also introduce a precise formalization of blockwise-linear schemes for transforming parallel PuRFs into compression func-

[3] Note that Knudsen and Preneel also propose to instantiate the underlying ideal compression functions with a blockcipher run in Davies-Meyer mode and to iterate the compression function to obtain a full blockcipher-based hash function (when iterated, one could compress the final state to a desired length for the security target).

Table 2. Knudsen-Preneel constructions (cf. [13, Table VIII]) based $3n$-to-n bit primitive (PuRF or double-key blockcipher).

Code	$sn + mn \to sn$	Preimage Resistance			
		Query Complexity	Our Attack Time	KP-Conj. Time	KP-Attack Time
$[r, k, d]_{2^e}$	$3kn \to rn$	$2^{rn/k}$	Sec. 6, 7	$2^{(d-1)n}$	Thm. 7
$[4, 2, 3]_8$	$(4+2)n \to 4n$	2^{2n}	2^{2n}	2^{2n}	2^{2n}
$[6, 4, 3]_8$	$(6+6)n \to 6n$	$2^{3n/2}$	$2^{3n/2}$	2^{2n}	2^{2n}
$[9, 7, 3]_8$	$(9+12)n \to 9n$	$2^{9n/7}$	$2^{9n/7}$	2^{2n}	2^{2n}
$[5, 2, 4]_8$	$(5+1)n \to 5n$	$2^{5n/2}$	2^{3n}	2^{3n}	2^{3n}
$[7, 4, 4]_8$	$(7+5)n \to 7n$	$2^{7n/4}$	$2^{9n/4}$	2^{3n}	2^{3n}
$[10, 7, 4]_8$	$(10+11)n \to 10n$	$2^{10n/7}$	2^{2n}	2^{3n}	2^{3n}

tions, of which the KP construction is one example. We directly address the KP-conjectured preimage-resistance by describing new attacks that go well *below* the conjectured complexity lower bound. This demonstrates its incorrectness and, more generally, suggests caution when considering intuition about hash-function security that derives from the number of active functions. Our attacks explicitly take into account both query *and* time-complexity. We see the latter as especially important when considering attacks.

Our new attack. So, our main result is a new preimage attack whose time and query complexities (ignoring the constant and logarithmic factors) are summarized[4] in Tables 1 and 2. From a practical point of view, the time complexity of our attack beats the one given by KP in every case but two, namely when the code is $[4, 2, 3]_8$ or $[5, 2, 4]_8$; in both cases we match the original attack, moreover we show that it is optimal for the former. Startlingly, in the $[12, 9, 3]_4$ case our preimage attack is *even faster than the collision attack* proposed by Knudsen and Preneel. In that case we have uncovered a new collision attack as well! The memory requirements of our attack, on the other hand, are typically higher than what Knudsen and Preneel require in their attacks. Let us give a high-level summary of the attack and what is done to reduce the overall complexity of it.

Reducing the query complexity. We begin with the simple observation that $(0^a \,||\, x_1) \oplus (0^a \,||\, x_2)$ yields a string of the form $(0^a \,||\, X)$. More generally, any linear combination of strings with the same pattern of fixed zero bits will yield a string with the same form. By restricting PuRF queries to strings with the same (blockwise) pattern we can optimize the *yield* of these queries, or the maximum number of KP compression function evaluations an adversary can compute for a given number of queries. This observation allows us to reduce the query complexity of a preimage-finding attack to the bare minimum and, significantly, and also allows the attack to be deterministic and non-adaptive.

These results (fully detailed in Section 4) are relevant beyond the KP construction. They apply to all constructions in which the inputs to the underlying idealized primitives (blockciphers or PuRFs) are determined by *blockwise* linear combinations of blocks of compression function input. This includes the schemes discussed by Peyrin et al. [21] and Seurin and Peyrin [26].

Exploiting the dual code to reduce the time complexity. When mounting our reduced-query attack against a KP construction with parameters $[r, k, d]_{2^e}$, the result is r lists with partial preimages (under each of the f_i respectively) and, with high probability, a full preimage is 'hiding' among these lists. That is to say, when we consider all possible combinations of partial preimages, some will correspond to a codeword and others will not. To reduce the time complexity of our attack, we need to be able to find a codeword (which is itself a full preimage) efficiently among all possibilities.

Finding full preimages from the lists of partial preimages is the core innovation of our attack. (Sections 6 and 7 give the details.) We exploit the fact that codewords in the *dual* code can be used to express

[4] We note that our attack is not specific to the cases $c \in \{2, 3\}$, it also works (for instance) against the compression functions suggested by Knudsen and Preneel with $c = 5$ (mimicking the MD4 and MD5 situation).

relations among PuRF-inputs that correspond to a preimage.We also leverage known techniques for solving the generalized birthday problem (see e.g. [30, 25, 4, 3]) in order to prune the partial-preimage lists, and consequently find a preimage for the compression function faster (than a naive approach or that of KP).

Here is a concrete example to give some quick, initial intuition. For the $[5, 3, 3]_4$ code given by Knudsen and Preneel [13, Section C], we observe that the (five) inputs to the PuRFs satisfy the equation $x_1 \oplus x_2 \oplus x_3 \oplus x_4 = 0$. This relationship extends to the lists of partial preimages, so we begin by enumerating all solutions to this equation using the elements of the first four lists. Known techniques indicate that this takes time only quadratic in the size of the lists. We finalize the preimage search by checking whether or not any of these solutions can be extended in a way that their fifth component x_5 is in the fifth list of partial preimages.

In fact, the relation used in this example corresponds to a codeword in the dual code. We'll see that using a dual codeword of minimal distance allows the 'merge' stage of the attack (i.e. enumerating all solutions) to be most efficient. Thus the minimum *dual* distance plays an important role in expressing the complexity of our attack. Furthermore, for maximum distance separable (MDS) codes we show that one merge stage is sufficient, and we can finalize the preimage search straight away (as in the example above). For non-MDS codes a second stage of merging may be necessary to improve the overall complexity of the preimage-finding attack. In non-MDS case expressing algebraically the time-complexity does become a bit unwieldy; nevertheless, it is always easy to evaluate once given the compression function generator matrix.

Extensions. Finally, we note that our *attacks* should carry over to the blockcipher setting without much extra work; adapting the security proofs rigourously is less straightforward. The attacks also carry over to the iterative setting, provided the adversary gets to choose the initial vector (free-start scenario); for a fixed IV we do not believe that our attacks work (however, a faster preimage attack on the compression function as we present it can be used in the Lai-Massey meet-in-the-middle attack [14]).

Proving optimality of our attack. A secondary result of this paper is a preimage-resistance security proof for the Knudsen-Preneel compression functions. In Theorem 10 we determine a lower bound on the query complexity of preimage-finding attacks, even attacks mounted by computationally unbounded adversaries. The theorem gives a concrete bound, and a related corollary provides an asymptotic assessment of it that is easier to grasp. In particular, it shows that the query complexity of our new attack is essentially optimal (up to a small factor).

Since the lower bounds on the query complexity serve as 'best case' lower bounds for the complexity of real-world attacks, we can conclude that our new preimage-finding attack is *optimal* whenever the time complexity of our attack matches its query complexity. This happens for 9 out of the 16 schemes: for the seven MDS schemes with $d = 3$, and for codes $[8, 5, 3]_4$ and $[12, 9, 3]_4$. For the remaining seven schemes we leave a gap between the information-theoretic lower bound and the real-life upper bound.

2 Preliminaries

Compression functions. A *compression function* is a mapping $H : \{0,1\}^{tn} \to \{0,1\}^{sn}$ for some blocksize[5] $n > 0$ and integer parameters $t > s > 0$. For positive integers c and n, we let $\mathrm{Func}(cn, n)$ denote the set of all functions mapping $\{0,1\}^{cn}$ into $\{0,1\}^n$. A compression function is *PuRF-based* if its mapping is computed by a program with oracle access to a finite number of specified oracles $f_1, \ldots, f_r$, where $f_1, \ldots, f_r \xleftarrow{\$} \mathrm{Func}(cn, n)$. When a PuRF-based compression function operates on input W, we write $H^{f_1,\ldots,f_r}(W)$ for the resulting value. Of primary interest for us will be *single-layer* PuRF-based compression functions without feedforward. These call all oracles in parallel and compute the output based only on the results of these calls; in particular, input to the compression function is not further considered.

[5] We include the blocksize in the definition for convenience later on—it is not a necessity and only mildly restrictive.

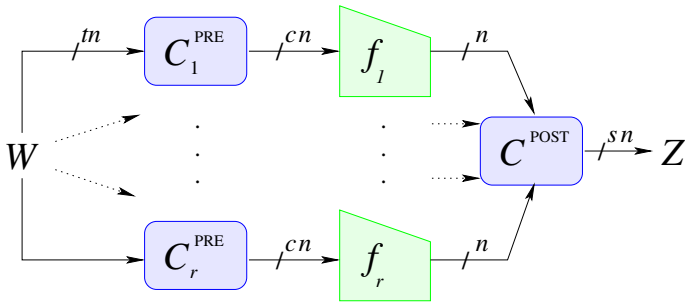

Fig. 1. General form of an tn-to-sn bit single layer PuRF-based compression function without feedforward based on r calls to underlying PuRFs with cn-bit inputs and n-bit outputs.

More formally, let $C_i^{\mathrm{PRE}} : \{0,1\}^{tn} \to \{0,1\}^{cn}$, for $i = 1, \ldots, r$, and $C^{\mathrm{POST}} : (\{0,1\}^n)^r \to \{0,1\}^{sn}$, be pre- and postprocessing functions respectively. Then given an tn-bit input W, compute output $Z = H^{f_1,\ldots,f_r}(W)$ as follows: for $i = 1, \ldots, r$ let $x_i \leftarrow C_i^{\mathrm{PRE}}(W)$ and $y_i \leftarrow f_i(x_i)$; return $Z \leftarrow C^{\mathrm{POST}}(y_1, \ldots, y_r)$. This is illustrated in Figure 1.

Blockwise-linear schemes. Most PuRF-based (and blockcipher-based) compression functions are of a special type. Instead of arbitrary pre- and postprocessing, one finds only functions that are blockwise linear. (The PGV hash functions are simple examples of this.) An advantage of a blockwise approach is that it yields simple-looking hash functions whose security is easily seen to be determined by the blocksize n. Linearity allows for relatively efficient implementation via bitwise exclusive-or of n-bit blocks. (In contrast, Lucks's and Stam's double-length schemes [15, 29] require full $\mathbb{F}_{2^n}$-arithmetic.) The Knudsen-Preneel construction is also blockwise linear, so let us define formally what is a blockwise-linear single-layer PuRF-based compression function without feedforward, an unwieldy name we shorten to *blockwise-linear scheme*. [6]

Definition 1 (Blockwise-linear scheme). *Let r, c, b, t, s be positive integers and let matrices $\mathsf{C}^{\mathrm{PRE}} \in \mathbb{F}_2^{rcb \times tb}$, $\mathsf{C}^{\mathrm{POST}} \in \mathbb{F}_2^{sb \times rb}$ be given. We define $H = \mathsf{BL}^b(\mathsf{C}^{\mathrm{PRE}}, \mathsf{C}^{\mathrm{POST}})$ to be a family of single-layer PuRF-based compression functions $H_n : \{0,1\}^{tn} \to \{0,1\}^{sn}$, for all positive integers n with $b|n$. Specifically, let $n'b = n$, and $f_1, \ldots, f_r \in \mathrm{Func}(cn, n)$. Then on input $W \in \{0,1\}^{tn}$ (interpreted as column vector), ${H_n}^{f_1 \cdots f_r}(W)$ computes the digest $Z \in \{0,1\}^{sn}$ as follows:*

1. *Compute $X \leftarrow (\mathsf{C}^{\mathrm{PRE}} \otimes I_{n'}) \cdot W$;*
2. *Parse $X = (x_i)_{i=1\ldots r}$ and for $i = 1 \ldots r$ compute $y_i = f_i(x_i)$;*
3. *Parse $(y_i)_{i=1\ldots r} = Y$ and output $Z = (\mathsf{C}^{\mathrm{POST}} \otimes I_{n'}) \cdot Y$.*

where $\otimes$ denotes the Kronecker product and $I_{n'}$ the identity matrix in $\mathbb{F}_2^{n' \times n'}$.

In the definition above we silently identified $\{0,1\}^n$ with the vector space $\mathbb{F}_2^n$, etc. The map corresponding to $(\mathsf{C}^{\mathrm{PRE}} \otimes I_{n'})$ will occasionally be denoted C^{PRE}. It will be convenient for us to write the codomain of C^{PRE} as a direct sum, so we identify $\{0,1\}^{rcn}$ with $\bigoplus_{i=1}^r V_i$ where $V_i = \mathbb{F}_2^{cn}$ for $i = 1, \ldots, r$. If $x_1 \in V_1$ and $x_2 \in V_2$, then consequently $x_1 + x_2$ will be in $V_1 \oplus V_2$. (This extends naturally to $L_1 + L_2$ when $L_1 \subset V_1, L_2 \subset V_2$.) If we want to add 'normally' in $\mathbb{F}_2^{cn}$ we write $x_1 \oplus x_2$ which conveniently corresponds to exclusive or and the result will be in $\mathbb{F}_2^{cn}$ as expected.

Security notions. A *preimage-finding adversary* is an algorithm with access to one or more oracles, and whose goal is to find a preimage of some specified compression function output. We will consider adversaries in two scenarios: the information-theoretic one and a more realistic concrete setting. For information-theoretic adversaries the only resource of interest is the number of queries made to their oracles. Otherwise, these adversaries are considered (computationally) unbounded. In the concrete setting

[6] Of course there is no real need to restrict to single-layer schemes without feedforward, but it suffices for our purposes.

we consider the actual runtime of the adversarial algorithm (relative to some understood and reasonable computational model) and, to a lesser extent, its memory consumption (and code-size[7]). Without loss of generality, in both settings adversaries are assumed not to repeat queries to oracles nor to query an oracle outside of its specified domain.

There exist several definitions of preimage resistance, depending on the distribution of the element for which a preimage needs to be found. The strongest notion is that preimage resistance should hold with respect to any distribution, which can be formalized as everywhere preimage resistance [23].

Definition 2 (Everywhere preimage resistance). *Let $c, r, s, t > 0$ be integer parameters, and fix a blocksize $n > 0$. Let $H\colon \{0,1\}^{tn} \to \{0,1\}^{sn}$ be a PuRF-based compression function taking r oracles $f_1, \ldots, f_r \in \mathrm{Func}(cn, n)$. The everywhere preimage-finding advantage of adversary $\mathcal{A}$ is defined to be*

$$\mathsf{Adv}^{\mathsf{epre}}_H(\mathcal{A}) = \max_{Z \in \{0,1\}^{sn}} \Big\{ \Pr\Big[f_1 \ldots f_r \xleftarrow{\$} \mathrm{Func}(cn, n), (W') \leftarrow \mathcal{A}^{f_1 \cdots f_r}(Z) : Z = H^{f_1 \cdots f_r}(W') \Big] \Big\}$$

Define $\mathsf{Adv}^{\mathsf{epre}}_H(q)$ and $\mathsf{Adv}^{\mathsf{epre}}_H(t)$ as the maximum advantage over all adversaries making at most q queries to each of their oracles respectively running in time at most t.

A *collision-finding adversary* is an algorithm whose goal is to find collisions in some specified compression or hash function.

Definition 3 (Collision resistance). *Let $c, r, s, t > 0$ be integer parameters, and fix a blocksize $n > 0$. Let $H\colon \{0,1\}^{tn} \to \{0,1\}^{sn}$ be a PuRF-based compression function taking r oracles $f_1, \ldots, f_r \in \mathrm{Func}(cn, n)$. The collision-finding advantage of adversary $\mathcal{A}$ is defined to be*

$$\mathsf{Adv}^{\mathsf{coll}}_H(\mathcal{A}) = \Pr\Big[f_1 \ldots f_r \xleftarrow{\$} \mathrm{Func}(cn, n), (W, W') \leftarrow \mathcal{A}^{f_1 \cdots f_r} : W \neq W' \text{ and } H^{f_1 \cdots f_r}(W) = H^{f_1 \cdots f_r}(W') \Big]$$

Define $\mathsf{Adv}^{\mathsf{coll}}_H(q)$ as the maximum advantage over all adversaries making at most q queries to each of their oracles and $\mathsf{Adv}^{\mathsf{coll}}_H(t)$ as the maximum advantage over all adversaries running in time at most t.

Instead of giving a full characterization of e.g. $\mathsf{Adv}^{\mathsf{epre}}_H(q)$, we will leave our statements somewhat informal—in accordance with prior art in the field—and say that finding a preimage takes at least (in the case of security proofs), or at most (in the case of attacks), so many queries. In the backs of our minds, we can regard the blocklength as a security parameter (for blockwise schemes at least) and thus we can look at the advantage as a function in n. The statement that at least (say) $2^{\delta n}$ queries are needed (to find preimages) then means that for all $\epsilon < \delta$ it holds that $\mathsf{Adv}^{\mathsf{epre}}_H(2^{\epsilon n})$ vanishes. (Consequently, we are unconcerned about constant or even polylogarithmic factors.)

Iterated hash functions. A compression function $H : \{0,1\}^{tn} \to \{0,1\}^{sn}$ can be made into a hash function by iterating it. For this let $m = s - t$ and identify $\{0,1\}^{tn}$ with $\{0,1\}^{mn} \times \{0,1\}^{sn}$. We briefly recall the standard Merkle-Damgård iteration [17,5] where we assume that there is already some injective padding from $\{0,1\}^* \to (\{0,1\}^{mn})^* \backslash \emptyset$ in place (note that we disallow the empty message $\mathbf{M} = \emptyset$, corresponding to $\ell = 0$, as output of the injective padding). Given an initial vector $V_0 \in \{0,1\}^{sn}$ define $\mathcal{H}^H : (\{0,1\}^{mn})^* \to \{0,1\}^{sn}$ as follows for $\mathbf{M} = (M_1, \ldots, M_\ell)$: for $i = 1, \ldots, \ell$, set $V_i \leftarrow H(M_i, V_{i-1})$; output $\mathcal{H}^H(\mathbf{M}) = V_\ell$.

For the iterated hash function $\mathcal{H}$ corresponding advantages can be expressed. In this case there is a choice how to deal with the initial vector in the definition: either adversarially chosen, randomly chosen, or fixed. Like Knudsen and Preneel, we will concentrate on the first option, in which case preimage and collision security of the iterated hash function are equivalent to that of the compression function [1,29]; henceforth we will concern ourselves with compression functions only. (For a fixed initial vector security of the compression function implies that of the iterated hash, but a break of the compression function is not guaranteed to carry over [1].)

[7] We force algorithms to read their own code so the runtime is naturally lower bounded by the code-size.

Linear error correcting codes. An $[r, k, d]_{2^e}$ linear error correcting code $\mathcal{C}$ is the set of elements (codewords) in a k-dimensional subspace of $\mathbb{F}_{2^e}^r$, where the minimum distance d is defined as the minimum Hamming weight (taken over all nonzero codewords in $\mathcal{C}$). The dual code $[r, r-k, d^\perp]_{2^e}$ is the set of all elements in the $r-k$-dimensional subspace orthogonal to $\mathcal{C}$ (with respect to the usual inner product), and its minimum distance is denoted $d^\perp$.

Not all parameter sets are possible, in particular $r \geq k$ (trivial) and the Singleton bound puts a (crude) limit on the minimum distance $d \leq r - k + 1$. Codes matching the Singleton bound are called maximum distance separable (MDS). An important property of MDS codes is that their duals are MDS as well, so $d^\perp = k + 1$.

An $[r, k, d]_{2^e}$ code $\mathcal{C}$ can be generated by a matrix $G \in \mathbb{F}_{2^e}^{k \times r}$, meaning that $\mathcal{C} = \{x \cdot G | x \in \mathbb{F}_{2^e}^k\}$ (using row vectors throughout). Without loss of generality, we restrict ourselves to systematic generator matrices, that is $G = [I_k | P]$ for $P \in \mathbb{F}_{2^e}^{k \times (r-k)}$ and I_k the identity matrix in $\mathbb{F}_{2^e}^{k \times k}$.

3 The Knudsen-Preneel Hash Functions

Knudsen and Preneel [11, 12] introduced a family of hash functions employing error correcting codes. (We use the journal version [13] as our frame of reference). Although their work was ostensibly targeted at blockcipher-based designs, the main technical thread of their work develops a transform that extends the range of an 'ideal' compression function (blockcipher-based, or not) in a manner that delivers some target level of security. As is nowadays typical, we understand an ideal compression function to be a PuRF. In fact, the KP transform is a special instance of a blockwise-linear scheme (Definition 1), in which the the inputs to the PuRFs are determined by a linear code over a binary field with extension degree $e > 1$, i.e. $\mathbb{F}_{2^e}$, and with C^{POST} being the identity matrix over $\mathbb{F}_2^{rb \times rb}$ (corresponding to concatenating the PuRF outputs). The extension field itself is represented as a subring of the matrix ring (of dimension equalling the extension degree) over the base field. We formalize this by an injective ring homomorphism $\varphi : \mathbb{F}_{2^e} \to \mathbb{F}_2^{e \times e}$ and let $\bar{\varphi} : \mathbb{F}_{2^e}^{r \times k} \to \mathbb{F}_2^{re \times ke}$ be the component-wise application of φ and subsequent identification of $(\mathbb{F}_2^{e \times e})^{r \times k}$ with $\mathbb{F}_2^{re \times ke}$ (we will use $\bar{\varphi}$ for matrices over $\mathbb{F}_{2^e}$ of arbitrary dimensions).

Definition 4 (Knudsen-Preneel transform). *Let $[r, k, d]$ be a linear code over $\mathbb{F}_{2^e}$ with generator matrix $G \in \mathbb{F}_{2^e}^{k \times r}$. Let $\varphi : \mathbb{F}_{2^e} \to \mathbb{F}_2^{e \times e}$ be an injective ring homomorphism and let b be a positive divisor of e such that $ek > rb$. Then the Knudsen-Preneel compression function $H = \mathsf{KP}^b([r, k, d]_{2^e})$ equals $H = \mathsf{BL}^b(\mathsf{C}^{\text{PRE}}, \mathsf{C}^{\text{POST}})$ with $\mathsf{C}^{\text{PRE}} = \bar{\varphi}(G^T)$ and $\mathsf{C}^{\text{POST}} = I_{rb}$.*

If $H = \mathsf{KP}^b([r, k, d]_{2^e})$, then $H_n : \{0,1\}^{kcn} \to \{0,1\}^{rn}$ with $c = e/b$ is defined for all n for which b divides n. Moreover, H_n is based on r PuRFs in $\text{Func}(cn, n)$. For use of H in an iterated hash function, note that per invocation (of H) one can compress $(ck - r)$ message blocks (hence the requirement $ek > rb$ ensures actually compression is taking place), and the rate of the compression function is $ck/r - 1$.

We will concentrate on the case $(b, e) \in \{(1, 2), (2, 4), (1, 3)\}$ and then in particular on the 16 parameter sets given by Knudsen and Preneel. (Since b is uniquely determined given e and c, we will often omit it.) Below is an example to help illustrate this formalism and Appendix C contains the generator matrices we used. (We only consider systematic generator matrices.)

Remark 5. The definition above of $\mathsf{KP}^b([r, k, d]_e)$ is actually slightly underdefined since the given parameters do not uniquely determine the matrix C^{PRE} (and consequently can lead to different compression functions). We list the three freedoms:

1. Choice of a linear code of the given parameters $[r, k, d]_{2^e}$.
2. Choice of a generator matrix G for the chosen linear code.
3. Choice of an injective homomorphism φ.

Our results—and those of Knudsen and Preneel—are largely unaffected by these three choices, with a possible exception for non-MDS codes, as we will see later. (On the other hand, the actual cost of

implementing the compression function in for instance the number of xor operations does depend on the choice of $\mathsf{C}^{\mathrm{PRE}}$.)

Example 6. Consider the compression function $H = \mathsf{KP}([5,3,3]_4)$. This builds a $6n \to 5n$ compression function using five underlying PuRFs each mapping $2n \to n$ (so the rate is $(2 \cdot 3 - 5)/5 = 1/5$). The preprocessing function C^{PRE} of H is defined by a generator matrix G of the code $[5,3,3]_4$. Using the G proposed in [13] and defining φ by (that is also the one given in [13])

$$\varphi(0) = \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}, \varphi(1) = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, \varphi(w) = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix} \text{ and } \varphi(w^2) = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}.$$

we get

$$G = \begin{pmatrix} 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & w \\ 0 & 0 & 1 & 1 & w^2 \end{pmatrix} \text{ and } \mathsf{C}^{\mathrm{PRE}} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{pmatrix}.$$

Therefore, given $W \in \{0,1\}^{6n}$, H_n computes the digest $Z \in \{0,1\}^{5n}$ as follows:

1. Compute $X \leftarrow (\mathsf{C}^{\mathrm{PRE}} \otimes I_n) \cdot W$;
2. Parse $X = (x_i)_{i=1...5}$ and for $i = 1...5$ compute $y_i = f_i(x_i)$;
3. Parse $(y_i)_{i=1...5} = Y$ and output $Z = (I_5 \otimes I_n) \cdot Y$, equivalently $Z = y_1||...||y_5$.

More precisely, for an input $W \in \{0,1\}^{6n}$ consisting of six n-bit blocks of the form $W = w_1 \| \ldots \| w_6$, the inputs to the underlying PuRFs are formed as follows:

$$\begin{gathered} x_1 = (w_1||w_2)\ ,\ x_2 = (w_3||w_4)\ ,\ x_3 = (w_5||w_6)\ , \\ x_4 = (w_1 \oplus w_3 \oplus w_5||w_2 \oplus w_4 \oplus w_6) \\ x_5 = (w_1 \oplus w_3 \oplus w_6||w_2 \oplus w_3 \oplus w_4 \oplus w_5 \oplus w_6). \end{gathered}$$

The rest is obvious as C^{POST} is simply the concatenation of the corresponding PuRF outputs y_i for all $i \in 1, \ldots, 5$.

Knudsen and Preneel's security claims. Knudsen and Preneel concentrate on the collision resistance of their compression function in the complexity theoretic model. Under a fairly generous (but plausible) assumption, they essentially[8] show that if $H = \mathsf{KP}^b([r,k,d]_{2^e})$, then finding collisions in H_n takes time at least $2^{(d-1)n/2}$. The intuition behind this result is fairly simple. The use of a code of minimum distance d implies that for any pair of differing compression function inputs $W \neq W'$ there are at least d different PuRF inputs. That is, if $(x_i)_{i=1..r}$ and $(x'_i)_{i=1..r}$ are the respective PuRF inputs, then there is an index set $\mathcal{I} \subseteq \{1, \ldots, r\}$ such that $|\mathcal{I}| \geq d$ and for all $i \in \mathcal{I}$ it holds that $x_i \neq x'_i$. Thus, for W and W' to collide, one needs to find collisions for the PuRFs f_i for all $i \in \mathcal{I}$ simultaneously. Also, as the dimension of the code is k, there exist k PuRFs that can be attacked independently, say $f_1, ..., f_k$. Now, this is where their assumption comes into play. Namely, finding a collision is assumed to take $2^{vn/2}$ time where v is the number of PuRFs f_j, $j \in \{k+1, \ldots, r\}$, whose inputs satisfy $x_j \neq x'_j$ once $W \neq W'$. From the Singleton bound, one has $r - k \geq d - 1$. Hence, $v \geq d - 1$.

For preimage resistance Knudsen and Preneel do not give a corresponding theorem and assumption, yet they do *conjecture* it to be essentially the square of the collision resistance, that is, they conjecture that finding a preimage will take at least time $2^{(d-1)n}$.

[8] Their actual theorem statements [13, Theorems 3 and 4] are phrased existentially.

Known attacks. To lowerbound the security of their construction, Knudsen and Preneel also present two attacks, one for finding preimages [13, Proposition 3] and one for finding collisions [13, Proposition 4]. We will only describe the preimage-finding attack in detail, since it is most relevant to our work—the collision-finding attack operates on the same principle.

Theorem 7 (Knudsen-Preneel attacks). *Let $H = \mathsf{KP}^b([r, k, d]_{2^e})$ be given and consider H_n (with b dividing n). Then*

1. *Preimages can be found in time $\max(2^{n(r-k)}, k2^{rn/k})$, using as many PuRF evaluations and requiring $ek2^{(r-k)n/k}$ n-bit blocks of memory;*
2. *Collisions can be found in time $\max(2^{n(r-k)/2}, k2^{(r+k)n/2k})$, using as many PuRF evaluations and requiring $ek2^{(r-k)n/2k}$ n-bit blocks of memory.*

Proof. (Sketch) For the preimage attack on H_n, given the target digest $Z \in \{0,1\}^{rn}$ (by construction $Z = y_1||\ldots||y_r$), the aim is to find $W \in \{0,1\}^{tn}$ such that $H(W) = Z$ holds. Note that (due to restricting to a systematic generator matrix for $\mathcal{C}$) the first k different PuRFs $f_1, \ldots, f_k$ have mutually independent inputs. This allows finding partial preimages for $y_1, \ldots, y_k$ independently. More precisely, for all $i \in \{1, \ldots, k\}$, $2^{(r-k)\frac{n}{k}+n}$ (arbitrary) queries are asked to each f_i and the ones whose answers hit to target partial image y_i are stored in the lists S_i. As each target value is expected to be hit $2^{(r-k)\frac{n}{k}}$ times, the lists contain about $2^{(r-k)\frac{n}{k}}$ partial preimages. This step can be performed in $k2^{(r-k)\frac{n}{k}+n}$ PuRF evaluations with a memory requirement of $ek2^{(r-k)n/k}$ n-bits.

Now, in the second step of the attack, all the lists are combined in all possible ways to query the remaining PuRFs $f_{k+1}, \ldots, f_r$. Notice that as $f_1, \ldots, f_k$ correspond to the systematic portion of the code, any tuple $(x_1, \ldots, x_k)$ of partial preimages *uniquely* defines a tuple of queries $(x_{k+1}, \ldots, x_r)$ to the remaining $r - k$ PuRFs. Following this remark (and the fact that the elements in k lists are independent), as there are roughly $2^{(r-k)\frac{n}{k}}$ elements in each list, the overall number of possible combinations is $2^{(r-k)n}$. This is exactly sufficient to expect a preimage for the remaining $(r - k)$ PuRFs, however to find it potentially requires $2^{(r-k)n}$ PuRF evaluations (and negligible additional memory).

For the collision attack, the same technique can be employed, but the number of possible combinations only need be $2^{(r-k)n/2}$ (in order to expect a collision for the remaining $(r - k)$ PuRFs). Consequently, in the first step of the attack, the attacker only need ask $2^{(r-k)\frac{n}{2k}+n} = 2^{(r+k)n/2k}$ queries to each f_i and after some bookkeeping the claim follows. □

4 Information-Theoretic Preimage Attack

Bounding the yield. In the information-theoretic setting, the yield (Definition 8) of an adversary captures the number of compression function evaluations the adversary can make given the queries made so far. The concept has proven very fruitful in attacking schemes [18] and proving security [26, 27].

Definition 8. *Let $H^{f_1,\ldots,f_r}$ be a compression function based on (ideal) primitives $f_1, \ldots, f_r$. The yield of an adversary after a set of queries to $f_1, \ldots, f_r$, is the number of inputs to H for which he can compute $H^{f_1,\ldots,f_r}$ given the answers to his queries. With $\mathrm{yield}(q)$ we denote the maximum expected yield given q queries to each of the oracles $f_1, \ldots, f_r$.*

For an arbitrary tn-to-sn bit compression function with r underlying cn-to-n primitives (each called once), the known lower bound on the yield [28, Theorem 6] is $\mathrm{yield}(q) \geq 2^{tn}(q/2^{cn})^r$. However, for the blockwise-linear schemes (Definition 1) it is possible to obtain a much bigger yield (being single-layer does not help either) and in particular it is *independent* of the number of primitive calls r.

Theorem 9. *Let $H = \mathsf{BL}^b(\mathsf{C}^{\mathrm{PRE}}, \mathsf{C}^{\mathrm{POST}})$ be a blockwise linear scheme with parameters c, t, s, r. Consider H_n with b dividing n. Then*

$$\mathrm{yield}(q) \geq 2^{\lfloor \frac{\lg q}{bc} \rfloor bt} \approx q^{t/c} .$$

Proof. Recall that t is the number of external n-bit input blocks and c the number of internal n-bit input blocks and that all n-bit blocks are subdivided into b n'-bit blocks. Set $n_q = \lfloor \lg q/(bc) \rfloor$ and $\mathcal{X} = (0^{n'-n_q} \times \{0,1\}^{n_q})^{bc}$. For each of the bc internal input subblocks, set the first $n' - n_q$ bits identical zero and let the rest range over all possibilities in $\{0,1\}^{n_q}$. All combinations of the internal input subblocks are combined (under concatenation) to give $(2^{n_q})^{bc} \leq q$ distinct inputs for any particular internal function. Query the f_i on these inputs (precisely corresponding to $\mathcal{X}$ defined above), for $i = 1, \ldots, r$.

Consider an external input W that consists of a concatenation of subblocks each with the first $n' - n_q$ bits set to zero. Then, due to linearity, $(\mathsf{C}^{\mathrm{PRE}} \otimes I_{n'}) \cdot W$ will map to a collection of PuRF-inputs all corresponding to the queries formed above, and hence this W will contribute to the yield. Since there are $2^{n_q bt}$ possible W that adhere to the format, we get the stated lower bound on the yield.

The approximation follows by ignoring the floor and simplifying the resulting expression. Although this can lead to slight inaccuracies, for increasing n there will be more and more values of q for which the expression is precise (so when $q = 2^{\alpha n}$ for rational α the expression is precise infinitely often). □

Ramifications for provable security. Intuitively, when the yield for a tn-to-sn compression function gets close to $2^{sn/2}$, a collision is expected (birthday bound) and once it surpasses 2^{sn} a collision is guaranteed (pigeon hole) and a preimage expected. The bounds for permutation-based compression functions by Rogaway and Steinberger [24] are based on formalizing this intuition. In the claims below we relate what the bound on the yield implies for preimage and collision resistance, assuming that the yield results in more or less uniform values. Note that the assumption certainly does not hold in general (hence 'presumably'), see also the discussion below.

Claim (Consequences for blockwise-linear schemes). Let $H = \mathsf{BL}^b(\mathsf{C}^{\mathrm{PRE}}, \mathsf{C}^{\mathrm{POST}})$ be a blockwise linear scheme with parameters c, t, s, r. Consider H_n with b dividing n.

1. If $q \geq 2^{scn/t}$ then $\mathrm{yield}(q) \geq 2^{sn}$ and a collision in H_n can be found with certainty; preimages can presumably be found with high probability.
2. If $q \geq 2^{scn/(2t)}$ then $\mathrm{yield}(q) \geq 2^{sn/2}$ and collisions in H_n can presumably be found with high probability.

Claim (Consequences for Knudsen-Preneel schemes). Let $H = \mathsf{KP}^b([r,k,d]_{2^e})$ be given. Consider H_n with b dividing n. Then preimages can be expected (and collisions guaranteed) after $2^{rn/k}$ queries and collisions can be expected after $2^{rn/(2k)}$ queries.

Exceptions. Stam [28] already showed that in some cases the intuitive yield-based bounds can be beaten by specific compression function design (achieving an arguably better efficiency-security tradeoff than otherwise possible). Similarly, it is not too hard to come up with designs that fit in the KP framework that beat the bounds from the claim. For instance, there is a $\mathsf{KP}^1([4,3,1]_4)$ code that results in digest $(f_1(x_1), f_2(x_2), f_3(x_3), f_4(x_1))$ yielding (IT-optimal) preimage resistance up to $2^{2n} > 2^{4n/3}$ queries. (Incidentally, this shows Knudsen and Preneel's preimage attack claim as summarized in Theorem 7 is stated too strongly as well). However, for most 'normal' compression functions counterintuitive behaviour is not to be expected (and if it does, might well be exploited by an attacker, e.g. to find collisions more efficiently); certainly no proposed blockwise-linear schemes was designed with defeating a yield-based argument in mind. Moreover, we believe that for *collision resistance* the claims might be correct for *all* (blockwise-linear) schemes.

Implications. A somewhat surprising implication of the claim for blockwise-linear schemes is that optimal security in the information theoretic sense does not seem possible when $t > c$ (neither for collision resistance nor for preimage resistance), *regardless* of the number of calls allowed to the underlying PuRF. This restriction also applies to the schemes considered by Peyrin et al. [21] and later analysed by Seurin and Peyrin [26]. While the former primarily consider real-life attacks, the latter also consider the information theoretic setting (mainly to prove security lower bounds). However, the attacks they considered were typically much weaker, given their emphasis on optimal security (so any attack will do).

5 Information-Theoretic Security Proof

The following result provides a security proof for preimage resistance of the Knudsen-Preneel compression functions in the information-theoretic model. That is, we give a lower bound on the query complexity (for a computationally unbounded adversary) of any preimage-finding attack. This bound shows that the query complexity of our new attack is optimal, up to a small factor. Therefore, the time complexity of our preimage attack is *optimal* whenever the time complexity of our attack matches its query complexity, and this is the case for 9 out of the 16 Knudsen-Preneel schemes.

Theorem 10 (Query-complexity epre-security bound). *Let $H = \mathsf{KP}^b([r,k,d]_{2^e})$ and, for b dividing n, consider H_n based on underlying PuRFs $f_i \in \mathrm{Func}(cn,n)$ for $i=1,\ldots,r$ with $c = e/b$. Then for $q \le 2^{cn}$ queries to each of the oracles and $\delta \ge 0$ an arbitrary real number:*

$$\mathsf{Adv}^{\mathsf{epre}}_{H}(q) \le \frac{q^{1+\delta}}{2^{(r-k)n}} + p$$

where $p = \mathsf{Pr}\left[B[kq;2^{-n}] > kq^{(1+\delta)/k}\right]$ and $B[kq;2^{-n}]$ denotes a random variable counting the number of successes in kq independent Bernoulli trials, each with success probability 2^{-n}.

Proof. Let $Z = z_1 \,||\, \cdots \,||\, z_r$ be the range point to be inverted. Recall that $f_1,\ldots,f_k$ are the functions corresponding to the systematic part of the $[r,k,d]_{2^e}$ code. Without loss of generality we will restrict our attention to an adversary $\mathcal{A}$ asking exactly q queries to each of its oracles. Consider the transcript of the oracle queries and responses. Necessarily, in this transcript there is at least one tuple $(x_1,\ldots,x_k)$ of queries to $f_1,\ldots,f_k$ such that for all $i=1,\ldots,k$ we have $f_i(x_i) = z_i$. Notice that because $f_1,\ldots,f_k$ correspond to the systematic portion of the code, any tuple $(x_1,\ldots,x_k)$ of queries to these k PuRFs *uniquely* defines a tuple of queries $(x_{k+1},\ldots,x_r)$ to the remaining $r-k$ PuRFs. Thus the number of tuples $(x_1,\ldots,x_k)$ in the transcript such $f_i(x_i) = z_i$ for all $i=1,\ldots,k$ determines the number of tuples $(x_{k+1},\ldots,x_r)$ that could possibly be a (simultaneous) preimage for $z_{k+1} \,||\, \cdots \,||\, z_r$. Intuitively, if this number is bounded to be sufficiently small, then the probability that $\mathcal{A}$ could have won the epre game will also be small. Let us make this formal.

For all $i=1,\ldots,k$, let L_i denote the list of partial preimages, that is the set of all x_i that $\mathcal{A}$ queried to f_i for which $f_i(x_i) = z_i$ and let $N_i = |L_i|$. Then N_i is the random variable denoting the number of partial preimages $\mathcal{A}$ found. Let bad be the event that $\sum_{i=1}^{k} N_i \ge kq^{(1+\delta)/k}$. Since each of the N_i are independent binomial random variables with success probability 2^{-n} and q trials each, we have that $\mathsf{Pr}[\mathsf{bad}] = \mathsf{Pr}\left[B[kq;2^{-n}] > kq^{(1+\delta)/k}\right] = p$.

Let $\mathcal{A}$ WINS be the event that $\mathcal{A}$ finds a preimage for the challenge Z; that is $\mathsf{Adv}^{\mathsf{epre}}_{H}(\mathcal{A}) = \mathsf{Pr}[\mathcal{A} \text{ WINS}]$. Then we have

$$\begin{aligned}
\mathsf{Pr}[\mathcal{A} \text{ WINS}] &\le \mathsf{Pr}[\mathcal{A} \text{ WINS} \mid \neg\mathsf{bad}]\,\mathsf{Pr}[\neg\mathsf{bad}] + \mathsf{Pr}[\mathcal{A} \text{ WINS} \mid \mathsf{bad}]\,\mathsf{Pr}[\mathsf{bad}] \\
&\le \mathsf{Pr}[\mathcal{A} \text{ WINS} \mid \neg\mathsf{bad}] + \mathsf{Pr}[\mathsf{bad}] \\
&\le \mathsf{Pr}[\mathcal{A} \text{ WINS} \mid \neg\mathsf{bad}] + p
\end{aligned}$$

Now, given that $\neg\mathsf{bad}$ holds, we know that $\sum_{i=1}^{k} N_i < kq^{(1+\delta)/k}$ and hence $\prod_{i=1}^{k} N_i < q^{(1+\delta)}$. Thus, no matter what queries are asked to $f_1,\ldots,f_k$ (and regardless of order), we know that there will be at most $q^{(1+\delta)}$ query tuples $(x_1,\ldots,x_k)$ that can be preimages of $z_1 \,||\, \cdots \,||\, z_k$. By our argument above, then, there will be at most this many tuples $(x_{k+1},\ldots,x_r)$ that can contribute to the occurrence of the event $\mathcal{A}$ WINS.

Consider the following experiment. Adversary $\mathcal{A}'$ is allowed to make at most $q^{(1+\delta)}$ queries to an oracle $\mathcal{O}^{f_{k+1},\ldots,f_r}$ that on input an $(r-k)$-tuple $(x_{k+1},\ldots,x_r)$ returns $(f_{k+1}(x_{k+1}),\ldots,f_r(x_r))$. $\mathcal{A}'$ wins if the oracle ever returns $(z_{k+1},\ldots,z_r)$. As the PuRFs $f_{k+1},\ldots,f_r$ are independent, it is easy to see that the probability that $\mathcal{A}'$ wins is at most $q^{(1+\delta)}/2^{(r-k)n}$. Thus in the epre experiment, the probability that $\mathcal{A}$'s queries to $f_{k+1},\ldots,f_r$ manage to produce a tuple $(x_{k+1},\ldots,x_r)$ that yields $z_{k+1},\ldots,z_r$, given that at most $q^{1+\delta}$ such tuples can be considered, is at most $q^{1+\delta}/2^{(r-k)n}$. So then

$$\mathsf{Pr}[\mathcal{A} \text{ WINS} \mid \neg\mathsf{bad}] \le \frac{q^{1+\delta}}{2^{(r-k)n}}\ ,$$

and our bound follows. □

The following corollary makes the theorem more concrete. By considering a parameter δ that provides a good balance between the first and second terms in the bound, it follows that $\Omega(2^{rn/k})$ queries are necessary to win the epre experiment. Since we already knew that $\mathcal{O}(2^{rn/k})$ queries are sufficient (under a reasonable uniformity assumption), this gives a complete characterization (up to increasingly small factors) of the query-complexity of finding preimages in Knudsen-Preneel construction.

Corollary 11. *Let $H = \mathsf{KP}^b([r,k,d]_e)$. Then asymptoticallly for n (with $b|n$) and $q \leq g(n)\left(\frac{2^n}{e}\right)^{r/k}$ with $g(n) = o(1)$:*

$$\mathsf{Adv}_H^{\mathsf{epre}}(q) = o(1)\ .$$

Proof. Set $\delta = \frac{r(k-1)-k^2}{r}$ and substitute $q = g(n)\left(\frac{2^n}{e}\right)^{r/k}$ in the statement of Theorem 10. We will show that asymptotically both terms vanish, starting with the first. After substitution, using $1+\delta = (rk-k^2)/r$, we get

$$2^{(k-r)n}\left(g(n)\left(\frac{2^n}{e}\right)^{(r/k)}\right)^{(rk-k^2)/r} = \left(\frac{1}{e}\right)^{(r-k)}(g(n))^{1+\delta}$$

which clearly vanishes whenever g does.

For the second term in the bound of Theorem 10 we need to bound the tail probability of a binomial distribution, for which we use Chernoff's bound applied to $\kappa = kq^{(1+\delta)/k} = kq^{(r-k)/r}$, so

$$p = \mathsf{Pr}\left[B[kq;2^{-n}] > \kappa\right] < (ekq/(\kappa 2^n))^\kappa = \left(\frac{e}{2^n}q^{k/r}\right)^\kappa$$

where we have left κ in the exponent for brevity. Substituting $q = g(n)\left(\frac{2^n}{e}\right)^{r/k}$ in the rightmost-side of the previous equation then leads to

$$p < \left(\frac{e}{2^n}\left(g(n)\left(\frac{2^n}{e}\right)^{r/k}\right)^{k/r}\right)^\kappa = (g(n))^{k\kappa/r}\ ,$$

where $k\kappa/r \geq (k^2)/r$, implying that p vanishes as well for $g(n) = o(1)$. □

6 Practical Preimage Attack against the MDS Schemes

Setting the stage. Section 4 contains a theoretical attack with a minimal number of queries. This already allows us to turn Knudsen-Preneel's preimage attack from an adaptive one into a non-adaptive one. In this section, we address reducing the time complexity.

Let $H = \mathsf{KP}^b([r,k,d]_{2^e})$ and n be given, where $b|n$ (and $bn' = n$ and $c = e/b$ as before). Consider a non-adaptive preimage-finding adversary $\mathcal{A}$ against H_n, trying to find a preimage for $Z \in \{0,1\}^{rn}$. For each $i = 1,\ldots,r$, $\mathcal{A}$ will commit to query lists $Q_i \subseteq V_i$ which, after querying, will result in a list of *partial* preimages $L_i = \{x_i \in Q_i | f_i(x_i) = z_i\}$. Since f_i is presumed random, we can safely assume that L_i is a set of approximately $|Q_i|/2^n$ randomly drawn elements of Q_i.

Finding a preimage then becomes equivalent to finding an element X in the range of C^{PRE} for which $x_i \in L_i$ for all $i = 1,\ldots,r$, or—exploiting the direct sum viewpoint—$X \in \sum_{i=1}^r L_i$. Due to the linearity of C^{PRE} at hand, the range-check itself is efficient for any given X, so a naive approach would be to simply exhaustively search $\sum_{i=1}^r L_i$. This would take time $|L|^r$.

An improvement can already be obtained by observing that, when a systematic matrix G is used to generate the code, any element $X_{1..k} \in \bigoplus_{i=1}^k V_i$ can uniquely (and efficiently) be extended to some X in the range of C^{PRE}. This lies at the heart of Knudsen and Preneel's adaptive attack and it can be adapted to the non-adaptive setting: for all $X_{1..k} \in \sum_{i=1}^k L_i$ compute its unique completion X and check whether for the remaining $i = k+1,\ldots,r$ the resulting $x_i \in L_i$. This reduces the time-complexity to $|L|^k$.

Organization. Still, we can do better. For concreteness, Section 6.1 provides a concrete warm-up example of our attack to the compression function $H = \mathsf{KP}([5,3,3]_4)$. Section 6.2 builds on this and contains the core idea of how to reduce the time complexity of this new non-adaptive attack, as well as its application against compression functions based on MDS codes. The slightly more complicated non-MDS case is discussed in Section 7.

6.1 Example: Preimages in $\mathsf{KP}([5,3,3])_4$ in $\mathcal{O}(2^{5n/3})$ Time

Before describing our preimage attack in its full generality, we present an example of it applied to the compression function $H = \mathsf{KP}([5,3,3]_4)$.

Claim. For the compression function $H = \mathsf{KP}([5,3,3]_4)$, preimages in H_n can be found in $\mathcal{O}(2^{5n/3})$ time with a memory requirement of $\mathcal{O}(2^{4n/3})$ n-bit blocks.

Proof. We refer the reader to our previous Example 6 for the details of G, φ and $\mathsf{C}^{\mathrm{PRE}}$. Let target digest $Z = z_1||\ldots||z_5$ be given. Our aim is to find the PuRF inputs $x_i = (x_i^1||x_i^2) \in \{0,1\}^{2n}$ such that $f_i(x_i) = z_i$ holds for all $i = 1,\ldots,5$, and $X = (\mathsf{C}^{\mathrm{PRE}} \otimes I_{n'}) \cdot W$ for some compression function input W (where X is comprised of the five x_i). In this case W is a preimage for Z. An abstract outline of the attack is given in Figure 2, we proceed with the details.

The attack starts with what we call the QUERY PHASE. Namely, for each $i = 1,\ldots,5$ and for all $x_i^1, x_i^2 \in 0^{n/6} \times \{0,1\}^{5n/6}$, we query $f_i(x_i)$ and keep a list L_i of pairs that hit the target digest z_i. As a result, a total of $2^{5n/3}$ queries are made (in $2^{5n/3}$ time) per PuRF, resulting in $|L_i| \approx 2^{5n/3}/2^n = 2^{2n/3}$ (as each query has probability 2^{-n} to hit its target).

Since any tuple $(x_1,x_2,x_3) \in L_1 \times \ldots \times L_3$ uniquely determines a preimage candidate W, finding a preimage is equivalent to finding an element $X \in L_1 \times \ldots \times L_5$ in the range of C^{PRE}. (The task is actually to determine whether a random vector y is a valid codeword; this can easily be detected by checking $y \cdot H^T = 0$ where H is the parity check matrix of the underlying code C.) To do this efficiently, we will first identify the tuples (x_1,x_2,x_3,x_4) in the lists that can be complemented (not necessarily using $x_5 \in L_5$) to an element in the range. From the generator matrix G it can be seen that this complementation is possible iff $x_1 \oplus x_2 \oplus x_3 \oplus x_4 = 0$. So let us define

$$L_{\{1,2,3,4\}} = \{(x_1,x_2,x_3,x_4) \in L_1 \times L_2 \times L_3 \times L_4 \mid x_1 \oplus x_2 \oplus x_3 \oplus x_4 = 0\} \ .$$

We can construct $L_{\{1,2,3,4\}}$ efficiently using a standard technique related to the generalized birthday problem. It starts with the MERGE PHASE, where we create the lists $\tilde{L}_{\{1,2\}}$ and $\tilde{L}_{\{3,4\}}$ defined by

$$\tilde{L}_{\{1,2\}} = \{((x_1,x_2), x_1 \oplus x_2) \mid (x_1,x_2) \in L_1 \times L_2\}\,,$$
$$\tilde{L}_{\{3,4\}} = \{((x_3,x_4), x_3 \oplus x_4) \mid (x_3,x_4) \in L_3 \times L_4\}$$

both sorted on their second component. In the JOIN PHASE we look for the collisions in their second components. Since $|L_i| \approx 2^{2n/3}$, creating either $\tilde{L}$ takes about $\mathcal{O}(n2^{4n/3})$ time and $\mathcal{O}(2^{4n/3})$ memory. (In general, the smallest $\tilde{L}$ is sorted and stored and the other is used for collision check.) Since $\tilde{L}_{\{1,2\}}$ and $\tilde{L}_{\{3,4\}}$ both have roughly $2^{4n/3}$ elements and they need to collide on $2n$ bits, of which $n/3$ bits are set to zero, the expected number of collisions is about $(2^{4n/3})^2/2^{(2-1/3)n} = 2^n = |L_{\{1,2,3,4\}}|$. (See Figure 2.)

We now have the collision list $L_{\{1,2,3,4\}}$ and all that needs to be done is to check, for each of its elements, whether the corresponding $x_5 \in L_i$. If this is the case, then (x_1,x_2,x_3) produces a valid preimage. This final phase we call the FINALIZATION phase. It is clear that it cannot take much longer than it took to create $L_{\{1,2,3,4\}}$. Moreover, the expected number of preimages output is 1. Note that $|L_{\{1,2,3,4\}}| \approx 2^n$ and $|L_5| \approx 2^{2n/3}$. Again, we need to check the correspondence on $2n$ bits, of which $n/3$ are set to zero. Hence, we do expect to find $2^{(1+2/3)n}/2^{(2-1/3)n} = 1$ preimage.

Picking up the stepwise time and memory complexities gives the desired result. □

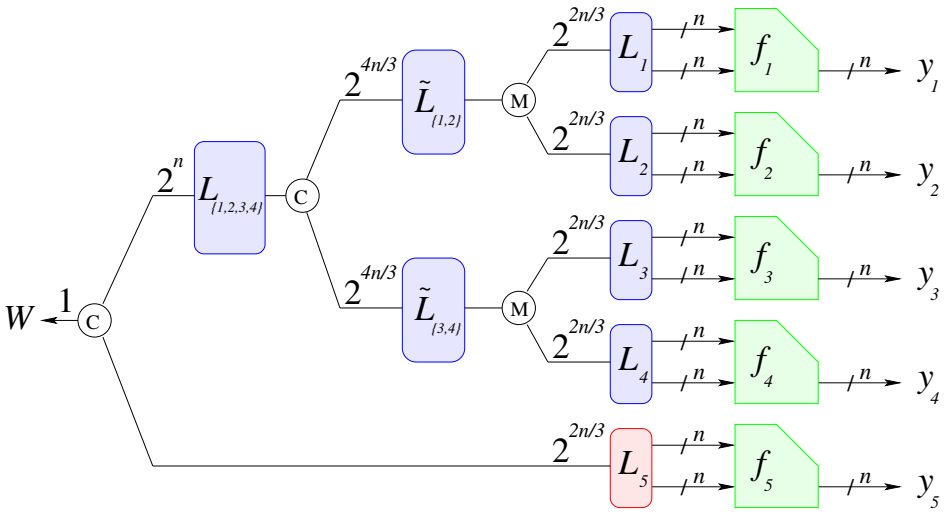

Fig. 2. Our preimage attack on the Knudsen-Preneel compression function $H = \mathsf{KP}([5,3,3]_4)$ illustrated. The (unlabelled) inputs to $f_1, \ldots, f_5$ correspond to $(x_1^1, x_1^2), \ldots, (x_5^1, x_5^2)$. Here, M denotes the merging of two lists whereas C denotes the collision search between two lists. The size of the lists is given by labelling the inputs to the M and C operations.

6.2 Generic Attack against MDS Schemes

Our attack on the compression function $\mathsf{KP}([5,3,3]_4)$ can be generalized to other Knudsen-Preneel compression functions. Note that the attack above consists of four steps:

1. QUERY PHASE to generate the lists of partial preimages;
2. MERGE PHASE where two sets of lists are each merged exhaustively;
3. JOIN PHASE where collisions between the two merged lists are selected resulting in fewer partial preimages that however are preimage of a larger part of the target digest.
4. FINALIZATION where the remaining partial preimages are filtered for being a full preimage.

Since all Knudsen-Preneel compression functions are blockwise-linear, we will always be able to run a non-adaptive attack. The question is whether we will always be able to *efficiently* find a full preimage given the lists of partial preimages L_i. For the $[5,3,3]_4$ example we could significantly reduce our workload because we found an easy-to-verify relation that the inputs to the PuRFs, thus in particular the elements in L_i, had to satisfy. The question is whether we were lucky to find this relation, or whether there is a deeper underlying reason for it. Additionally, it is not immediate that, even if we can somehow efficiently merge and join, there is an efficient way to *finalize*. We will proceed to describe the reason why the attack worked, and show that this naturally leads to a generalization to all Knudsen-Preneel schemes based on MDS codes. In Appendix D we describe some further variants to reduce memory.

The core observation. From a high level, our approach is simple: we first identify an index set $\mathcal{I} \subseteq \{1, \ldots, r\}$ defining a subspace $\bigoplus_{i \in \mathcal{I}} V_i$ for which the range of C^{PRE} (when restricted to this subspace), is not surjective. By (blockwise-linear) construction, C^{PRE} will then map to a subspace of $\bigoplus_{i \in \mathcal{I}} V_i$ of at most dimension $(|\mathcal{I}| - 1)cn$ (over $\mathbb{F}_2$). As a consequence, we will be able to prune significantly the total collection of candidate preimages in $\sum_{i \in \mathcal{I}} L_i$, keeping only those elements that are possibly in the range of C^{PRE} restricted to $\bigoplus_{i \in \mathcal{I}} V_i$. In the following, we will show how to *efficiently* find an index set $\mathcal{I}$, and how to *efficiently* prune.

It turns out that an important parameter determining the runtime of our preimage attack is $d^\perp$, the minimum distance of the dual code. Let χ be the function that maps $h \in \mathbb{F}_{2^e}^r$ to the set of indices of non-zero entries in h. Thus, $\chi(h) \subseteq \{1, \ldots, r\}$ and $|\chi(h)|$ equals the Hamming weight of the codeword. If $h \in \mathcal{C}^\perp$, then for $\mathcal{I} = \chi(h)$ we have precisely the property that allows us to prune $\sum_{i \in \mathcal{I}} L_i$ for partial preimages. The following proposition develops the key result for understanding our attack and the role the dual code plays in it. The interpretation follows the proposition.

Proposition 12. *Let $H = \mathsf{KP}^b([r,k,d]_{2^e})$ and $M \in \mathbb{F}_2^{e\times re/b}$ be given. Suppose that $M = \bar{\varphi}(h^T)$ for some $h \in \mathbb{F}_{2^e}^r$, then for all positive integers n' it holds that $(M \otimes I_{n'}) \cdot (\mathsf{C}^{\mathrm{PRE}} \otimes I_{n'}) \cdot W = 0$ for all $W \in \{0,1\}^{ken'}$ iff $h \in \mathcal{C}^\perp$.*

Proof. Let $h \in \mathbb{F}_{2^e}^r$ and $W \in \{0,1\}^{ken'}$ be given. Let $M = \bar{\varphi}(h^T)$ and recall that $\mathsf{C}^{\mathrm{PRE}} = \bar{\varphi}(G^T)$ where G is a generator of $\mathcal{C}$. Then

$$\begin{aligned}(M \otimes I_{n'}) \cdot (\mathsf{C}^{\mathrm{PRE}} \otimes I_{n'}) \cdot W &= (\bar{\varphi}(h^T) \otimes I_{n'}) \cdot (\bar{\varphi}(G^T) \otimes I_{n'}) \cdot W \\ &= ((\bar{\varphi}(h^T) \cdot \bar{\varphi}(G^T)) \otimes I_{n'}) \cdot W \\ &= (\bar{\varphi}((Gh)^T) \otimes I_{n'}) \cdot W\end{aligned}$$

The statement that $(\bar{\varphi}((Gh)^T) \otimes I_{n'}) \cdot W = 0$ for all $W \in \{0,1\}^{ken'}$ is equivalent to the statement that $\bar{\varphi}((Gh)^T) = 0$. Since φ is injective, this in turn is equivalent to $(Gh)^T = 0$. By definition, it holds that $Gh = 0$ iff $h \in \mathcal{C}^\perp$. □

In essence, this proposition tells us that if we are given a codeword $h \in \mathcal{C}^\perp$ and an element $X \in \mathbb{F}_2^{rcn}$ (to be input to the PuRFs), then X can only be in the range of C^{PRE} if $(\bar{\varphi}(h^T) \otimes I_{n'}) \cdot X = 0$. Since the only parts of X relevant for this check are those lining up with the nonzero entries of h, we get that $\mathcal{I} = \chi(h)$ is the droid we are looking for. Indeed, an element $X \in \sum_{i\in\chi(h)} L_i$ can be completed to an element in the range of C^{PRE} iff $(\bar{\varphi}(h) \otimes I_{n'}) \cdot (X + 0) = 0$ (where we write $X + 0$ for embedding into the larger $\oplus_{i=1}^r V_i$).

Efficient creation of

$$L_h = \left\{ X \in \sum_{i\in\chi(h)} L_i \mid (\bar{\varphi}(h) \otimes I_{n'}) \cdot (X + 0) = 0 \right\}$$

is done adapting standard techniques [4, 30, 25, 3] by splitting the codeword in two and looking for all collisions. Suppose that $h = h_0 + h_1$ with $\chi(h_0) \cap \chi(h_1) = \emptyset$, and define, for $j = 0, 1$

$$\tilde{L}_{h_j} = \left\{ (X_j, (\bar{\varphi}(h_j) \otimes I_{n'}) \cdot (X_j + 0)) \mid X_j \in \sum_{i\in\chi(h_j)} L_i \right\} .$$

Then L_h consists of the elements $X_0 + X_1$ for which $(X_0, Y_0) \in \tilde{L}_{h_0}, (X_1, Y_1) \in \tilde{L}_{h_1}$, and $Y_0 = Y_1$. By sorting the two $\tilde{L}$'s the complexity of creating L_h is then roughly the maximum cardinality of the three sets $\tilde{L}_{h_0}, \tilde{L}_{h_1}$, and L_h involved. It therefore clearly pays dividends to minimize the Hamming weights of h_0 and h_1, which is done by picking a codeword $h \in \mathcal{C}^\perp$ of minimum distance $d^\perp$ and splitting it (almost) evenly.

For an MDS code, we know that $d^\perp = k + 1$. As a result, if h attains this, the map C^{PRE} is injective when restricted to $\bigoplus_{i\in\chi(h)} V_i$ (or else the minimum distance would be violated). Hence, we know that all possible preimages given *all* the lists L_i are represented by the partial preimages contained in $\tilde{L}_h$. We can finalize by simply checking for all elements in $\tilde{L}_h$ whether its unique completion to $X \in \bigoplus_{i=1}^r V_i$ corresponds to $x_i \in L_i$ for all $i = 1, \ldots, r$ (where checking for $i \in \chi(h)$ can be omitted). The complete preimage-finding algorithm is given in Algorithm 14.

Remark 13. Note that the memory requirements of the preimage-finding algorithm given in Algorithm 14 can be further reduced using the techniques introduced in [4, 30]. We investigate this, i.e. a more space efficient variant of our attack (without violating the time complexity), in Appendix D.

Reinterpreting the example. Let us revisit our preimage attack example on $H = \mathsf{KP}([5,3,3]_4)$ to see how it fits within the general framework. In the example we more or less magically came up with the relation $x_1 \oplus x_2 \oplus x_3 \oplus x_4 = 0$. We can now appreciate that this constraint is really imposed by the dual

Algorithm 14 (Preimage attack against MDS-based schemes).

Input: $H = \mathsf{KP}^b([r,k,d]_{2^e})$, block size n with $b|n$ and target digest $Z \in \{0,1\}^{rn}$.

Output: A preimage $W \in \{0,1\}^{tn}$ such that $H_n(W) = Z$.

1. QUERY PHASE. Define
$$\mathcal{X} = (\{0\}^{\frac{n}{b} - \frac{rn}{ek}} \times \{0,1\}^{\frac{rn}{ek}})^e$$
and, for $i = 1, \ldots, r$ let $Q_i = \mathcal{X} \subset V_i$. Query f_i on all $x_i \in Q_i$. Keep a list L_i of all partial preimages $x_i \in Q_i$ satisfying $f_i(x_i) = y_i$.
2. FIRST MERGE PHASE. Find a nonzero codeword $h \in \mathcal{C}^\perp$ of minimum Hamming weight $d^\perp$. Let $h = h_0 + h_1$ with $\chi(h_0) \cap \chi(h_1) = \emptyset$ and of Hamming weights $\lfloor d^\perp/2 \rfloor$ and $\lceil d^\perp/2 \rceil$ respectively. Create, for $j = 0, 1$
$$\tilde{L}_{h_j} = \left\{ (X_j, (\bar{\varphi}(h_j) \otimes I_{n'}) \cdot (X_j + 0)) \mid X_j \in \sum_{i \in \chi(h_j)} L_i \right\}$$
both sorted on their second component.
3. FIRST JOIN PHASE. Create L_h consisting exactly of those elements $X_0 + X_1$ for which $(X_0, Y_0) \in \tilde{L}_{h_0}, (X_1, Y_1) \in \tilde{L}_{h_1}$, and $Y_0 = Y_1$.
4. FINALIZATION. For all $X \in L_h$ create the unique W corresponding to it and check whether it results in $x_i \in L_i$ for all $i = 1, \ldots, r$. If so, output W.

codeword $h = (1\ 1\ 1\ 1\ 0)$. Thus our example corresponds to Algorithm 14 with $\chi(h_0) = \{1,2\}$ and $\chi(h_1) = \{3,4\}$ (leading to a completely even division).

Note that one can also perform the attack based on other dual codewords of minimum distance, for instance $h = \left(1\ w\ w^2\ 0\ 1\right)$. These two minimum distance dual codewords can easily be found (in particular for MDS codes) based on the given systematic generator matrix $G = [I_k|P]$ of the original code. Namely, the dual codeword in the $(j-k)^{th}$ row of the corresponding generator matrix of the dual code $G^\perp = [P^T|I_{r-k}]$ is used as h for $j > k$. (In general finding a minimum distance codeword might be more involved, but the dimensions are sufficiently small to allow exhaustive search.)

Analysis of the preimage attack. We proceed with the analysis of the generic preimage attack by providing the justifications of our claims and the overall time and memory complexities. We initially maintain $d^\perp$ in the expressions for future use (when discussing non-MDS codes). The proof of Theorem 15 (together with that of Theorem 17) is given in Appendix A.

Theorem 15. *Let $H = \mathsf{KP}^b([r,k,d]_{2^e})$ be given and let $d^\perp$ be the minimum distance of the dual code of $\mathcal{C}$. Suppose $\mathcal{C}$ is MDS and consider the preimage attack described in Algorithm 14 run against H_n using $q = 2^{rn/k}$ queries ($|Q_i| = 2^{rn/k}$). Then the expected number of preimages output equals one and the expectations for the internal list sizes are:*

$$|L_i| = 2^{\frac{(r-k)n}{k}}\ ,\ |L_h| = 2^{\frac{(d^\perp(r-k)-r)n}{k}}\ ,\ |\tilde{L}_{h_0}| = 2^{\lfloor \frac{d^\perp}{2} \rfloor \frac{(r-k)n}{k}}\ ,\ |\tilde{L}_{h_1}| = 2^{\lceil \frac{d^\perp}{2} \rceil \frac{(r-k)n}{k}}\ .$$

The average case time and memory complexity (expressed in the number cn-bit blocks) of the algorithm is $\mathcal{O}(2^{\alpha n})$ and $\mathcal{O}(2^{\beta n})$ respectively where (substituting $d^\perp = k+1$)

$$\alpha = \max\left(\frac{r}{k}, \lceil \frac{k+1}{2} \rceil \left(\frac{r-k}{k}\right), r-k-1\right)\ ,\quad \beta = \lfloor \frac{k+1}{2} \rfloor \left(\frac{r-k}{k}\right)$$

which for $d = 3$ simplifies to $\alpha = \frac{r}{k} = 1 + \frac{2}{k}$ and $\beta \leq \frac{k+1}{k}$.

In our attack, we set the number of queries as suggested by a yield-based bound. Hence, as long as this first querying phase is dominating, we know that our attack is optimal, as in the case for example against $\mathsf{KP}([5,3,3]_4)$. When the querying phase is not dominating (indicated by a gap between the time complexities of our attack and the lower bounds given in Tables 1 and 2) further improvements might be possible.

Tables 3 and 4 contain an overview for the various cardinalities and complexities for all the compression functions based on MDS-codes that were suggested by Knudsen and Preneel [13].

Table 3. Preimage attacks on the Knudsen-Preneel compression functions based on $2n \to n$ PuRFs and MDS-codes.

Code	$d^\perp$	Query Complexity	Time Complexity of Our Attack (List Sizes)				Overall Time	Overall Memory
$[r,k,d]_{2^e}$		$\lvert Q_i\rvert$	$\lvert L_i\rvert$	$\lvert\tilde{L}_{h_0}\rvert$	$\lvert\tilde{L}_{h_1}\rvert$	$\lvert L_h\rvert$	Thm. 15	
$[5,3,3]_4$	4	$2^{5n/3}$	$2^{2n/3}$	$2^{4n/3}$	$2^{4n/3}$	2^n	$2^{5n/3}$	$2^{4n/3}$
$[6,4,3]_{16}$	5	$2^{3n/2}$	$2^{n/2}$	2^n	$2^{3n/2}$	2^n	$2^{3n/2}$	2^n
$[8,6,3]_{16}$	7	$2^{4n/3}$	$2^{n/3}$	2^n	$2^{4n/3}$	2^n	$2^{4n/3}$	2^n
$[12,10,3]_{16}$	11	$2^{6n/5}$	$2^{n/5}$	2^n	$2^{6n/5}$	2^n	$2^{6n/5}$	2^n
$[9,6,4]_{16}$	7	$2^{3n/2}$	$2^{n/2}$	$2^{3n/2}$	2^{2n}	2^{2n}	2^{2n}	$2^{3n/2}$
$[16,13,4]_{16}$	14	$2^{16n/13}$	$2^{3n/13}$	$2^{21n/13}$	$2^{21n/13}$	2^{2n}	2^{2n}	$2^{21n/13}$

Table 4. Preimage attacks on the Knudsen-Preneel compression functions based on $3n \to n$ PuRFs and MDS codes.

Code	$d^\perp$	Query Complexity	Time Complexity of Our Attack (List Sizes)				Overall Time	Overall Memory
$[r,k,d]_{2^e}$		$\lvert Q_i\rvert$	$\lvert L_i\rvert$	$\lvert\tilde{L}_{h_0}\rvert$	$\lvert\tilde{L}_{h_1}\rvert$	$\lvert L_h\rvert$	Thm. 15	
$[4,2,3]_8$	3	2^{2n}	2^n	2^n	2^{2n}	2^n	2^{2n}	2^n
$[6,4,3]_8$	5	$2^{3n/2}$	$2^{n/2}$	2^n	$2^{3n/2}$	2^n	$2^{3n/2}$	2^n
$[9,7,3]_8$	8	$2^{9n/7}$	$2^{2n/7}$	$2^{8n/7}$	$2^{8n/7}$	2^n	$2^{9n/7}$	$2^{8n/7}$
$[5,2,4]_8$	3	$2^{5n/2}$	$2^{3n/2}$	$2^{3n/2}$	2^{3n}	2^{2n}	2^{3n}	$2^{3n/2}$
$[7,4,4]_8$	5	$2^{7n/4}$	$2^{3n/4}$	$2^{3n/2}$	$2^{9n/4}$	2^{2n}	$2^{9n/4}$	$2^{3n/2}$
$[10,7,4]_8$	8	$2^{10n/7}$	$2^{3n/7}$	$2^{12n/7}$	$2^{12n/7}$	2^n	2^{2n}	$2^{12n/7}$

7 Extending the Attack to Non-MDS Constructions

For non-MDS codes we can try to mount the preimage attack given by Algorithm 14, but in the FINALIZATION we encounter a problem. Since $d^\perp < k+1$ for non-MDS codes, the map C^{PRE} restricted to $\oplus_{i\in\chi(h)}V_i$ is no longer injective and we can no longer reconstruct a unique W corresponding to some $X \in L_h$. There are two possible fixes to this problem. One is to simply merge as yet unused lists L_i into L_h until reconstruction does become unique. We will refer to this as Algorithm $14'$.

However, a more efficient approach is to perform a second stage of merging and joining. In Algorithm 16 we simply paste in extra MERGE and JOIN phases in order to maintain the low complexity. We have only included one extra merge-join phase for non-MDS codes. For the parameters proposed by Knudsen and Preneel, this will always suffice. For other parameters possibly extra merge-join phases are required before full rank is achieved, we did not investigate this.

Table 5. Preimage attacks on the KP compression functions based on $2n \to n$ PuRFs and *non*-MDS-codes.

Code		Cardinalities related to our attack						Overall		Alg. $14'$
$[r,k,d]_{2^e}$	$d^\perp$	$\lvert Q_i\rvert$	$\lvert\tilde{L}_{h_0}\rvert$	$\lvert\tilde{L}_{h_1}\rvert$	$\lvert L_h\rvert$	$\max\left(\lvert\tilde{L}_{h'_0}\rvert,\lvert\tilde{L}_{h'_1}\rvert\right)$	$\lvert L_{h'}\rvert$	Time	Memory	Time
$[8,5,3]_4$	4	$2^{8n/5}$	$2^{6n/5}$	$2^{6n/5}$	$2^{4n/5}$	$2^{7n/5}$	2^n	$2^{8n/5}$	$2^{6n/5}$	2^{2n}
$[12,9,3]_4$	7	$2^{4n/3}$	2^n	$2^{4n/3}$	2^n	$2^{4n/3}$	2^n	$2^{4n/3}$	2^n	2^{2n}
$[9,5,4]_4$	4	$2^{9n/5}$	$2^{8n/5}$	$2^{8n/5}$	$2^{7n/5}$	$2^{11n/5}$	2^{2n}	$2^{11n/5}$	$2^{8n/5}$	2^{3n}
$[16,12,4]_4$	11	$2^{4n/3}$	$2^{5n/3}$	2^{2n}	$2^{7n/3}$	$2^{7n/3}$	2^{2n}	$2^{7n/3}$	$2^{5n/3}$	2^{3n}

Analysis of the attack. Although the addition of one extra round of MERGEing and JOINing sounds relatively simple, the analysis of it is slightly tedious, mainly because the first joining creates some asymmetry between the lists (that was not present before). We note that in Theorem 17 below the value

Algorithm 16 (Preimage attack against non-MDS-based schemes).

Input: $H = \mathsf{KP}^b([r,k,d]_{2^e})$, block size n with $b|n$ and target digest $Z \in \{0,1\}^{rn}$.

Output: A preimage $W \in \{0,1\}^{tn}$ such that $H_n(W) = Z$.

1. QUERY PHASE. As in Algorithm 14.
2. FIRST MERGE PHASE. As in Algorithm 14.
3. FIRST JOIN PHASE. As in Algorithm 14.
4. SECOND MERGE PHASE. Find a codeword $h' \in \mathcal{C}^\perp \backslash \mathbb{F}_{2^e} h$ of minimum Hamming weight (possibly exceeding $d^\perp$). Let $h' = h'_0 + h'_1$ with $\chi(h'_0) \cap \chi(h'_1) = \emptyset$, $\chi(h'_1) \cap \chi(h) = \emptyset$, and of Hamming weights yet to be determined. Create
$$\tilde{L}_{h'_0} = \left\{ \big(X_0, (\bar{\varphi}(h'_0) \otimes I_{n'}) \cdot (X_0 + 0)\big) \mid X_0 \in L_h + \sum_{i \in \chi(h'_0) \backslash \chi(h)} L_i \right\}$$
$$\tilde{L}_{h'_1} = \left\{ \big(X_1, (\bar{\varphi}(h'_1) \otimes I_{n'}) \cdot (X_1 + 0)\big) \mid X_1 \in \sum_{i \in \chi(h'_1)} L_i \right\} .$$
5. SECOND JOIN PHASE. Create $L_{h'}$ consisting exactly of those elements $X_0 + X_1$ for which $(X_0, Y_0) \in \tilde{L}_{h'_0}, (X_1, Y_1) \in \tilde{L}_{h'_1}$, and $Y_0 = Y_1$.
6. FINALIZATION. For all $X \in L_{h'}$ create the unique W corresponding to it and check whether it results in $x_i \in L_i$ for all $i = 1, \ldots, r$. If so, output W.

i for which T_1 attains its minimum really only has a choice of two, but its algebraic optimization would not ease readability and obscure the underlying meaning. Note that for the memory analysis, we use a modified version of the algorithm, that is memory-optimized without aversely affecting the running time.

Because it is less clear from the theorem what the actual cardinalities will end up being (and consequently which step will be dominating), Table 5 summarizes the relevant quantities for the four non-MDS compression functions $\mathsf{KP}([r,k,d]_4)$ suggested by Knudsen and Preneel [13]. Only for the $[9,5,4]_4$ code the second stage dominates the overall runtime.

Theorem 17. *Let $H = \mathsf{KP}^b([r,k,d]_{2^e})$ be given and let $d^\perp$ be the minimum distance of the dual code of $\mathcal{C}$. Consider the preimage attack described in Algorithm 16 run against H_n using $q = 2^{rn/k}$ queries ($|Q_i| = 2^{rn/k}$). Then, the expected number of preimages output equals one and the expectations for the internal list sizes are for the first merge-join are as before (see Theorem 15) and for the second merge-join phase*

$$\max(|\tilde{L}_{h'_0}|, |\tilde{L}_{h'_1}|) \leq 2^{T_1 n} \ , \ \min(|\tilde{L}_{h'_0}|, |\tilde{L}_{h'_1}|) \leq 2^{T_2 n} \ , \ |L_{h'}| \leq 2^{(r-k-2)n}$$

where

$$T_1 = \min_{i \in \{0,\ldots,k-d^\perp+1\}} \left(\max\{(k - d^\perp + 2 - i)\frac{r-k}{k}, i\frac{r-k}{k} + \frac{d^\perp(r-k) - r}{k}\} \right)$$

and $T_2 = \min\{(k - d^\perp + 2 - i_0)\frac{r-k}{k}, i_0\frac{r-k}{k} + \frac{d^\perp(r-k)-r}{k}\}$ and $i_0 \in \{0, \ldots, k - d^\perp + 1\}$ is the value for which T_1 attains the minimum. The expected time complexity of the algorithm is a small constant multiple of

$$\max\left(q, |\tilde{L}_{h_1}|, |L_h|, |\tilde{L}_{h'_0}|, |\tilde{L}_{h'_1}|, |L_{h'}|\right)$$

requiring expected memory around $\max\left(|\tilde{L}_{h_0}|, \min(|\tilde{L}_{h'_0}|, |\tilde{L}_{h'_1}|)\right)$ (expressed in the number cn-bit blocks).

Choice of code. Our attacks against the four non-MDS codes were based on the generator matrix given by Magma's BKLC routine. It is conceivable that different, non-equivalent codes perform differently

under our attack. Most importantly, they might not have the same $d^\perp$ which will certainly change some of the cardinalities involved in our attack. Although this does not automatically means the attack becomes faster or slower, it is certainly a possibility. We note that there is a trivial bound $d^\perp \leq k$ (or else the code would be MDS), but in none of the four cases we achieved this bound. Stronger bounds on $d^\perp$ might be possible by extending the recently developed primal-dual distance bounds [16] to the $\mathbb{F}_4$ setting.

Additionally, even for non-equivalent codes with identical dual distance, slight variations in the runtime are conceivable. In the analysis upon which Theorem 17 is based, we conservatively assume that in the second merge all remaining lists take part in the merge. However, if the second case dominates, a second codeword of minimum weight or a low-weight codeword with significant overlap with the first, might reduce the listed complexity.

8 Conclusion

In this paper, we provide a new security analysis of the KP construction by directly addressing its conjectured preimage-resistance security. Firstly, we describe an attack taking into account both query and time-complexities. Our attacks demonstrate that the conjectured lower bound by Knudsen and Preneel is incorrect and exemplify that security bounds derived from the number of active functions can be misleading. Secondly, we determine a lower bound on the query complexity for a computationally unbounded adversary to successfully find preimages. This shows that the query complexity of our new attack is essentially optimal (up to a small factor). Moreover, we can conclude that the time complexity of our new preimage-finding attack is optimal for 9 out of the 16 schemes. For the remaining seven schemes we leave a gap between the information-theoretic lower bound and the real-life upper bound.

Acknowledgement. We thank the anonymous referees for their comments, in particular pointing out the work of Watanabe [31].

References

1. J. Black, P. Rogaway, and T. Shrimpton. Black-box analysis of the block-cipher-based hash-function constructions from PGV. In Yung [32], pages 320–335.
2. G. Brassard, editor. *Advances in Cryptography—Crypto'89*, volume 435 of *LNCS*. Springer, Heidelberg, 1990.
3. P. Camion and J. Patarin. The knapsack hash function proposed at crypto'89 can be broken. In D. W. Davies, editor, *Advances in Cryptography—Eurocrypt'91*, volume 547 of *LNCS*, pages 39–53. Springer, Heidelberg, 1991.
4. P. Chose, A. Joux, and M. Mitton. Fast correlation attacks: An algorithmic point of view. In L. Knudsen, editor, *Advances in Cryptography—Eurocrypt'02*, volume 2332 of *LNCS*, pages 209–221. Springer, Heidelberg, 2002.
5. I. Damgård. A design principle for hash functions. In Brassard [2], pages 416–427.
6. O. Dunkelman, editor. *Fast Software Encryption (FSE'09)*, volume 5665 of *LNCS*. Springer, Heidelberg, 2009.
7. E. Fleischmann, M. Gorski, and S. Lucks. On the security of Tandem-DM. In Dunkelman [6], pages 84–103.
8. E. Fleischmann, M. Gorski, and S. Lucks. Security of cyclic double block length hash functions. In Parker [20], pages 153–175.
9. D. Giry. BlueKrypt - v 24.6, Cryptographic Key length Recommendation, 2010. Available at `www.keylength.com`.
10. L. Knudsen and F. Muller. Some attacks against a double length hash proposal. In B. K. Roy, editor, *Advances in Cryptography—Asiacrypt'05*, volume 3788 of *LNCS*, pages 462–473. Springer, Heidelberg, 2005.
11. L. R. Knudsen and B. Preneel. Hash functions based on block ciphers and quaternary codes. In K. Kim and T. Matsumoto, editors, *Advances in Cryptography—Asiacrypt'96*, volume 1163 of *LNCS*, pages 77–90. Springer, Heidelberg, 1996.
12. L. R. Knudsen and B. Preneel. Fast and secure hashing based on codes. In J. Burt Kaliski and S. Burton, editors, *Advances in Cryptography—Crypto'97*, volume 1294 of *LNCS*, pages 485–498. Springer, Heidelberg, 1997.
13. L. R. Knudsen and B. Preneel. Construction of secure and fast hash functions using nonbinary error-correcting codes. *IEEE Transactions on Information Theory*, 48(9):2524–2539, 2002.
14. X. Lai and J. L. Massey. Hash function based on block ciphers. In R. A. Rueppel, editor, *Advances in Cryptography—Eurocrypt'92*, volume 658 of *LNCS*, pages 55–70. Springer, Heidelberg, 1992.
15. S. Lucks. A collision-resistant rate-1 double-block-length hash function. In E. Biham, H. Handschuh, S. Lucks, and V. Rijmen, editors, *Symmetric Cryptography*, number 07021 in Dagstuhl Seminar Proceedings, Dagstuhl, Germany, 2007. Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany.
16. R. Matsumoto, K. Kurosawa, T. Itoh, T. Konno, and T. Uyematsu. Primal-dual distance bounds of linear codes with application to cryptography. *IEEE Transactions on Information Theory*, 52(9):4251–4256, 2006.
17. R. C. Merkle. One way hash functions and DES. In Brassard [2], pages 428–446.
18. M. Nandi, W. Lee, K. Sakurai, and S. Lee. Security analysis of a 2/3-rate double length compression function in black-box model. In H. Gilbert and H. Handschuh, editors, *FSE'05*, volume 3557 of *LNCS*, pages 243–254. Springer, Heidelberg, 2005.

19. O. Özen and M. Stam. Another glance at double-length hashing. In Parker [20], pages 176–201.
20. M. G. Parker, editor. *Cryptography and Coding 2009*, volume 5921 of *LNCS*. Springer, Heidelberg, 2009.
21. T. Peyrin, H. Gilbert, F. Muller, and M. Robshaw. Combining compression functions and block cipher-based hash functions. In X. Lai and K. Chen, editors, *Advances in Cryptography—Asiacrypt'06*, volume 4284 of *LNCS*, pages 315–331. Springer, Heidelberg, 2006.
22. B. Preneel, R. Govaerts, and J. Vandewalle. Hash functions based on block ciphers: A synthetic approach. In D. Stinson, editor, *Advances in Cryptography—Crypto'93*, volume 773 of *LNCS*, pages 368–378. Springer, Heidelberg, 1993.
23. P. Rogaway and T. Shrimpton. Cryptographic hash-function basics: Definitions, implications and separations for preimage resistance, second-preimage resistance, and collision resistance. In B. K. Roy and W. Meier, editors, *FSE*, volume 3017 of *LNCS*, pages 371–388. Springer, Heidelberg, 2004.
24. P. Rogaway and J. Steinberger. Security/efficiency tradeoffs for permutation-based hashing. In N. P. Smart, editor, *Advances in Cryptography—Eurocrypt'08*, volume 4965 of *LNCS*, pages 220–236. Springer, Heidelberg, 2008.
25. R. Schroeppel and A. Shamir. A $T = O(2^{n/2})$, $S = O(2^{n/4})$ algorithm for certain np-complete problems. *SIAM Journal on Computing*, 10:456–464, 1981.
26. Y. Seurin and T. Peyrin. Security analysis of constructions combining FIL random oracles. In A. Biryukov, editor, *FSE'07*, volume 4593 of *LNCS*, pages 119–136. Springer, Heidelberg, 2007.
27. T. Shrimpton and M. Stam. Building a collision-resistant compression function from non-compressing primitives. In *ICALP 2008, Part II*, volume 5126, pages 643–654. Springer-Verlag, 2008.
28. M. Stam. Beyond uniformity: Better security/efficiency tradeoffs for compression functions. In D. Wagner, editor, *Advances in Cryptography—Crypto'08*, volume 5157 of *LNCS*, pages 397–412. Springer, Heidelberg, 2008.
29. M. Stam. Block cipher based hashing revisited. In Dunkelman [6], pages 67–83.
30. D. Wagner. A generalized birthday problem. In Yung [32], pages 288–303.
31. D. Watanabe. A note on the security proof of Knudsen-Preneel construction of a hash function. Unpublished manuscript, Available at `http://csrc.nist.gov/groups/ST/hash/documents/WATANABE_kp_attack.pdf`, 2006.
32. M. Yung, editor. *Advances in Cryptography—Crypto'02*, volume 2442 of *LNCS*. Springer, Heidelberg, 2002.
33. G. Yuval. How to swindle rabin. *Cryptologia*, 3:187–189, 1979.

A Runtime Analysis (Proof of Theorems 15 and 17)

Proof (Theorem 15 and 17). We will proceed step by step to prove our claims. The first three steps are common for the MDS and non-MDS case (and for MDS codes $d^\perp = k + 1$). The remaining steps are treated separately. In complexity estimations, we mainly ignore the effect of constants and polylogarithmic factors.

1. QUERY PHASE. In this step, the total number of queries asked per PuRF is $2^{rn/k}$, so $|Q_i| \approx 2^{rn/k}$ and $|L_i| \approx 2^{(r-k)n/k}$ for all $i = 1, \ldots, r$. Hence, this step requires about $2^{rn/k}$ PuRF evaluations and $2^{(r-k)n/k}$ cn-bit of memory.

2. FIRST MERGE PHASE. The main computational part of this step is the generation of the lists $\tilde{L}_{h_j}$ for $j = 0, 1$. By construction, $|\chi(h_0)| = \lfloor d^\perp/2 \rfloor$ and $|\chi(h_1)| = \lceil d^\perp/2 \rceil$ where $|L_i| = 2^{(r-k)n/k}$. Hence, the time required for generating $\tilde{L}_{h_0}$ and $\tilde{L}_{h_1}$ is determined by their sizes, i.e. $2^{(r-k)\lfloor d^\perp/2 \rfloor n/k}$ and $2^{(r-k)\lceil d^\perp/2 \rceil n/k}$ respectively which is clearly dominated by the latter.

3. FIRST JOIN PHASE. This step constructs L_h by finding collisions between $\tilde{L}_{h_0}$ and $\tilde{L}_{h_1}$ in their second components. Since $|\tilde{L}_{h_0}| \cdot |\tilde{L}_{h_1}| \approx 2^{d^\perp(r-k)n/k}$ and we are interested in collisions on rn/k bits, we have $|L_h| \approx 2^{(d^\perp(r-k)-r)n/k}$ (which is equal to $2^{(r-k-1)n}$ for MDS codes once $d^\perp = k + 1$ is substituted). Here, each colliding element is directly forwarded to the next step without even storing L_h (Collision search can be performed with step 2 in conjunction roughly in $2^{(r-k)\lceil d^\perp/2 \rceil n/k}$ time and $2^{(r-k)\lfloor d^\perp/2 \rfloor n/k}$ memory).

4. SECOND MERGE PHASE and 5. SECOND JOIN PHASE (for non-MDS codes). At the end of this step, for a fair number of codes suggested by Knudsen and Preneel we reach *full rank*. Namely, for all $i \in \{1 \ldots k\}$ and $i \notin \chi(h)$ we have $i \in \chi(h')$.

This results in $k - d^\perp + 2$ lists (L_i for which $i \notin \chi(h)$ and $i \in \chi(h')$) with $|L_i| = 2^{(r-k)n/k}$ and L_h with $|L_h| = 2^{(d^\perp(r-k)-r)n/k}$ to be used with the relation defined by h'. Hence, regardless of the way of MERGEing and JOINing there will be $2^{(r-k+\frac{r}{k}-2)n}$ elements in total to be checked for collisions. Collisions will be searched similarly as done in previous step on rn/k bits. This leads to a list $L_{h'}$ of roughly $|L_{h'}| = 2^{(r-k-2)n}$ elements at the end of SECOND JOIN PHASE.

What is tricky in this step is to find the optimal way of merging. So, we need to find the sets $\chi(h'_0)$ and $\chi(h'_1)$ such that $\chi(h'_0) \cap \chi(h'_1) = \emptyset$ and merging time is optimal. In fact, the main task is to distribute

$2^{(r-k+\frac{r}{k}-2)n}$ elements as evenly as possible without violating the constraints imposed by the asymmetric list sizes.

We proceed as follows. Let $i \in \{1, ..., k-d^{\perp}+1\}$ be an integer such that $|\chi(h'_1)| = (k-d^{\perp}+2-i)$ and $|\chi(h'_0)| = i+1$ where $|\tilde{L}_{h'_1}| = 2^{(k-d^{\perp}+2-i)(r-k)/k}$ and $|\tilde{L}_{h'_0}| = 2^{i(r-k)/k+(d^{\perp}(r-k)-r)/k}$. Assuming the algorithm follows for this particular i, the merging time will be the maximum of the latter two whereas the storage requirement is similarly the minimum of the same pair. In order to optimize the overall time complexity, we take the minimum over all i and denote the value by T_1. Collision finding then can be performed in $2^{T_1 n}$ time with a storage requirement of roughly $2^{T_2 n}$ cn-bits where T_2 is the corresponding value depending on T_1.

6. FINALIZATION. For MDS codes, after the FIRST JOIN PHASE, we have a list L_h of size (substituting $d^{\perp} = k+1$) roughly $|L_h| = 2^{(d^{\perp}(r-k)-r)n/k} = 2^{(r-k-1)n}$. In this step, we will keep checking memberships for the lists L_j for each elements in L_h for $j \notin \chi(h)$ until we end up with a preimage. Due to the MDS property ($|\chi(h)| = k+1$), it is always possible to find dual codewords satisfying $i \in \chi(h)$ for all $i \in \{1, \ldots, k\}$. So, we need to check membership lists only for the lists L_j for $j > k+1$.

As $|L_h| = 2^{(r-k-1)n}$ and $|L_j| = 2^{(r-k)n/k}$, after the first membership check (again correspondence on $2^{rn/k}$ bits), there remain $2^{((r-k-1)+(r-k)/k-r/k)n} = 2^{(r-k-2)n}$ possible tuples. So, the number of possible preimage candidates W' is reduced by a factor of 2^n per membership check. Since $r-k$ is a small integer, the algorithm produces a preimage after a finite number of steps. Time complexity of this step is dominated by the first stage of this step; i.e. $2^{(r-k-1)n}$ operations. So, the general statement follows for MDS case.

For $d = 3$ we have $r - k = 2$ as $\mathcal{C}$ is MDS. So, the FINALIZATION takes only 2^n operations which is clearly dominated by the QUERY PHASE requiring $2^{rn/k}$ PuRF evaluations where $r > k$. JOIN and MERGE PHASES, on the other hand, can be performed in $2^{2\lceil (k+1)/2 \rceil n/k}$ time using $2^{2\lfloor (k+1)/2 \rfloor n/k}$ memory. Since $r > k$, the former is clearly dominated by $2^{rn/k}$ whereas the latter can be upper bounded by $2^{(k+1)n/k}$. Hence, we always get optimal time complexity for MDS case whenever $d = 3$.

For non-MDS case, we have $|L_{h'}| = 2^{(r-k-2)n}$ and $|L_j| = 2^{(r-k)n/k}$ for $j \notin \chi(h)$ and $j \in \chi(h')$. This results similarly in $2^{(r-k+\frac{(r-k)}{k}-2)n}$ elements in total which need to collide on cn bits, of which $cn - rn/k$ bits are set to zero. So, the expected number of preimage candidates W' is about $2^{(r-k+\frac{(r-k)}{k}-2)n-\frac{rn}{k}} = 2^{(r-k-3)n}$. Hence, the number of possible tuples is again reduced by a factor of 2^n per membership check. Similarly, because $r-k$ is a small integer, this step takes finite number of iterations. Note also that time complexity of this step is dominated by $|L_{h'}|$.

So, picking up the obtained complexities gives the desired overall complexity. (See our results on the compression functions suggested by Knudsen and Preneel in Tables 3 and 4).

□

Remark 18. For the schemes for which our attack is not optimal the time complexity is higher than the query complexity. Thus it is natural to consider increasing the query complexity in order to bring the overall complexity down, as was so effectively demonstrated by Wagner [30] for the ordinary generalized birthday problem. However, our situation differs crucially from that ordinary one.

Normally the lists are constructed as *outputs* of a compression function (or PuRF). Thus increasing the amount of queries, simply leads to more elements (uniformly distributed) in the same set. In other words, the density of available elements increases with the number of queries. In our setting, the lists are constructed as *inputs* of a compression function, depending on whether they hit a specific target. Moreover, for our attack to work, we choose our inputs very carefully and exhaust all inputs of the prescribed format. To increase the number of queries we need to relax the prescribed format, so although we get more answers, they will be chosen from a *larger* space (which will be felt throughout the algorithm). Thusly, the density of elements remains the same (namely 2^{-n}), making it less likely for a Wagner-style attack to go through. (From a more technical perspective, the way Wagner achieves his speedup is by selecting those elements that have a certain zero-bit-pattern; since we already control the zero-bit pattern of the elements that end up in the list to some extent, intuitively there is less for us to gain by relaxing this control.)

B Attacking $H = \mathsf{KP}([8,5,3]_{2^2})$ in $\mathcal{O}(2^{8n/5})$ Time

To illustrate our preimage attack on non-MDS schemes, we use $H = \mathsf{KP}([8,5,3]_4)$ as an example. As in the $\mathsf{KP}([5,3,3]_4)$ example, we are able to mount a preimage attack with almost optimal time complexity.

Claim. For the compression function $H = \mathsf{KP}([8,5,3]_4)$ preimages can be found in $\mathcal{O}(2^{8n/5})$ time with a memory requirement of $\mathcal{O}(2^{6n/5})$ $2n$-bits.

Proof. In this case, we are building a $10n \rightarrow 8n$ compression function using eight calls to respective $2n \rightarrow n$ ideal primitives. So, given the target digest $Z = z_1||\dots||z_8$, our aim is now to find PuRF inputs $x_i = (x_i^1||x_i^2) \in \{0,1\}^{2n}$ such that $f_i(x_i) = z_i$ holds for all $i = 1,\dots,8$, and $X = (\mathsf{C}^{\mathrm{PRE}} \otimes I_{n'}) \cdot W$ for some compression function input W (where X is comprised of the eight x_i). In this case W is a preimage for Z. We refer to Fig. 3 for the illustration and follow with the details.

Below is a systematic generator matrix for the code $[8,5,3]_{2^2}$ that we use as a frame reference (note that the choice of this matrix does not affect the runtime of our attack as long as $d^{\perp} = 4$):

$$G = \begin{pmatrix} 1\,0\,0\,0\,0 & 0 & w^2 & w^2 \\ 0\,1\,0\,0\,0 & w^2 & w & 1 \\ 0\,0\,1\,0\,0 & 0 & w^2 & w \\ 0\,0\,0\,1\,0 & w & 1 & 0 \\ 0\,0\,0\,0\,1 & w^2 & 0 & w^2 \end{pmatrix}.$$

Using the G above and defining φ by (the one in Section 6.1)

$$\varphi(0) = \begin{pmatrix} 0\,0 \\ 0\,0 \end{pmatrix}, \varphi(1) = \begin{pmatrix} 1\,0 \\ 0\,1 \end{pmatrix}, \varphi(w) = \begin{pmatrix} 1\,1 \\ 0\,1 \end{pmatrix} \text{ and } \varphi(w^2) = \begin{pmatrix} 0\,1 \\ 1\,1 \end{pmatrix},$$

we can construct the corresponding $\mathsf{C}^{\mathrm{PRE}}$.

In the QUERY PHASE, we ask queries (to each PuRF) of a special form and keep the partial preimage lists to be used in the later stages of the algorithm. More precisely, we pick the queries of the form $x_i^1, x_i^2 \in 0^{n/5} \times \{0,1\}^{4n/5}$ to evaluate $f_i(x_i^1, x_i^2)$ and keep a list L_i of pairs that hit the target partial digest z_i for all $i = 1,\dots,8$. As a result, $2^{8n/5}$ elements are queried per PuRF (in $\mathcal{O}(2^{8n/5})$ time) which produces an approximate partial preimage size of $|L_i| \approx 2^{3n/5}$ (as each partial digest is assumed to be hit with a probability of 2^{-n}).

Next, we make use of the dual codeword $h = (0\,w^2\,0\,w\,w^2\,1\,0\,0)$ in the FIRST MERGE-JOIN PHASE so as to efficiently prune the irrelevant preimage candidates. This will be done using the techniques (related to generalized birthday or k-list problem) described in Sec. 6 and 7. We have the following relations (defined by the above h) between the inputs of the PuRFs:

$$x_2^2 \oplus x_4^1 \oplus x_4^2 \oplus x_5^2 = x_6^1 \text{ and } x_2^1 \oplus x_2^2 \oplus x_4^1 \oplus x_5^1 \oplus x_5^2 = x_6^2$$

Note that the merging will be performed with the conditions $\chi(h_0) = \{2,4\}$ and $\chi(h_1) = \{5,6\}$. We now create the lists $\tilde{L}_{h_j}$, for $j = 0,1$:

$$\tilde{L}_{h_j} = \left\{ (X_j, (\bar{\varphi}(h_j) \otimes I_{n'}) \cdot (X_j + 0)) \mid X_j \in \sum_{i \in \chi(h_j)} L_i \right\}$$

both sorted again on their second component. More precisely, we have the lists $\tilde{L}_{h_0}$ and $\tilde{L}_{h_1}$ of the form

$$\tilde{L}_{h_0} = \left\{ (x_2, x_4), \left((x_2^2 \oplus x_4^1 \oplus x_4^2) || (x_2^1 \oplus x_2^2 \oplus x_4^1) \right) \mid (x_2, x_4) \in L_2 \times L_4 \right\},$$
$$\tilde{L}_{h_1} = \left\{ (x_5, x_6), \left((x_5^2 \oplus x_6^1) || (x_5^1 \oplus x_5^2 \oplus x_6^2) \right) \mid (x_5, x_6) \in L_5 \times L_6 \right\}.$$

Since $|L_i| \approx 2^{3n/5}$, creating $\tilde{L}_{h_j}$ takes roughly $\mathcal{O}(n2^{6n/5})$ time and $\mathcal{O}(2^{6n/5})$ memory.

The lists $\tilde{L}_{h_j}$ contribute roughly $2^{12n/5}$ elements in total for the FIRST JOIN PHASE which need to collide on $2n$ bits, (in the second component of the lists $\tilde{L}_{h_j}$) of which $2n/5$ bits are set to zero;

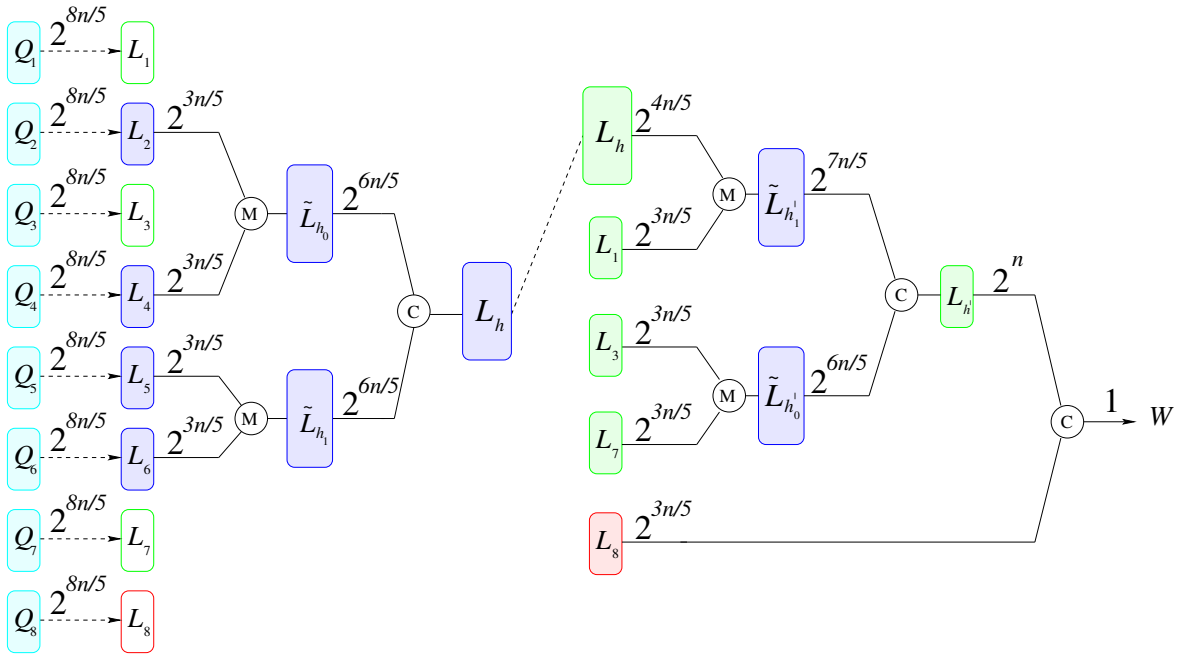

Fig. 3. An illustration of our preimage attack on $H = \mathsf{KP}([8, 5, 3]_4)$. Here, M denotes the merging of two lists whereas C denotes the collision search between two lists. The size of the lists is given by labelling the inputs to the M and C operations.

hence $|L_h| \approx 2^{12n/5}/2^{(2-2/5)n} = 2^{4n/5}$. Similar to the attack in Section 6.1, collision finding can be performed in conjunction with the FIRST MERGE PHASE in $\mathcal{O}(n2^{6n/5})$ time and $\mathcal{O}(2^{6n/5})$ memory.

Now, by using the dual codeword $h' = (w^2\ w\ w^2\ 1\ 0\ 0\ 1\ 0)$, we will perform another MERGE-JOIN PHASE (as we cannot generate a unique preimage W using the elements contained in L_h and it is more efficient in terms of time complexity). In order to further optimize the time complexity, we take $\chi(h_0{}') = \{1, 2, 4\}$ and $\chi(h_1{}') = \{3, 7\}$ to create the lists $\tilde{L}_{h_j{}'}$, for $j = 0, 1$ (note that this is the optimal solution considering the asymmetry between the list sizes). Projected to the L_2 and L_4 components, we get only $2^{4n/5}$ possibilities in L_h corresponding to L_2 and L_4. Each of he remaining lists L_1, L_3 and L_7, on the other hand, contains roughly $2^{3n/5}$ elements. Therefore, $|\tilde{L}_{h_1{}'}| \approx 2^{7n/5}$ and $|\tilde{L}_{h_0{}'}| \approx 2^{6n/5}$. Since both lists contribute roughly $2^{13n/5}$ elements which need to collide on $2n$ bits, of which $2n/5$ bits are set to zero, we have $|L_{h'}| = 2^{13n/5}/2^{(2-2/5)n} = 2^n$. Collision finding can be performed in conjunction with the FIRST MERGE-JOIN PHASE in $\mathcal{O}(n2^{7n/5})$ time and $\mathcal{O}(2^{6n/5})$ memory.

We now have 2^n preimage candidates W' satisfying $C_6^{\mathrm{PRE}}(W') = x_6$ and $C_7^{\mathrm{PRE}}(W') = x_7$. So, for the FINALIZATION, we have a list of 2^n partial preimages on all but one output. So, we expect one of these partial preimages to be part of list L_8 and be a full collision; hence creates a preimage. This step can be performed by simply doing another membership-check with the list L_8. Time complexity of this step (which is $\mathcal{O}(2^n)$) is dominated by the previous stages of the algorithm. Therefore, picking up the stated complexities results in a time complexity of $\mathcal{O}(2^{8n/5})$ (due to QUERY PHASE) with an additional memory requirement of $\mathcal{O}(2^{6n/5})$ $2n$-bits. □

C The Codes for Knudsen Preneel Compression Functions

D A Space-Efficient Preimage Attack

In the preimage attacks presented in Sec. 6 and 7, our main tool (to reduce the time complexity in the MERGE and JOIN PHASES) we consider is a special instance of the well known k-list problem: Given k-lists $L_1 \ldots, L_k$ of elements drawn uniformly and independently random from $\{0, 1\}^n$, find $x_i \in L_i$ for

Code	$sn + mn \to sn$	Dual Code	P^T
$\mathbb{F}_{2^2} - [5, 3, 3]$	$6n \to 5n$	$[5, 2, 4]$	$\begin{pmatrix} 1 & 1 & 1 \\ 1 & w & w^2 \end{pmatrix}$
$\mathbb{F}_{2^2} - [8, 5, 3]$	$10n \to 8n$	$[8, 3, 4]$	$\begin{pmatrix} 0 & w^2 & 0 & w & w^2 \\ w^2 & w & w^2 & 1 & 0 \\ w^2 & 1 & w & 0 & w^2 \end{pmatrix}$
$\mathbb{F}_{2^2} - [12, 9, 3]$	$18n \to 12n$	$[12, 3, 7]$	$\begin{pmatrix} w^2 & 1 & w & 0 & 0 & 0 & w^2 & w^2 & w^2 \\ w^2 & 0 & 1 & 1 & 1 & w & 0 & w & w \\ w^2 & w^2 & 0 & w^2 & 1 & w^2 & w^2 & w^2 & 1 \end{pmatrix}$
$\mathbb{F}_{2^2} - [9, 5, 4]$	$10n \to 9n$	$[9, 4, 4]$	$\begin{pmatrix} 0 & w & 1 & w & 0 \\ 1 & w^2 & 0 & 1 & w^2 \\ 1 & 1 & w^2 & 0 & 1 \\ w & w & w & w^2 & w^2 \end{pmatrix}$
$\mathbb{F}_{2^2} - [16, 12, 4]$	$24n \to 16n$	$[16, 4, 11]$	$\begin{pmatrix} 1 & w^2 & 0 & 1 & w^2 & w & w & w^2 & 1 & 0 & w^2 & 1 \\ 0 & w & w^2 & 1 & w & 1 & 0 & 1 & w & 1 & w^2 & w \\ w^2 & 1 & w & 1 & 0 & 1 & w & 1 & w^2 & w & 0 & 1 \\ w^2 & 0 & 1 & w^2 & w & w & w^2 & 1 & 0 & w^2 & 1 & 1 \end{pmatrix}$
$\mathbb{F}_{2^4} - [6, 4, 3]$	$8n \to 6n$	$[6, 2, 5]$	$\begin{pmatrix} w^8 & w^{12} & w^{14} & w^9 \\ w^{12} & w^{14} & w^9 & 1 \end{pmatrix}$
$\mathbb{F}_{2^4} - [8, 6, 3]$	$12n \to 8n$	$[8, 2, 7]$	$\begin{pmatrix} w^{10} & w^7 & w^8 & w^{12} & w^{14} & w^9 \\ w^7 & w^8 & w^{12} & w^{14} & w^9 & 1 \end{pmatrix}$
$\mathbb{F}_{2^4} - [12, 10, 3]$	$20n \to 12n$	$[12, 2, 11]$	$\begin{pmatrix} w^7 & w^{10} & w^3 & w^3 & w^{10} & w^7 & w^8 & w^{12} & w^{14} & w^9 \\ w^{10} & w^3 & w^3 & w^{10} & w^7 & w^8 & w^{12} & w^{14} & w^9 & 1 \end{pmatrix}$
$\mathbb{F}_{2^4} - [9, 6, 4]$	$12n \to 9n$	$[9, 3, 8]$	$\begin{pmatrix} w^4 & w^8 & w^6 & w^{11} & w^2 & w^{14} \\ w^{10} & w^3 & w^4 & w^7 & w^6 & w^{14} \\ w^8 & w^6 & w^{11} & w^2 & w^{14} & 1 \end{pmatrix}$
$\mathbb{F}_{2^4} - [16, 13, 4]$	$26n \to 16n$	$[16, 3, 14]$	$\begin{pmatrix} w^{14} & w^2 & w^{11} & w^6 & w^8 & w^4 & w^{12} & w^4 & w^8 & w^6 & w^{11} & w^2 & w^{14} \\ w^6 & w^7 & w^4 & w^3 & w^{10} & w^{13} & w^{13} & w^{10} & w^3 & w^4 & w^7 & w^6 & w^{14} \\ w^2 & w^{11} & w^6 & w^8 & w^4 & w^{12} & w^4 & w^8 & w^6 & w^{11} & w^2 & w^{14} & w^1 \end{pmatrix}$

Table 6. The codes over $\mathbb{F}_{2^2} - \mathbb{F}_{2^4}$ and the leading compression functions suggested by Knudsen-Preneel. Here, the generator matrix G is of the form $G = [I_k|P]$ and the table contains P^T.

Code	$sn + mn \to sn$	Dual Code	P^T
$\mathbb{F}_{2^3} - [4, 2, 3]$	$6n \to 4n$	$[4, 2, 3]$	$\begin{pmatrix} w^3 & w^4 \\ w^4 & 1 \end{pmatrix}$
$\mathbb{F}_{2^3} - [6, 4, 3]$	$12n \to 6n$	$[6, 2, 5]$	$\begin{pmatrix} w^5 & w^5 & w^3 & w^4 \\ w^5 & w^3 & w^4 & 1 \end{pmatrix}$
$\mathbb{F}_{2^3} - [9, 7, 3]$	$21n \to 9n$	$[9, 2, 8]$	$\begin{pmatrix} 1 & w^4 & w^3 & w^5 & w^5 & w^3 & w^4 \\ w^4 & w^3 & w^5 & w^5 & w^3 & w^4 & w^1 \end{pmatrix}$
$\mathbb{F}_{2^3} - [5, 2, 4]$	$6n \to 5n$	$[5, 3, 3]$	$\begin{pmatrix} w^4 & w^3 \\ w^2 & w^3 \\ w^3 & 1 \end{pmatrix}$
$\mathbb{F}_{2^3} - [7, 4, 4]$	$12n \to 7n$	$[7, 3, 5]$	$\begin{pmatrix} w^4 & w^6 & w^4 & w^3 \\ w & w & w^2 & w^3 \\ w^6 & w^4 & w^3 & 1 \end{pmatrix}$
$\mathbb{F}_{2^3} - [10, 7, 4]$	$21n \to 10n$	$[10, 3, 8]$	$\begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & w & w^2 & w^3 & w^4 & w^5 & w^6 \\ 1 & w^2 & w^4 & w^6 & w & w^3 & w^5 \end{pmatrix}$

Table 7. The codes over $\mathbb{F}_{2^3}$ and the leading compression functions suggested by Knudsen-Preneel. Here, the generator matrix G is of the form $G = [I_k|P]$ and the table contains P^T.

all $i \in 1, \ldots, k$ such that $x_1 \oplus \ldots \oplus x_k = 0$. (In the sequel, we simply focus on the case $k = 4$ to better visualize the known results and its application to our case. The generalization will follow subsequently.)

The technique used in our attacks is the trivial way to solve the k-list problem (by creating two lists through merging and looking for collisions in the corresponding entries). Despite of the fact that this approach provides an efficient attack in terms of time complexity (at least for our case,) it is not sufficiently space efficient. We deal with this issue by adapting known techniques [4, 30] (if at all possible) to our case without violating the time complexity of our attack.

Known Results in brief. Note that this problem is highly useful in cryptologic content and it was studied in a flaw of papers [4, 30, 25, 3] (each providing a solution with different time/memory trade-offs) with a lot of significant applications. If we look at a particular case where $k = 4$, it is easy to see that to find a single solution for 4-list problem (with high probability), the cardinalities of the lists should be around $2^{n/4}$. Interestingly enough, this also serves as the minimum memory requirement [4, 25] lying in the trade-off space among all the known solutions to 4-list problem. For the time complexity, on the other hand, the known algorithms requiring $2^{n/4}$ memory could reduce the attack time only upto $2^{n/2}$. This is always the case for a single solution. In his famous work, Wagner [30] presented an algorithm (exploiting another point in the trade-off space) that draws down the time requirement to $2^{n/3}$ using a memory of roughly $2^{n/3}$. This is the best known result (in terms of time complexity) to find a (single) solution to the 4-list problem.

Adaptation to our case. The major difference in our attacks is the fact that we are interested in all solutions of the k-list problem (this is what we require in the FINALIZATION step) which can be found either by adapting Wagner's technique or using the algorithm of Chose et al. [4] (the latter is essentially equivalent to the former when the problem is modified properly for $k = 4$, see the application on $H = \mathsf{KP}([5, 3, 3]_4)$ below). Moreover, we need to deal with the issue that the relations in our attack for which we need to find solutions may vary. However, this problem is not as tough as it seems as shown in Sec. 6 and 7.

To find all solutions, it is possible to apply Wagner's algorithm iteratively; yet it may not be the best approach. For instance, let us take a look at the relevant steps (MERGE and JOIN PHASES) in our warm-up example from Sec. 6.1; if we apply Wagner's algorithm 2^n times (to find that many solutions) each with $|L_i| = 2^{n/3}$, (although we can significantly reduce the memory requirements) we need to generate $2^{4n/3}$ partial preimages in total in the QUERY PHASE which will obviously blow up the time complexity of this step; hence the overall attack. Instead, we will adapt a more involved technique which is due to Chose et al. [4] that can be seen as a special instance of Wagner's tree algorithm (in particular for $k = 4$). Here is the novel attack applied on $H = \mathsf{KP}([5, 3, 3]_4)$:

Claim. For the compression function $H = \mathsf{KP}([5, 3, 3]_4)$, preimages in H_n can be found in $\mathcal{O}(2^{5n/3})$ time with a memory requirement of $\mathcal{O}(2^{2n/3})$ n-bit blocks.

Proof. The only difference of this attack from the one given in Sec. 6.1 is that now we present a more space-efficient version of the MERGE and JOIN phases. Thus, we refer to Sec. 6.1 for the details of the QUERY PHASE and FINALIZATION and simply follow with the MERGE and JOIN phases (where we need to find all the solutions of the 4-list problem over the lists $L_1, \ldots, L_4$). We refer to Fig. 4 for a better illustration.

The list of tuples containing all the solutions of the 4-list problem is defined similarly by

$$L_{\{1,2,3,4\}} = \{(x_1, x_2, x_3, x_4) \in L_1 \times L_2 \times L_3 \times L_4 \mid x_1 \oplus x_2 \oplus x_3 \oplus x_4 = 0\} \ .$$

We now follow a step-wise approach using a Wagner style tree-like algorithm given the lists L_i with $|L_i| \approx 2^{2n/3}$. Let l be a positive integer less than $2n$. We denote by $(x_i)_l$ the concatenation of the least significant $l/2$ bits of the $n-$bit strings x_i^1 and x_i^2. We first construct the lists $\tilde{L}_{\{1,2\}}$ and $\tilde{L}_{\{3,4\}}$ defined by (for some l-bit constant c).

$$\tilde{L}_{\{1,2\}} = \{(x_1, x_2) \mid (x_1)_l \oplus (x_2)_l = c \text{ and } (x_1, x_2) \in L_1 \times L_2\} \, ,$$
$$\tilde{L}_{\{3,4\}} = \{(x_3, x_4) \mid (x_3)_l \oplus (x_4)_l = c \text{ and } (x_3, x_4) \in L_3 \times L_4\}$$

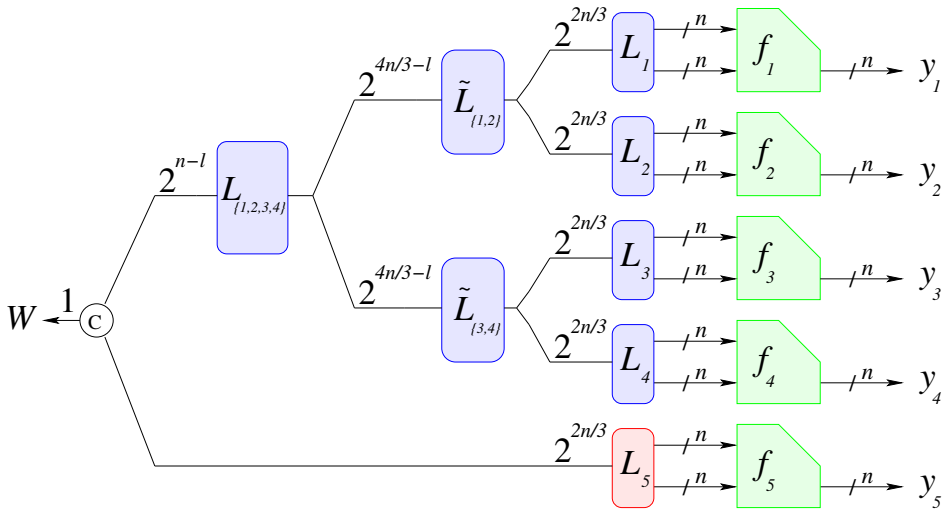

Fig. 4. Our space-efficient preimage attack on the Knudsen-Preneel compression function $H = \mathsf{KP}([5,3,3]_{2^2})$ illustrated. The attack works for $l = 2^{2n/3}$.

Note that constructing each list takes $\mathcal{O}(2^{2n/3})$ time (for scanning e.g. L_1 and L_3) and $\mathcal{O}(2^{2n/3})$ memory (for storing $L_1, \ldots, L_4$). We are now interested in the cardinalities of the lists $\tilde{L}_{\{1,2\}}$ and $\tilde{L}_{\{3,4\}}$ as they are the main factors affecting the time and memory complexity of this phase. It is expected that both lists have roughly $2^{4n/3-l}$ elements as one can generate $2^{4n/3}$ tuples in total (each from $L_1 - L_2$ and $L_3 - L_4$) and the probability of hitting one of the $c \in \{0,1\}^l$ is 2^{-l}. So, we are done with this middle step. Next we look for the correspondence on the remaining $5n/3 - l$ bits of the surviving elements in $\tilde{L}_{\{1,2\}}$ and $\tilde{L}_{\{3,4\}}$ to obtain all the solutions.

As there are $2^{8n/3-2l}$ tuples in total ($2^{4n/3-l}$ from each of $\tilde{L}_{\{1,2\}}$ and $\tilde{L}_{\{3,4\}}$) and we are interested in correspondence on $5n/3 - l$ bits, we expect to produce $2^{8n/3-2l} \cdot 2^{l-5n/3} = 2^{n-l}$ solutions, i.e. $|L_{\{1,2,3,4\}}| = 2^{n-l}$. Constructing $L_{\{1,2,3,4\}}$ takes (on average) $\mathcal{O}(2^{4n/3-l})$ time and $\mathcal{O}(2^{4n/3-l})$ memory. So, picking $l = 2n/3$ gives the optimal solution for this particular case. Hence, we can produce $2^{n/3}$ solutions for the 4-list problem in $\mathcal{O}(2^{2n/3})$ time using $\mathcal{O}(2^{2n/3})$ n-bit of memory for single $c \in \{0,1\}^{2n/3}$.

Now, the idea is to iterate this technique for all $c \in \{0,1\}^{2n/3}$ to obtain 2^n solutions at the end (which is essentially what we require for the FINALIZATION phase). Note that, the trick is to rid of the internal lists $\tilde{L}_{\{1,2\}}$ and $\tilde{L}_{\{3,4\}}$ at the end of each iteration to keep the memory requirements small. All in all, MERGE and JOIN phases take $\mathcal{O}(2^{4n/3})$ time ($2^{2n/3}$ iterations each requiring $\mathcal{O}(2^{2n/3})$ time) and $\mathcal{O}(2^{2n/3})$ memory. Hence, the time complexity of the overall attack is still dominated by the QUERY PHASE and the memory requirement is $\mathcal{O}(2^{2n/3})$. So, the claim follows.

Generalization to $k > 4$. The main idea of generalizing the algorithm for $k > 4$ is to reduce the k-list problem to solving a respective 4-list problem. In brief, the algorithm works as follows. Firstly, the k lists are evenly divided into four groups of lists each containing $l_1, \ldots, l_4$ smaller lists. Namely, we first determine the integers $l_1, \ldots, l_4$ with $l_1 \geq l_2$, $l_3 \geq l_4$ such that $l_1 + l_2 + l_3 + l_4 = k$ and $l_i = \lfloor k/4 \rfloor$ or $l_i = \lceil k/4 \rceil$ for $i \in 1, \ldots, 4$. Then, a merging phase is performed (similar to the one presented in our preimage attacks) in each group to generate the larger lists. Finally, the algorithm to find the all solutions of the 4-list problem is mounted over the newly generated four big lists. In turn, we obtain all the solutions to the corresponding k-list problem. We refer to [4] for the details of the algorithm.

In our preimage attacks, it is always the case that (except the non-MDS case where we need second merging and joining) the relevant lists have (roughly) the same number of elements. Therefore, for the list sizes of N, the above algorithm is expected to produce the solutions in $\mathcal{O}(N^{\max(l_1+l_2,l_3+l_4)} logN)$ time using $\mathcal{O}(N^{\max(\min(l_1,l_2),\min(l_3,l_4))})$ memory. When there is an asymmetry between the list sizes, on the other hand, the straightforward adaptation follows (cf. Sec. 7). We note that because $d^{\perp} < 4$ for

$\mathsf{KP}([4,2,3]_8)$ and $\mathsf{KP}([5,2,4]_8)$, we cannot apply this space-efficent algorithm for these compression functions. These are the only exceptions. We present the ramifications of the algorithm to all other compression functions suggested by Knudsen and Preneel in Tab.8 and 9.

Table 8. Space-efficient results on $2n$-to-n bit primitive (PuRF or single-key blockcipher) Knudsen-Preneel Compression Functions. Non-MDS parameters in italic.

Code	$sn+mn \to sn$	Our Attack			KP Results		
		Query Complexity	Time Complexity	Memory Complexity	KP-Conj. Time	KP-Attack Time	KP-Attack Memory
$[r,k,d]_{2^e}$	$2kn \to rn$	$2^{rn/k}$			$2^{(d-1)n}$	Thm. 7	Thm. 7
$[5,3,3]_4$	$(5+1)n \to 5n$	$2^{5n/3}$	$2^{5n/3}$	$2^{2n/3}$	2^{2n}	2^{2n}	$2^{2n/3}$
$[\mathit{8},\mathit{5},\mathit{3}]_4$	$(8+2)n \to 8n$	$2^{8n/5}$	$2^{8n/5}$	$2^{3n/5}$	2^{2n}	2^{3n}	$2^{3n/5}$
$[\mathit{12},\mathit{9},\mathit{3}]_4$	$(12+6)n \to 12n$	$2^{4n/3}$	$2^{4n/3}$	2^{n}	2^{2n}	2^{3n}	$2^{n/3}$
$[\mathit{9},\mathit{5},\mathit{4}]_4$	$(9+1)n \to 9n$	$2^{9n/5}$	$2^{11n/5}$	$2^{7n/5}$	2^{3n}	2^{4n}	$2^{4n/5}$
$[\mathit{16},\mathit{12},\mathit{4}]_4$	$(16+8)n \to 16n$	$2^{4n/3}$	$2^{7n/3}$	2^{n}	2^{3n}	2^{4n}	$2^{n/3}$
$[6,4,3]_{16}$	$(6+2)n \to 6n$	$2^{3n/2}$	$2^{3n/2}$	$2^{n/2}$	2^{2n}	2^{2n}	$2^{n/2}$
$[8,6,3]_{16}$	$(8+4)n \to 8n$	$2^{4n/3}$	$2^{4n/3}$	2^{n}	2^{2n}	2^{2n}	$2^{n/3}$
$[12,10,3]_{16}$	$(12+8)n \to 12n$	$2^{6n/5}$	$2^{6n/5}$	$2^{3n/5}$	2^{2n}	2^{2n}	$2^{n/5}$
$[9,6,4]_{16}$	$(9+3)n \to 9n$	$2^{3n/2}$	2^{2n}	2^{n}	2^{3n}	2^{3n}	$2^{n/2}$
$[16,13,4]_{16}$	$(16+10)n \to 16n$	$2^{16n/13}$	2^{2n}	$2^{12n/13}$	2^{3n}	2^{3n}	$2^{3n/13}$

Table 9. Space-efficient results on $3n$-to-n bit primitive (PuRF or single-key blockcipher) Knudsen-Preneel Compression Functions.

Code	$sn+mn \to sn$	Our Attack			KP Results		
		Query Complexity	Time Complexity	Memory Complexity	KP-Conj. Time	KP-Attack Time	KP-Attack Memory
$[r,k,d]_{2^e}$	$3kn \to rn$	$2^{rn/k}$			$2^{(d-1)n}$	Thm. 7	Thm. 7
$[4,2,3]_8$	$(4+2)n \to 4n$	2^{2n}	2^{2n}	2^{n}	2^{2n}	2^{2n}	2^{n}
$[6,4,3]_8$	$(6+6)n \to 6n$	$2^{3n/2}$	$2^{3n/2}$	$2^{n/2}$	2^{2n}	2^{2n}	$2^{n/2}$
$[9,7,3]_8$	$(9+12)n \to 9n$	$2^{9n/7}$	$2^{9n/7}$	$2^{4n/7}$	2^{2n}	2^{2n}	$2^{2n/7}$
$[5,2,4]_8$	$(5+1)n \to 5n$	$2^{5n/2}$	2^{3n}	$2^{3n/2}$	2^{3n}	2^{3n}	$2^{3n/2}$
$[7,4,4]_8$	$(7+5)n \to 7n$	$2^{7n/4}$	$2^{9n/4}$	$2^{3n/4}$	2^{3n}	2^{3n}	$2^{3n/4}$
$[10,7,4]_8$	$(10+11)n \to 10n$	$2^{10n/7}$	2^{2n}	$2^{6n/7}$	2^{3n}	2^{3n}	$2^{3n/7}$