The Grindahl hash functions

Lars R. Knudsen[1], Christian Rechberger[2], and Søren S. Thomsen[1] ⋆

[1] Technical University of Denmark, DK-2800 Kgs. Lyngby, Denmark
{lars@ramkilde.com,crypto@znoren.dk}
[2] Graz University of Technology, A-8010 Graz, Austria
christian.rechberger@iaik.tugraz.at

Abstract. In this paper we propose the Grindahl hash functions, which are based on components of the Rijndael algorithm. To make collision search sufficiently difficult, this design has the important feature that no low-weight characteristics form collisions, and at the same time it limits access to the state. We propose two concrete hash functions, Grindahl-256 and Grindahl-512 with claimed security levels with respect to collision, preimage and second preimage attacks of 2^{128} and 2^{256}, respectively. Both proposals have lower memory requirements than other hash functions at comparable speeds and security levels.

Keywords: hash functions, Rijndael, AES, design strategy, proposal

1 Introduction

As a result of a large number of attacks [5, 6, 11, 19, 42, 44–46] on hash functions such as MD5 [41] and SHA-1 [23] of the so-called MD4 family, and general attacks [30, 31, 33] on the typical construction method [18, 37], there is an increasing need for considering alternative construction methods and principles for future hash functions.

In this paper we develop an alternative design strategy for hash functions, and we propose a collection of hash functions named Grindahl[3]. We also propose the two instances Grindahl-256 and Grindahl-512. The main motivation is as follows. Among the many properties a cryptographic hash function is expected to have, collision resistance seems to be the hardest to achieve. Shortcut collision attacks faster than birthday attacks efficiently identify a subset of the message space with the property that a random pair of messages from the subset collide with probability higher than $2^{-n/2}$, where n is the output size of the hash function. A popular tool is differential cryptanalysis. In addition to restricting the pairs that are chosen from the subset of the message space to those having a certain difference, one might impose restrictions on the relations between actual values of message bits. Usually, the aim is to prevent differences from spreading uncontrollably. Most members of the MD4 family as well as recent proposals like FORK-256 [29] allow this kind of control.

Our aim is to obtain hash functions which do not allow this type of control by using building blocks of the block cipher Rijndael [16]. These well analysed and well understood building blocks are used to ensure that any difference introduced at the input needs to spread over large parts of the state before a collision is possible.

⋆ The first two authors have been supported by the European Commission through the IST Programme under Contract IST-2002-507932 ECRYPT. The third author is supported by the Danish Research Council for Technology and Production Sciences, grant no. 274-05-0151.

[3] To be pronounced 'grijndael'.
grind /graɪnd/: to break or crush sth into very small pieces [. . .] using a special machine.

While aiming for a security level up to the birthday bound for collision and (second) preimage attacks, we obtain speeds comparable to members of the SHA-2 family [24], but with a fundamentally different design and lower memory requirements for an implementation. Additionally, the computational overhead for small messages compares favourably to other design strategies.
Our design has a structure similar to some of those of J. Daemen (et al.) such as SubHash and StepRightUp [14], Panama [15], and the recent proposal RadioGatún [4]. The similarity lies in the way small pieces of message via a round function sequentially update a state in an invertible fashion.

Apart from being hash function proposals, we describe how our proposals can be turned into compression functions accepting only a fixed-size input. In order to test and develop cryptanalytic methods, variants with reduced cryptographic strength are helpful. For the MD4 family of hash functions, changing the number of rounds serves this purpose very well. Our design allows for such simple modifications and we encourage the reader to analyse such simpler variants. See Appendix B for suggestions.

2 The Grindahl design

In this section we present our proposal for a collection of hash functions. We start off with a description of the general design strategy which we call "Concatenate-Permute-Truncate". This design principle was first proposed by R. Merkle and used in his hash function Snefru [38], and it requires the existence of a non-linear permutation. We propose a highly parameterisable permutation hence in effect a collection of non-linear permutations, in Section 2.3. The general design and our proposed permutations together form the Grindahl hash functions. Concrete proposals are presented in Section 3.

2.1 General strategy

The proposal of this paper was conceived from the following general design strategy for an n-bit hash function. In the following, we denote by m the *state size*, and by b the *message size*. We require $m \geq n$ and $b > 0$. We denote by $\mathrm{trunc}_k(x)$ the least significant k bits of x. Let $P : \{0,1\}^{m+b} \rightarrow \{0,1\}^{m+b}$ be a non-linear permutation, and let s_0 be the *initial state* with $|s_0| = m$.

The principle behind our design is "Concatenate-Permute-Truncate"; let d be the message (appropriately padded) to be hashed, and split d into t blocks of b bits, i.e. $d = d_1 \| \cdots \| d_t$, $|d_i| = b$. Then for $0 < i \leq t$ do

$$S_i \leftarrow d_i \| s_{i-1} \qquad \text{(Concatenate)} \qquad (1)$$

$$\hat{S}_i \leftarrow P(S_i) \qquad \text{(Permute)} \qquad (2)$$

$$s_i \leftarrow \mathrm{trunc}_m(\hat{S}_i) \qquad \text{(Truncate)} \qquad (3)$$

Hence, a message block is concatenated with the state to form what we shall call the *extended state*, on which some permutation P is applied. Subsequently, the extended state is truncated down to the new state. The steps (1)–(3) form an *input round*.

We define an output transformation consisting of *blank rounds* and a truncation step at the end. Blank rounds are defined as follows. For $t < i \leq t + \nu_{\mathrm{br}}$, $\nu_{\mathrm{br}} \geq 0$, do

$$\hat{S}_i \leftarrow P(\hat{S}_{i-1}) \,.$$

Hence, the blank rounds work only on the extended state, which means that in the processing of the final message block d_t, (3) above can be omitted. Finally, the output of the hash function is $\mathrm{trunc}_n(\hat{S}_{t+\nu_{\mathrm{br}}})$.

2.2 Invertibility

Assuming that the permutation P is invertible, the hash function is not one-way in the sense that for a given output, some initial state and a message producing that output can be easily found. However, this does not directly give rise to proper (second) preimage attacks. If P has sufficient cryptographic properties then an attacker will have no control over the initial state obtained.

The success probability of meet-in-the-middle attacks is affected in part by the value of m above. If no weaknesses of P are exploited then internal collision attacks (collisions before the blank rounds) and meet-in-the-middle attacks have complexity $2^{m/2}$. If one requires that no (second) preimage attacks better than a brute force search exist, then one has to choose $m \geq 2n$.

2.3 Design approach for the permutation

A well-known family of permutations is the block cipher algorithm Rijndael [16], a subset of which was adopted as the Advanced Encryption Standard (AES) by the US government in 2001 [25]. What follows is an approach that uses the design principle of Rijndael to build a permutation to be used in our hash proposal. This bears some resemblance to the leak-extraction method of the stream cipher LEX [8] or the message authentication code framework ALRED [17].

We operate with a more general description of the algorithm than Rijndael itself. As in Rijndael, we view the (in our case extended) state as a matrix of bytes, although here the extended state is a matrix α of ν_{rw} rows and ν_{cl} columns of bytes. The entries of α are denoted by $\alpha_{i,j}$, meaning the entry in row i, column j (numbering starts from zero, and indices are always to be reduced modulo ν_{rw} and ν_{cl}, respectively). Hence, the extended state is the matrix

$$\alpha = \begin{bmatrix} \alpha_{0,0} & \alpha_{0,1} & \cdots & \alpha_{0,\nu_{\mathrm{cl}}-1} \\ \alpha_{1,0} & \alpha_{1,1} & \cdots & \alpha_{1,\nu_{\mathrm{cl}}-1} \\ \vdots & \vdots & \ddots & \vdots \\ \alpha_{\nu_{\mathrm{rw}}-1,0} & \alpha_{\nu_{\mathrm{rw}}-1,1} & \cdots & \alpha_{\nu_{\mathrm{rw}}-1,\nu_{\mathrm{cl}}-1} \end{bmatrix}.$$

We assume that b is a multiple of 8, and we define $\nu_{\mathrm{mb}} = b/8$ as the number of bytes in a message block. Hence, according to (3) and (1), in the process of truncation followed by concatenation (with the following message block), ν_{mb} extended state bytes are overwritten. These extended state bytes do not have to be computed in all except the last input round.

The reader is expected to be familiar with the transformations defined in the Rijndael specification [16]. Here we adopt the new names introduced in the actual standard [25], i.e. SubBytes, ShiftRows, MixColumns and AddRoundKey.
We do not use AddRoundKey directly in this design. Instead, we introduce a related transformation, AddConstant. We now comment on the four transformations used in the Grindahl design. See also Fig. 1.

SubBytes. The non-linear substitution function SubBytes is defined exactly as in the Rijndael specification, i.e. we use the same S-box.

ShiftRows. The ShiftRows transformation cyclically shifts bytes a number of positions along each row. We introduce the *rotation constants* as the ν_{rw}-tuple (ρ_0, ρ_1, …, $\rho_{\nu_{\mathrm{rw}}-1}$) of integers, $0 \leq \rho_i < \nu_{\mathrm{cl}}$, with the meaning that in row i bytes should be cyclically shifted ρ_i positions to the right. (Hence, with this definition the rotation constants of Rijndael are $(0, 3, 2, 1)$).

MixColumns. The transformation MixColumns is defined as in the Rijndael specification whenever $\nu_{\mathrm{rw}} = 4$. For other values of ν_{rw}, the transformation must be

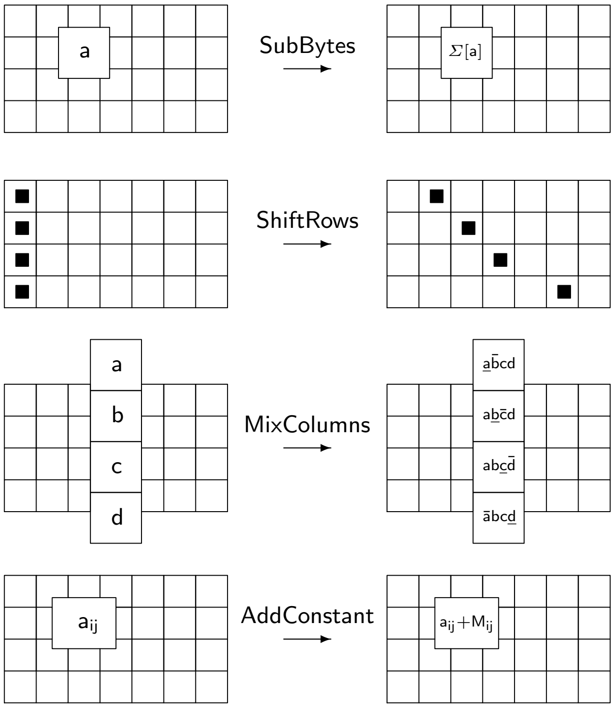

Fig. 1: The four transformations on the extended state. Here an example with an extended state of 4 rows and 7 columns. Σ is the S-box, and example rotation constants are $(1, 2, 3, 5)$.

redefined. The important property of maximal difference propagation should be maintained: when a difference is introduced to $k > 0$ bytes in a column before MixColumns, the effect should be that after MixColumns at least $\nu_{\mathrm{rw}} - k + 1$ bytes have changed.

AddConstant. As mentioned we replace the AddRoundKey transformation known from the Rijndael design by AddConstant, which introduces asymmetry to each round: Let α be the extended state matrix, and let M be some matrix of the same size. Then define AddConstant as

$$\mathsf{AddConstant}(\alpha) = M \oplus \alpha.$$

We note that if an extended state consists of ν_{cl} equal columns, then by defining e.g., $M = 0$ the extended state will still consist of ν_{cl} equal columns after one round. By flipping some bits of the extended state we can circumvent this property.

The four transformations operate on a matrix of bytes. However, given an invertible mapping from an extended state to a bit string, we may also apply them on bit strings. The mapping from a bit string to an extended state matrix is done as follows. Let x be an $(8\nu_{\mathrm{rw}}\nu_{\mathrm{cl}})$-bit string. Map this into an extended state α by splitting x into $\nu_{\mathrm{rw}}\nu_{\mathrm{cl}}$ 8-bit chunks $x_0, \ldots, x_{\nu_{\mathrm{rw}}\nu_{\mathrm{cl}}-1}$, and then let $\alpha_{i,j} = x_{i+\nu_{\mathrm{rw}}j}$, $0 \le i < \nu_{\mathrm{rw}}$ and $0 \le j < \nu_{\mathrm{cl}}$. This mapping has a natural inverse.

Given appropriate definitions of the four transformations we define the permutation P as

$$P(\alpha) = \mathsf{MixColumns} \circ \mathsf{ShiftRows} \circ \mathsf{SubBytes} \circ \mathsf{AddConstant}(\alpha).$$

2.4 Birthday attacks

If P were an ideal permutation, then internal collisions and (second) preimages would have complexity $2^{m/2}$. However, P is obviously not ideal. In fact, it is easy to see that the complexity of e.g. an internal collision attack for this choice of P is at most $2^{(m-b)/2}$: assume that the first column of the matrix is overwritten by the message input. Compute the extended state before the blank rounds of a number of different messages. Now append two constant blocks to all messages. The first constant message block overwrites the first column of the extended state. Then the permutation is applied, where ShiftRows moves ν_{rw} bytes into the first column of the extended state, and subsequently MixColumns mixes the bytes in the column. The second constant message block overwrites this column. This means that if two extended states agree on all bytes except the first column and the bytes that are moved into the first column by the first constant message block, then the two extended states will agree on all bytes after the second constant message block. The expected number of messages needed for this attack to succeed is $2^{(m-b)/2}$. In fact this approach can be generalised to every way of mapping an input message to the extended state.

Since the permutation is invertible, the entire hash function is invertible, and hence meet-in-the-middle attacks can be applied: Compute the intermediate (extended) state for $2^{(m-b)/2}$ different messages all having the same last two blocks. Given a target image of the hash function, compute in the backward direction the intermediate state for the same number of message suffices. With good probability, there is a match between the two sets of intermediate states. This yields a (second) preimage with complexity about $2^{(m-b)/2}$.

2.5 Design parameters for the permutation

We now present some considerations with respect to the design parameters introduced above.

Rotation constants. The rotation constants used in ShiftRows should be chosen carefully. Most importantly, the rotation constants should ensure that the entire state depends on the message input as quickly as possible when considering that the first ν_{mb} bytes of the state are overwritten by message input in every round. This means, for instance, that ρ_i, $0 \le i < \nu_{\mathrm{mb}}$, cannot be 0, and that $\rho_i \neq \rho_j$ whenever $i \neq j$.

Several tuples of rotation constants ensure full diffusion after the same (minimum) number of rounds μ. However, of these some are better than others in the sense that a larger part of the state depends on every message byte after $\mu - 1$ rounds, after $\mu - 2$ rounds etc. In Appendix C we suggest a method for choosing rotation constants given a particular geometry of the extended state.

State geometry. Parameters that affect the size m of the state should be chosen with Section 2.4 in mind. Usually designers of hash functions aim for full (second) preimage resistance, meaning these have expected complexity 2^n. We have chosen to accept a lower complexity, since it is already required that n is large enough to resist birthday collision attacks, which have complexity $2^{n/2}$. Other hash functions have been proposed and standardised, for which (second) preimage resistance is lower than for an ideal hash function, e.g., the MDC-2 construction [39] which has preimage complexity at most $2^{3n/4}$ [40]. Appendix A discusses known generic attacks that depend on the state size. By adjusting the state geometry accordingly, resistance against these attacks can be achieved conveniently.

Choices for ν_{rw} and ν_{cl} are a trade-off between two distinct properties. If the two numbers are about the same, diffusion happens faster than with more columns than rows. On the other hand, with a wider state and only a single column being used for message input, the birthday attack as described above has a higher complexity, and hence the extended state need only be slightly larger than the output. For implementation purposes it makes sense to choose ν_{rw} to be a multiple of 4, since then on a 32-bit machine SubBytes and MixColumns can be performed in one by table lookups.

2.6 Design parameters for the output transformation

The number ν_{br} of blank rounds in the output transformation should be chosen such that the last message block affects all output bytes.

It might be desirable that the output transformation have the property that when given only black-box access to it, one is unable to distinguish the output transformation from a pseudo-random function. It is unclear how useful this property is in practice, since the output transformation is invertible (by guessing the discarded state bytes), but it would seem to complicate attempts at finding external collisions.

With ν_{br} blank rounds, the actual number of invocations of P after the last message block is concatenated with the state is $\nu_{\mathrm{br}} + 1$. During these rounds, no extended state bytes are overwritten by message input. Hence, if the requirement above is fulfilled after μ rounds, then one should choose $\nu_{\mathrm{br}} \geq \mu - 1$. Suggestions for actual values for ν_{br} are given in Sections 3.1 and 3.2.

3 Proposals for hash functions

We propose concrete instantiations of the design strategy presented in Section 2. Hash functions with 256-bit and with 512-bit output size are given in Sections 3.1 and 3.2, respectively. For both proposals, the initial state is the all-zero state, and padding is performed as described in Section 3.3. Additionally we give a preliminary security analysis in Section 3.4.

In the following, constant bytes (or elements of $\mathbb{F}_{256}$) are written in hexadecimal using this font, e.g., c5.

3.1 Grindahl-256

Grindahl-256 is defined as follows. Let the parameters have the following values: $\nu_{\mathrm{rw}} = 4$, $\nu_{\mathrm{cl}} = 13$, $\nu_{\mathrm{mb}} = 4$, $\nu_{\mathrm{br}} = 8$. Hence, the extended state has 4 rows and 13 columns, and the message input is 32 bits in size. In the truncation/concatenation process these overwrite the contents of the first column of the extended state. The proposal has 8 blank rounds in the output transformation.

The rotation constants used in the ShiftRows transformation are $(1, 2, 4, 10)$, chosen by the method described in Appendix C. For these rotation constants every message byte affects the entire extended state after four rounds. SubBytes and MixColumns are defined as in Rijndael, and AddConstant is defined simply as

$$\alpha_{3,12} \leftarrow \alpha_{3,12} \oplus \mathsf{01}.$$

Security claim. We claim that the effort to find collisions, second preimages and preimages is of the order of 2^{128} iterations.

3.2 Grindahl-512

We also propose the 512-bit hash function Grindahl-512. For this variant, we propose the following parameter values: $\nu_{\mathrm{rw}} = 8$, $\nu_{\mathrm{cl}} = 13$, $\nu_{\mathrm{mb}} = 8$, $\nu_{\mathrm{br}} = 8$. In other words, the state has 8 rows and 13 columns, and the message input is 64 bits in size, corresponding to the first column of the extended state. The output transformation contains 8 blank rounds.

The proposed rotation constants for ShiftRows are $(1, 2, 3, 4, 5, 6, 7, 8)$, which cause full diffusion after three rounds (these were chosen according to Appendix C). MixColumns has to be redefined since the extended state contains 8 rows. We define this transformation as

$$\mathsf{MixColumns}(\alpha) = A \cdot \alpha,$$

with

$$A = \begin{bmatrix} \mathsf{02} & \mathsf{0c} & \mathsf{06} & \mathsf{08} & \mathsf{01} & \mathsf{04} & \mathsf{01} & \mathsf{01} \\ \mathsf{01} & \mathsf{02} & \mathsf{0c} & \mathsf{06} & \mathsf{08} & \mathsf{01} & \mathsf{04} & \mathsf{01} \\ \mathsf{01} & \mathsf{01} & \mathsf{02} & \mathsf{0c} & \mathsf{06} & \mathsf{08} & \mathsf{01} & \mathsf{04} \\ \mathsf{04} & \mathsf{01} & \mathsf{01} & \mathsf{02} & \mathsf{0c} & \mathsf{06} & \mathsf{08} & \mathsf{01} \\ \mathsf{01} & \mathsf{04} & \mathsf{01} & \mathsf{01} & \mathsf{02} & \mathsf{0c} & \mathsf{06} & \mathsf{08} \\ \mathsf{08} & \mathsf{01} & \mathsf{04} & \mathsf{01} & \mathsf{01} & \mathsf{02} & \mathsf{0c} & \mathsf{06} \\ \mathsf{06} & \mathsf{08} & \mathsf{01} & \mathsf{04} & \mathsf{01} & \mathsf{01} & \mathsf{02} & \mathsf{0c} \\ \mathsf{0c} & \mathsf{06} & \mathsf{08} & \mathsf{01} & \mathsf{04} & \mathsf{01} & \mathsf{01} & \mathsf{02} \end{bmatrix}.$$

In this product, bytes are to be considered elements of the field $\mathbb{F}_{256}$, which is defined as in Rijndael. This transformation ensures maximal difference propagation, since the error-correcting code over $\mathbb{F}_{256}$ with generator matrix $\left[I \ A^{\mathrm{T}}\right]$ is MDS (see e.g., [35]). SubBytes is defined as in Rijndael, and AddConstant is defined as

$$\alpha_{7,12} \leftarrow \alpha_{7,12} \oplus \mathsf{01}.$$

Security claim. We claim that the effort to find collisions, second preimages and preimages is of the order of 2^{256} iterations.

3.3 Padding rule

Padding for both proposals is performed as follows. Append a '1'-bit to the message, and then a number of '0'-bits to fill the last message block. Finally, append a 64-bit representation of the number of message blocks in the padded message. This means that both hash functions can digest messages of size at most $2^{64} - 1$ message blocks including the padding itself. (Note the difference between this and most existing padding rules in that the number of message *blocks* is appended rather than the number of message *bits*.)

3.4 Security analysis

We claim the same security level against collision, preimage and second preimage attacks. We do not know of any (second) preimage attacks with complexity as low as the birthday bound, and we expect the actual security level against these attacks to be higher. However, here we focus on collision resistance.

Collision resistance. Collisions can either be internal or external collisions. In the case of external collisions we believe the number of blank rounds is high enough to rule out any differential with a low number of active bytes. For internal collisions, the complete state which is considerably larger than the output size needs to collide for different input messages. Additionally, we give more evidence that shortcut collision finding methods are unlikely. For Grindahl-256, we give a lower bound for the number of rounds to arrive at an internal collision.

Exhaustive search through all possible input difference patterns and difference propagation patterns shows that Grindahl-256 has the following property.

Property 1 *An internal collision for Grindahl-256 requires at least 6 input rounds. Moreover, any characteristic starting or ending in the extended state with no difference contains at least one round where at least half the extended state bytes (excluding the first column) are active.*

A characteristic leading from a zero-difference state to a collision via different message inputs is given in Appendix D. This characteristic spans 6 rounds, and no shorter characteristic was found in an exhaustive search.

We cannot completely rule out the possibility that high probability characteristics exist, but we believe that a classical differential attack will not be successful.

For Grindahl-512, collisions after 4 input rounds cannot be ruled out with this method. However, due to efficient mixing of the used building blocks, we believe that no low-weight differentials lead to an internal collision.

It remains to be seen if and how methods that improved collision search methods for other hash functions can be adapted to analyse the Grindahl design. Recently developed candidate techniques are message modification as introduced in the cryptanalysis of several members of the MD4 family [44–46], neutral bits as used to speed up collision search for SHA-0 [5], internal meet-in-the-middle techniques as used in the analysis of Tiger [32] or the greedy-like approach as pursued in the analysis of SHA-1 in [10]. They all have in common that they exploit the attacker's knowledge of all intermediate states to improve the effectiveness of traditional differential cryptanalysis. We argue that the building blocks used and the limited direct access to the internal state in the Grindahl design make these techniques difficult to apply efficiently.

A potential attack method. An anonymous reviewer proposed such a method tailored to the Grindahl design. In the following, we give a brief sketch of the proposed method. First, a chain of differentials in which in every round the number of active bytes is low must be found. Given such a differential chain, the attack is launched as follows. In the attack, we do not care about actual differences; we only care about whether there is a difference or not.

For a column containing active bytes to satisfy the differential through the MixColumns transformation, some linear constraints must be satisfied. If the actual differences do not matter, these are the only constraints for a full round. Additionally, the fact that new input bytes do not affect some parts of the state for a limited number of rounds can be exploited. This means that often a (small) number of active bytes can be arbitrary (we call these "neutral bytes"), and the remaining ones are fixed ("control bytes"). A neutral byte may later be used as a control byte to ensure that a characteristic is followed. Given that every round introduces up to four neutral bytes, this attack may or may not be applicable. It seems hard to make use of a neutral byte that has been through several S-boxes, since (1) the S-box is non-linear, and so one cannot deterministically select a given output difference by a given input difference, and (2) after a number of S-boxes a message byte affects a large part of the state, and hence it becomes less useful as a control byte. We leave it as an open problem to investigate the feasibility of this attack.

4 Designing secure compression functions

Any proposal following the Grindahl design strategy of hash functions accepting variable length inputs can easily be turned into a compression function for fixed length inputs. Let H be an instance of the Grindahl hash functions with initial state s_0, where padding is omitted. Hence, H accepts only messages of size an integer multiple of b in bits. Let the output size of H be n. Based on H, define a compression function $h : \{0,1\}^{tb} \rightarrow \{0,1\}^n$, where t is an integer greater than n/b, as $h(x) = H(x)$.

Note that here, the compression function is defined as taking only one input, whereas usually one thinks of a compression function as taking two inputs, a chaining value and a message block. We think it makes good sense to treat the two inputs as one: in most modes of operation of the compression function an attacker is expected to have the same control over the chaining value as over the message. In practice we would suggest that the two inputs are simply concatenated, and hence if we instead describe the compression function as $h : \{0,1\}^n \times \{0,1\}^{tb-n} \rightarrow \{0,1\}^n$, then we would define h as $h(c, x) = H(c\|x)$.

The choice of t implies a certain trade-off between speed and security. By decreasing t security is increased, but speed is reduced since in effect the rate of blank rounds to input rounds increases.

If an additional input (e.g., a salt, a key or a counter) to the compression function is required, then we suggest that this is prepended to the chaining/message input. This way, many of the newly proposed modes of operation which turn a secure compression function into a secure hash function are directly applicable to our proposal. We now describe instances of our design strategy acting as a compression function to be used with modes like Merkle-Damgård [18, 37], EMD [2], randomized hashing [28], HAIFA [7] etc.

Compression function mode for concrete proposals. We suggest compression function modes for both Grindahl-256 and Grindahl-512 by letting t above be equal to $40+s$. Here s denotes the number of input blocks used for an additional input, with the possibility of $s = 0$. Hence, the compression functions both take $(40+s)\nu_{\mathrm{mb}}$-byte inputs. This corresponds to $1280+32s$ bits and $2560+64s$ bits for Grindahl-256 and Grindahl-512, respectively. Of these, respectively 1024 and 2048 bits form the message input, and the rest is reserved for chaining input and, if applicable, a salt/key/counter.

5 Implementation

Implementations of Grindahl can directly inherit most of the extensive research done to optimise implementations of the AES block cipher on different platforms. Also side-channel attacks might be an issue if a hash function is used to process secret key material, be it as a key-derivation-function (KDF) or as a hash-based message authentication code (MAC). Here we refer to extensive work done to protect implementations of the AES against these kinds of attacks, e.g. a very fast bit-sliced AES implementation immune to timing attacks [36].

5.1 Software performance

One usually defines the rate of a hash function based on a block cipher as the number of blocks that are processed for each block encryption. We may do the same here, except that we have to take into account the extended state size, and the fact that ν_{mb} bytes are processed per round, whereas in AES-128, 16 bytes are processed for every 10 rounds. Hence, the rate of a Grindahl instance is $\frac{10\nu_{\text{mb}}}{\nu_{\text{rw}}\nu_{\text{cl}}-\nu_{\text{mb}}}$. Here we also take into account that the ν_{mb} bytes that are overwritten by the following message block do not have to be computed. As an example, the rate of both Grindahl-256 and Grindahl-512 is 5/6.

In an optimised software implementation of Grindahl-n on a 32-bit platform, $n/64$ tables of 256 32-bit words are needed. In the discussion (Section 5.4) on memory requirements this is not taken into account as the need for these tables is a consequence of the optimisation only.

Implementation report. Grindahl-256 has been implemented in C on a 32-bit Pentium 4 processor. It runs at about 32 cycles/byte. This might be compared with the Rijndael-128 implementation from the Crypto++ [13] package, which, according to the website, performs at about 33 cycles/byte on a Pentium 4. As expected, performance is similar. For additional comparison, on the same platform SHA-256 is benchmarked at about 45 cycles/byte [13]. As with Rijndael, we expect that hand-optimised implementations will improve performance by a factor up to about 2. Grindahl-512 is more suited for 64-bit architectures, and here we expect its speed to be similar to Grindahl-256 on a 32-bit architecture.

Program code and test vectors. The interested reader may find C implementations of and test vectors for Grindahl-256 and Grindahl-512 at [27].

5.2 On hardware implementations

For passively powered designs, the number of active registers/logic per clock cycle should be small. In contrast to the MD4 family, the Grindahl hash function design allows for hardware designs with a small data path width without penalties. Also, compared to the MD4 family and other proposals a smaller number of registers is needed in the Grindahl design, which allows for low-cost and low-power implementations. In addition, various trade-offs towards high speed are possible, utilising the many implementation options already pioneered for the AES and benefiting from the smaller number of needed registers.

Regarding low-cost hardware implementations, based on [21] we estimate area requirements to about 5-6,000 gate equivalents for Grindahl-256. This compares favourably with the smallest known SHA-256 implementation [20], which requires more than 10,000 gate equivalents.

5.3 On hashing small messages

The blank rounds add fixed costs even for very small messages. However, in absolute terms, for both Grindahl-256 and Grindahl-512 it is equivalent to only 32 bytes of additional padding. Note that members of the SHA-2 family operate on input blocks of size 64 or 128 bytes. Hence, given the much smaller input block size of 4 (or 8) bytes, small messages are still handled more efficiently.

5.4 Memory requirements

Due to the small message blocks, the needed working memory for Grindahl-256 and Grindahl-512 is small. This certainly suits implementations in constrained environments. It is interesting to note that implementations of the Grindahl strategy need only $b + m$ bits of working memory. Implementations of the MD4 family, on the other hand, require $b + 2m$ bits of memory, where usually $b = 512$ and m is

128-bit security		256-bit security	
name	memory	name	memory
Grindahl-256	416	Grindahl-512	832
SHA-256	1024	SHA-512	2048
RadioGatún-14	812	RadioGatún-27	1566
FORK-256	1280	Whirlpool	1536
LASH-256	1536	LASH-512	3072

Table 1: Needed working memory in bits for different hash functions.

the same as the output size. The reason for the difference is the feed-forward of the initial state which counters meet-in-the-middle attacks. However, even if this feed-forward operation would be omitted, the general picture would not change.

We see the low memory requirements as an important feature of our proposal, and in Table 1 we give a comparison with other hash functions [1, 3, 4, 24, 29] claiming a comparable security level against collision search attacks. Note that we do not consider memory needed to store temporary variables or constants.

6 Conclusion

We proposed the Grindahl hash functions which overcome several identified weaknesses of the commonly used MD4 family of hash functions. The identified weaknesses are addressed on two layers. By using the well analysed building blocks of Rijndael we obtain a design for which we claim that no efficient differential collision structures exist. In addition, we limit access to the internal state with the idea to thwart advanced techniques that improve effectiveness for collision search methods.

We proposed two instantiations of the Grindahl hash collection, Grindahl-256 and Grindahl-512. The claimed security level with respect to collision, preimage and second preimage attacks is 2^{128} and 2^{256}, respectively. Grindahl-256 performs at about the same rate as AES-128 without the key schedule, and on a 64-bit platform we expect that implementations of Grindahl-512 can achieve similar speeds.

Other intriguing implementation aspects of the proposals are very low working memory requirements which aid implementations in constrained environments, as well as the efficient handling of small messages.

Acknowledgments. The authors would like to thank Charanjit Jutla and the anonymous reviewers for many helpful comments.

References

1. P. S. L. M. Barreto and V. Rijmen. The Whirlpool hashing function. Available at `https://www.cosic.esat.kuleuven.be/nessie/tweaks.html`, May 2003.
2. M. Bellare and T. Ristenpart. Multi-Property-Preserving Hash Domain Extension and the EMD Transform. In Lai and Chen [34], pages 299–314.
3. K. Bentahar, D. Page, M.-J. O. Saarinen, J. H. Silverman, and N. Smart. LASH. Presented at Second Cryptographic Hash Workshop, Santa Barbara, August 24-25, 2006.
4. G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche. RadioGatún, a belt-and-mill hash function. Presented at Second Cryptographic Hash Workshop, Santa Barbara, August 24-25, 2006. See `http://radiogatun.noekeon.org/`.
5. E. Biham and R. Chen. Near-Collisions of SHA-0. In Franklin [26], pages 290–305.

6. E. Biham, R. Chen, A. Joux, P. Carribault, C. Lemuet, and W. Jalby. Collisions of SHA-0 and Reduced SHA-1. In Cramer [12], pages 36–57.
7. E. Biham and O. Dunkelman. A Framework for Iterative Hash Functions – HAIFA. Presented at Second Cryptographic Hash Workshop, Santa Barbara, August 24-25, 2006.
8. A. Biryukov. The Design of a Stream Cipher LEX. In *Selected Areas in Cryptography*, Lecture Notes in Computer Science. Springer, 2006. To appear.
9. G. Brassard, editor. *Advances in Cryptology - CRYPTO '89, 9th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 1989, Proceedings*, volume 435 of *Lecture Notes in Computer Science*. Springer, 1990.
10. C. D. Cannière and C. Rechberger. Finding SHA-1 Characteristics: General Results and Applications. In Lai and Chen [34], pages 1–20.
11. F. Chabaud and A. Joux. Differential Collisions in SHA-0. In H. Krawczyk, editor, *CRYPTO*, volume 1462 of *Lecture Notes in Computer Science*, pages 56–71. Springer, 1998.
12. R. Cramer, editor. *Advances in Cryptology - EUROCRYPT 2005, 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Aarhus, Denmark, May 22-26, 2005, Proceedings*, volume 3494 of *Lecture Notes in Computer Science*. Springer, 2005.
13. The Crypto++ website. `http://www.cryptopp.com/`, 2007.
14. J. Daemen. *Cipher and hash function design strategies based on linear and differential cryptanalysis*. PhD thesis, Katholieke Universiteit Leuven, March 1995.
15. J. Daemen and C. S. K. Clapp. Fast Hashing and Stream Encryption with PANAMA. In S. Vaudenay, editor, *Fast Software Encryption*, volume 1372 of *Lecture Notes in Computer Science*, pages 60–74. Springer, 1998.
16. J. Daemen and V. Rijmen. The Block Cipher Rijndael. In J.-J. Quisquater and B. Schneier, editors, *CARDIS*, volume 1820 of *Lecture Notes in Computer Science*, pages 277–284. Springer, 1998.
17. J. Daemen and V. Rijmen. A New MAC Construction ALRED and a Specific Instance ALPHA-MAC. In H. Gilbert and H. Handschuh, editors, *Fast Software Encryption*, volume 3557 of *Lecture Notes in Computer Science*, pages 1–17. Springer, 2005.
18. I. Damgård. A Design Principle for Hash Functions. In Brassard [9], pages 416–427.
19. H. Dobbertin. Cryptanalysis of MD4. *Journal of Cryptology*, 11(4):253–271, 1998.
20. M. Feldhofer and C. Rechberger. A Case Against Currently Used Hash Functions in RFID Protocols. Presented at the Workshop on RFID Security 2006.
21. M. Feldhofer, J. Wolkerstorfer, and V. Rijmen. AES Implementation on a Grain of Sand. *IEE Proceedings on Information Security*, 152(1):13–20, October 2005.
22. N. Ferguson and B. Schneier. *Practical Cryptography*. Wiley Publishing, 2003.
23. FIPS 180-1, Secure Hash Standard. Federal Information Processing Standards Publication 180-1, U.S. Department of Commerce/NIST, National Technical Information Service, Springfield, Virginia, April 1995. Supersedes FIPS 180.
24. FIPS 180-2, Secure Hash Standard. Federal Information Processing Standards Publication 180-2, U.S. Department of Commerce/NIST, National Technical Information Service, Springfield, Virginia, August 2002. Supersedes FIPS 180 and FIPS 180-1.
25. FIPS 197, Advanced Encryption Standard (AES). Federal Information Processing Standards Publication 197, U.S. Department of Commerce/NIST, National Technical Information Service, Springfield, Virginia, November 2001.
26. M. K. Franklin, editor. *Advances in Cryptology - CRYPTO 2004, 24th Annual International CryptologyConference, Santa Barbara, California, USA, August 15-19, 2004, Proceedings*, volume 3152 of *Lecture Notes in Computer Science*. Springer, 2004.
27. The Grindahl web page. `http://www.ramkilde.com/grindahl`, 2007.
28. S. Halevi and H. Krawczyk. Strengthening Digital Signatures via Randomized Hashing. In C. Dwork, editor, *CRYPTO*, volume 4117 of *Lecture Notes in Computer Science*, pages 41–59. Springer, 2006.
29. D. Hong, D. Chang, J. Sung, S. Lee, S. Hong, J. Lee, D. Moon, and S. Chee. A New Dedicated 256-Bit Hash Function: FORK-256. In M. Robshaw, editor, *Fast Software Encryption*, volume 4047 of *Lecture Notes in Computer Science*, pages 195–209. Springer, 2006.

30. A. Joux. Multicollisions in Iterated Hash Functions. Application to Cascaded Constructions. In Franklin [26], pages 306–316.
31. J. Kelsey and T. Kohno. Herding Hash Functions and the Nostradamus Attack. In S. Vaudenay, editor, *EUROCRYPT*, volume 4004 of *Lecture Notes in Computer Science*, pages 183–200. Springer, 2006.
32. J. Kelsey and S. Lucks. Collisions and Near-Collisions for Reduced-Round Tiger. In Robshaw [43], pages 111–125.
33. J. Kelsey and B. Schneier. Second Preimages on n-bit Hash Functions for Much Less than 2^n Work. In Cramer [12], pages 474–490.
34. X. Lai and K. Chen, editors. *Advances in Cryptology – ASIACRYPT 2006, the 12th International Conference on the Theory and Application of Cryptology and Information Security, Shanghai, China, December 3-7, 2006, Proceedings*, volume 4284 of *Lecture Notes in Computer Science*. Springer, 2006.
35. F. J. MacWilliams and N. J. A. Sloane. *The Theory of Error-Correcting Codes*. North-Holland Mathematical Library, 1977.
36. M. Matsui. How Far Can We Go on the x64 Processors? In Robshaw [43], pages 341–358.
37. R. C. Merkle. One Way Hash Functions and DES. In Brassard [9], pages 428–446.
38. R. C. Merkle. A Fast Software One-Way Hash Function. *Journal of Cryptology*, 3(1):43–58, 1990.
39. C. H. Meyer and M. Schilling. Secure program load with Manipulation Detection Code. In *Proceedings of the 6th Worldwide Congress on Computer and Communications Security and Protection (SECURICOM'88)*, pages 111–130, 1988.
40. B. Preneel. *Analysis and Design of Cryptographic Hash Functions*. PhD thesis, Katholieke Universiteit Leuven, January 1993.
41. RFC 1321, The MD5 Message-Digest Algorithm. Internet Request for Comments 1321, R. Rivest, April 1992.
42. V. Rijmen and E. Oswald. Update on SHA-1. In A. Menezes, editor, *CT-RSA*, volume 3376 of *Lecture Notes in Computer Science*, pages 58–71. Springer, 2005.
43. M. J. B. Robshaw, editor. *Fast Software Encryption, 13th International Workshop, FSE 2006, Graz, Austria, March 15-17, 2006, Revised Selected Papers*, volume 4047 of *Lecture Notes in Computer Science*. Springer, 2006.
44. X. Wang, X. Lai, D. Feng, H. Chen, and X. Yu. Cryptanalysis of the Hash Functions MD4 and RIPEMD. In Cramer [12], pages 1–18.
45. X. Wang, Y. L. Yin, and H. Yu. Finding Collisions in the Full SHA-1. In V. Shoup, editor, *CRYPTO*, volume 3621 of *Lecture Notes in Computer Science*, pages 17–36. Springer, 2005.
46. X. Wang and H. Yu. How to Break MD5 and Other Hash Functions. In Cramer [12], pages 19–35.

A Resistance of Concatenate-Permute-Truncate to known attacks

The well-known Merkle-Damgård construction [18, 37] is a construction method for hash functions based on an underlying compression function $f : \{0,1\}^n \times \{0,1\}^b \to \{0,1\}^n$. The output size of the hash function is n. The main differences in the "Concatenate-Permute-Truncate" design are

- that we allow $m > n$, and hence attacking the internals of the hash function may be different (and hopefully harder) than attacking the hash function itself, and
- that we allow an output transformation, the blank rounds.

If $m = n$ and $\nu_{\mathrm{br}} = 0$, then the design can be seen as a special case of the Merkle-Damgård construction: we iterate "as usual" over a compression function of a certain kind. However, the possibility of varying m and ν_{br} makes it possible to obtain different properties for our design than the properties of hash functions based on the Merkle-Damgård construction.

In the following we describe some known attacks on the Merkle-Damgård construction, and we argue whether or not our design protects against these attacks, and, if applicable, how resistance to these attacks depends on the parameters n, m and ν_{br}. The analysis will assume that P is a black-box permutation, i.e. that the internals of P are unknown.

A.1 The length-extension attack

Let H be a hash function based on the Merkle-Damgård construction. Then H is susceptible to the so-called length-extension attack [22]. Given a collision, i.e. two messages d and d', with $|d| = |d'|$, such that $H(d) = H(d')$, then for any suffix x it holds also that $H(d\|x) = H(d'\|x)$.

This attack can be applied to the design described above only if the collision occurs before the blank rounds. If $m > n$ and we only consider the birthday attack, then an internal collision is harder to find than a collision for the full hash function.

A related property of the Merkle-Damgård construction is that if the length of an unknown message d, and hence the padding p of d, is known, and also $H(d)$ is known, then $H(d\|p\|x)$ can be computed for any suffix x. This attack is particularly a threat in some schemes that use a hash function for message authentication.

The attack can be mounted on a hash function following our design only if the attacker correctly guesses the $b+m-n$ bits that are truncated away. If he does so, he can go backwards through the blank rounds and obtain the extended state after the processing of the last message block.

A.2 Multi-collisions

An efficient method for constructing multi-collisions was described by A. Joux in 2004 [30]. A multi-collision is a set of (at least two) messages that all have the same hash. The attack of Joux has complexity $t2^{n/2}$ to find a 2^t-way multi-collision for an n-bit hash function in the Merkle-Damgård construction. In our design, the complexity would be $t2^{m/2}$. This is to be compared with a brute-force search for multi-collisions, which for even a modest t gets very close to 2^n. Hence, if one wants full resistance against the Joux multi-collision attack, one should choose $m \geq 2n$.

A.3 The herding attack

The herding attack [31] by J. Kelsey and T. Kohno shows another slight weakness of the Merkle-Damgård construction. Here, a binary tree of collisions is used to form a hash result h, which an attacker publishes. Subsequently he chooses a message, finds a message linking to one of the leaves of the binary tree, and then he has a complete, partially chosen message d with the hash h.

The complexity of the simplest version of this attack in the Merkle-Damgård construction is about $2^{(2n-5)/3}$. In our design, n can be replaced by m, and so with $m \geq (3n+5)/2$, the complexity of the attack is the same as for a preimage, which is what one would expect from a random hash function.

There is another important version of the attack for which the complexity decreases with increased size of the message d. If the size of d is about 2^r, then the complexity is about $2^{(2n-5)/3-r}$. Hence, for a given upper limit on the size of messages, m must be chosen accordingly if one wants to completely protect against this kind of attack.

A.4 Second preimage attack

There is a second preimage attack [33] by J. Kelsey and B. Schneier on the Merkle-Damgård construction. This attack requires finding an expandable message, meaning a set of messages of varying sizes such that all these messages collide internally in the hash function, given some fixed initial value. To find this expandable message one either makes use of fixed points or the ability to find (internal) collisions between a one-block message and a t-block message for varying values of t. Finally, the complexity of the attack depends on the complexity of finding the expandable message, and on the length of the target message, i.e. the message for which one tries to find a second message with the same hash. This complexity amounts to roughly $2^{n/2}+2^{n-k}$, where k is the number of blocks in the target message. In our design, n can be replaced by m, and hence with no upper limit on the message size, m should be at least $2n$ for this attack to have the same complexity as a brute-force search. If the target message can have length at most 2^r blocks, then $m \geq \min(2n, n+r)$ is required.

B Reduced variants

We now suggest some methods of reducing the cryptographic strength of the two proposals Grindahl-256 and Grindahl-512, without reducing the output size. The methods can be combined, but some reductions rule out others, or modify the way that others can be applied.

- Increase the size of each message block to more than one column.
- Reduce the number of columns in the extended state. The number of columns should be at least 8 plus twice the number of columns that are overwritten by the message block. Otherwise, as described in Section 2.4, birthday attacks are simplified.
- Reduce the number of blank rounds.

C A method for choosing rotation constants

We suggest the following method for choosing rotation constants in the Grindahl hash collection, given a particular geometry of the extended state. Let R_1 be the set of ν_{rw}-tuples of rotation constants that ensure optimal diffusion, i.e. all state bytes depend on all message bytes as quickly as possible. Let $R_2 \subseteq R_1$ be the subset of R_1 of rotation constants that ensure optimal diffusion in the blank rounds, i.e. when no extended state bytes are overwritten by message bytes.
Now let $f_j(d_i)$ be the number of state bytes that message byte d_i affects after j rounds. Let μ be the number of rounds needed for every state byte to be affected by every message byte. Now let $R_3 \subseteq R_2$ be the subset of R_2 of rotation constants for which the sum

$$\sum_{j=1}^{\mu-1} \sum_{i=0}^{\nu_{\mathrm{mb}}-1} f_j(d_i)$$

is maximal. Sort R_3 lexicographically, i.e. such that (r_1, r_2, r_3, r_4) comes before (s_1, s_2, s_3, s_4) if and only if $r_1 < s_1$, or $r_1 = s_1$ and $r_2 < s_2$, or $(r_1, r_2) = (s_1, s_2)$ and $r_3 < s_3$, or $(r_1, r_2, r_3) = (s_1, s_2, s_3)$ and $r_4 < s_4$. Choose as rotation constants the first tuple in the sorted R_3.

D A characteristic leading to a collision in Grindahl-256

Below is given a characteristic that leads from a zero-difference state to a zero-difference state (a collision) via message inputs containing differences. Exact differences are not given, instead a single bit for each byte is given stating whether or not there is a difference on that particular byte.

Round no.	Initial state	Message	Mesg. input →	ShiftRows →	MixColumns →
1	0000000000000	1111	1000000000000	0100000000000	0110100000100
	0000000000000		1000000000000	0010000000000	0110100000100
	0000000000000		1000000000000	0000100000000	0110100000100
	0000000000000		1000000000000	0000000000100	0110100000100
2	0110100000100	0111	0110100000100	0011010000010	0111101110111
	0110100000100		1110100000100	0011101000001	0111011110111
	0110100000100		1110100000100	0100111010000	0101111110111
	0110100000100		1110100000100	0100000100111	0111111110111
3	0111101110111	1001	1111101110111	1111110111011	1100010010010
	0111011110111		0111011110111	1101110111101	1000100100101
	0101111110111		0101111110111	0111010111111	1010010010110
	0111111110111		1111111110111	1111110111111	1011110001001
4	1100010010010	1100	1100010010010	0110001001001	1010000000000
	1000100100101		1000100100101	0110001001001	1100000000000
	1010010010110		0010010010110	0110001001001	1000000001001
	1011110001001		0011110001001	1110001001001	1000001000000
5	1010000000000	0000	0010000000000	0001000000000	1000000000000
	1100000000000		0100000000000	0001000000000	1000000000000
	1000000001001		0000000001001	1001000000000	1000000000000
	1000001000000		0000001000000	0001000000000	1001000000000
6	1000000000000	0000	0000000000000	0000000000000	1000000000000
	1000000000000		0000000000000	0000000000000	1000000000000
	1000000000000		0000000000000	0000000000000	1000000000000
	1001000000000		0001000000000	1000000000000	1000000000000
7	1000000000000	0000	0000000000000	0000000000000	0000000000000
	1000000000000		0000000000000	0000000000000	0000000000000
	1000000000000		0000000000000	0000000000000	0000000000000
	1000000000000		0000000000000	0000000000000	0000000000000

Evidently, after 6 rounds the state contains no difference (the only difference in the extended state is in the first column). Hence, this characteristic spans 6 rounds, but an additional message block with no difference (e.g., a padding block) is needed before the blank rounds in order for the difference to disappear entirely.