# Design and Analysis of a New Hash Function Gear

Mohammad A. AlAhmad
Public Authority for Applied Education and Training
College of Basic Education
Computer Science Department
P.O.Box 34567 Adaliyah, 73205 Kuwait City, Kuwait
malahmads@yahoo.com

Imad Fakhri Alshaikhli
Department of Computer Science, International
Islamic University of Malaysia, 53100 Jalan Gombak
Kuala Lumpur, Malaysia
imadf@iium.edu.my

## ABSTRACT.
A hash function usually has two main components: a compression function or permutation function and mode of operation. In this paper, we propose a new concrete novel design of a permutation based hash functions called Gear. It is a hash function based on block cipher in Davies-Meyer mode. It uses the patched version of Merkle-Damgård, i.e. the wide pipe construction as its mode of operation. Thus, the intermediate chaining value has at least twice larger length than the output hash. Also, we analyze Gear and prove it is hard to attack it with complexities significantly less than brute force and it resists all the generic attacks. And the permutations functions used in Gear are inspired from the SHA-3 finalist Grøstl hash function which is originally inspired from Rijndael design (AES). As a consequence there is a very strong confusion and diffusion in Gear.

## Categories and Subject Descriptors
K.6.5 [**MANAGEMENT OF COMPUTING AND INFORMATION SYSTEMS**]: Security and Protection – *Authentication, Insurance.*

## General Terms
Security

**Keywords:** WP - permutation – block cipher – AES

## 1. INTRODUCTION
Cryptographic hash functions have indeed proved to be the workhorses for modern cryptographic hash functions. Another name given to cryptographic hash functions is "Swiss knife army" because it can serve many different purposes such as digital signatures, conventional message authentication to secure passwords storage or forensics data identification. Cryptographic hash functions take an unfixed size of input and produce a fixed size of an output.


*SIN '14,* September 09 - 11 2014, Glasgow, Scotland Uk
Copyright 2014 ACM 978-1-4503-3033-6/14/09…$15.00.
http://dx.doi.org/10.1145/2659651.2659652

A hash function usually built from two main components: (1) a basic primitive compression function $C$ and (2) an iterative mode of operation $H$, where the symbol $H^C$ denotes the hash function $H^C$ based on the compression function $C$. Most hash functions in use today are so-called iterated hash functions, i.e. Merkle-Damgård (MD), based on iterating a compression function. Examples of iterated hash functions are MD4, MD5, SHA and RIPEMD-160. For a cryptographic hash function $H^C$, if the compression function $C$ is resistant to the following attacks, then the hash function considered secure:

- **Preimage**: given y = H(x), find x′ such that H(x′) = y,
- **2nd preimage**: given an x and y=H(x) find x′ ≠ x such that H(x′) = y,
- **Collision**: find x and x′ such that x′ ≠ x and H(x) =H(x′).

Recently, several collisions were announced which decreased the security of some of the existing hash functions. Particularly, collisions were announced in SHA-0, MD4, MD5, HAVAL-128, and RIPEMD. French researcher Antoine Joux et al. [17] presented the collision in SHA-0, and a group of collisions against MD4, MD5, HAVAL- 128, and RIPEMD were found by the Chinese researcher Xiaoyun Wang with co-authors Dengguo Feng, Lai, and Hongbo Yu [30]. After that, in February 2005, the same Xiaoyun Wang, Lisa Yiqun Yin, and Hongbo Yu found collisions in SHA-1 using $2^{69}$ hash computations [30]. Several strategies were developed to thwart these attacks. Stefan lucks et al. [23] introduced the Wide Pipe (WP) hash construction as an intermediate version of Merkle-Damgård to improve the structural weaknesses of Merkle-Damgård design. The process is similar to Merkle-Damgård algorithm steps except of having a larger internal state size, which means the final hash digest is smaller than the internal state size of bit length. For example, the final compression function compresses the internal state length (for ex, *2n-bits*) to output a hash digest of *n-bit*. This simply can be achieved by discarding the last half of *2n-bit* output. WP is used in this paper to construct Gear hash function. It is used as an operation of mode for Gear. Mridul Nandi and Souradyauti Paul et al. [31] proposed the fast wide pipe (FWP) construction to overcome these attacks. It is twice faster than the wide pipe construction. HAsh Iterated FrAmework (HAIFA) is also a patched version Merkle-Damgård construction [32]. HAIFA design solves many of the internal collision problems associated with the classic MD

construction design by adding a fixed (optional) salt of s-bits along with a (mandatory) counter $C_i$ of *t-bits* to every message block in the iteration *i* of the hash function. Wide-pipe and HAIFA are very similar designs. Where, sponge construction is an iterative construction designed by Guido Bertoni, Joan Daemen, Micheal Peeter and Gilles Van Assche to replace Merkle-Damgård construction [2]. It is a construction that maps a variable length input to a variable length output. Keccak (SHA-3 winner) hash function uses sponge construction. In the next section, we demonstrate our new proposal Gear hash function in more details.

## 2. OUR PROPOSAL

We propose a new hash function called Gear that supports *256-512 bits* digests. The basic building block of our hash is a block cipher. By applying standard design approaches next we create a compression function (based on the cipher), and finally a hash function. We use the following design techniques:

- The block cipher applies the wide trail strategy.
- A compression function based on the block cipher in Davies-Meyer mode.
- A hash built upon an iterative compression function with the Merkle-Damgård construction.
- A wide pipe construction, i.e. the intermediate chaining value has at least twice larger length than the output hash.

## 3. Design Goals

The main design goal of any modern hash function is the security of the construction. In the last several years, the notion of security has expanded to include not only the basic requirements on collisions and second preimage resistance, but also a wide variety of distinguishers. In fact, non-formally a hash function is supposed to behave as a random oracle. Although in this model, trivial distinguishers do exist for every hash function, the designers aim to construct hash function that will be resistant against all possible non-trivial distinguishers, i.e. the hash does not exhibit any structural distinguishers, and, in a line of notation from the Sponge design [2], it is a hermetic design.

We aim to achieve this high security requirement with our proposal as well. More precisely, we would like to achieve the standard security margin against the following attacks and structural distinguishers:

- No collisions can be found in *n-bit* Gear with significantly less than *2n*
  hash function invocations
- No (second) preimage can be found in *n-bit* Gear with significantly less
  then 2n invocations
- No non-trivial structural distinguishers can be found for Gear with a complexity significantly lower than the complexity required to find (or confirm) such property in a secure hash function (such as SHA-2, SHA-3, etc.) Here, we would like to point out that the deviation "significantly lower" from "lower" is introduced to annulate the analysis based on the recently discovered bicliques[7] – the latest results suggest that such analytical results are most likely applicable to all cryptographic primitives, thus one cannot expect the achieve the ideal security level. On the other hand, the complexity of the attacks not based on granulation of the compression function (i.e. all other analysis except bicliques), should always exceed our claimed security bound.

## 4. DESCRIPTION OF Gear

Our proposal Gear is a wide pipe hash function with an internal state of *1024 bits*. It supports digests of *1* to *512 bits*. For security reasons, we suggest a minimal output of *256 bits* – further we describe the two main versions Gear - *256* and Gear - *512*, with an output length of *256* and *512 bits*, respectively. We emphasize that these two versions, as well as all the possible versions with a hash output between *256* and *512 bits*, are based on the same primitive, and differ only in the number of bits that are truncated at the output of the primitive. Our hash function is based on a cipher *C*- Gear used in the Davies-Meyer mode to build a compression function. We use Merkle-Damgård to construct the hash upon this compression function. Further we describe in details the cipher and give a brief recall of the mode.

### 4.1 The Cipher C-Gear

The block cipher *C*- Gear (P, K) is an SP network with 16 rounds and designed according to the wide trail strategy. It has a state of *1024 bits* and supports *1024-bit* keys. The state as well as the key is seen as 8x16 matrix of bytes – with ai,j,bi,j,i = 0,...,7,j = 0,...,15 we denote the individual bytes of the state and key matrices, respectively.

In each of the 16 rounds, the state S undergoes four byte-oriented transformations, i.e. round R can be represented as:

$$R = AK \circ MC \circ SR \circ SB,$$

Where AK, MC, SR, SB are acronyms for AddRoundKey, MixColumns, ShiftRows, and SubBytes, respectively. An additional AddRoundKey is perform at the beginning of the state update transformations (known as key prewhitening).

The *1024-bit* subkey $K_i$ used in the *i*-th round is produced from the previous subkey $K_{i-1}$ with similar operations:

$$K_i = AC \circ MC \circ SR \circ SB(K_{i-1}),$$

Where AC stands for AddRoundConstant. The prewhitening key $K_0$ is the initial master key. The round and key schedule transformations are the standard operations used in most of the Rijndael-based primitives. For completeness of the description, in the sequel we give a brief definition. The superscripts new, old are used to denote the updated, previous values for the bytes (or the columns).

**SubBytes (SB).** This transformation is the only non-linear part of the cipher. It consists of independent application of 8x8 bit S-box to all the bytes of the state (or the subkey), etc.

$$a_{i,j}^{new} = S(a_{i,j}^{old}),$$
$$b_{i,j}^{new} = S(b_{i,j}^{old}).$$

We use the invertible AES S-box S(·) for this purpose which is a composition of a finite field inversion and an affine transformation. The precise definition of the S-box is given in Table 1 in the form $S(X_1X_2) = Y$.

**ShiftRows (SR).** It performs a cyclic shift of the rows of the matrix on different offsets that depend on the row index. The value of the offsets ria, rib, i = 0, . . . , 7 is different for the state and the key schedule:

$$a_{i,j}^{new} = a_{i,j+r_a^i \bmod 16}^{old},$$
$$b_{i,j}^{new} = b_{i,j+r_b^i \bmod 16}^{old}.$$

The precise values are given in Table 2.

**MixColumns (MC).** The diffusion among the bytes is achieved with this transformation. It is a multiplication of the columns $a_j$ , $b_j$ of the state/subkeys by a matrix M:

$$a_j^{new} = M \cdot a_j^{old},$$
$$b_j^{new} = M \cdot b_j^{old},$$

Where M is defined as:

$$M = \begin{pmatrix} 02 & 0c & 06 & 08 & 01 & 04 & 01 & 01 \\ 01 & 02 & 0c & 06 & 08 & 01 & 04 & 01 \\ 01 & 01 & 02 & 0c & 06 & 08 & 01 & 04 \\ 04 & 01 & 01 & 02 & 0c & 06 & 08 & 01 \\ 01 & 04 & 01 & 01 & 02 & 0c & 06 & 08 \\ 08 & 01 & 04 & 01 & 01 & 02 & 0c & 06 \\ 06 & 08 & 01 & 04 & 01 & 01 & 02 & 0c \\ 0c & 06 & 08 & 01 & 04 & 01 & 01 & 02 \end{pmatrix}.$$

**Table 1: The S-box used in Gear**

| $X_1/X_2$ | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | a | b | c |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 63 | 7c | 77 | 7b | f2 | 6b | 6f | c5 | 30 | 01 | 67 | 2b | fe |
| 1 | ca | 82 | c9 | 7d | fa | 59 | 47 | f0 | ad | d4 | a2 | af | 9c |
| 2 | b7 | fd | 93 | 26 | 36 | 3f | f7 | cc | 34 | a5 | e5 | f1 | 71 |
| 3 | 04 | c7 | 23 | c3 | 18 | 96 | 05 | 9a | 07 | 12 | 80 | e2 | eb |
| 4 | 09 | 83 | 2c | 1a | 1b | 6e | 5a | a0 | 52 | 3b | d6 | b3 | 29 |
| 5 | 53 | d1 | 00 | ed | 20 | fc | b1 | 5b | 6a | cb | be | 39 | 4a |
| 6 | d0 | ef | aa | fb | 43 | 4d | 33 | 85 | 45 | f9 | 02 | 7f | 50 |
| 7 | 51 | a3 | 40 | 8f | 92 | 9d | 38 | f5 | bc | b6 | da | 21 | 10 |
| 8 | cd | 0c | 13 | ec | 5f | 97 | 44 | 17 | c4 | a7 | 7e | 3d | 64 |
| 9 | 60 | 81 | 4f | dc | 22 | 2a | 90 | 88 | 46 | ee | b8 | 14 | de |
| a | e0 | 32 | 3a | 0a | 49 | 06 | 24 | 5c | c2 | d3 | ac | 62 | 91 |
| b | e7 | c8 | 37 | 6d | 8d | d5 | 4e | a9 | 6c | 56 | f4 | ea | 65 |
| c | ba | 78 | 25 | 2e | 1c | a6 | b4 | c6 | e8 | dd | 74 | 1f | 4b |
| d | 70 | 3e | b5 | 66 | 48 | 03 | f6 | 0e | 61 | 35 | 57 | b9 | 86 |
| e | e1 | f8 | 98 | 11 | 69 | d9 | 8e | 94 | 9b | 1e | 87 | e9 | ce |
| f | 8c | a1 | 89 | 0d | bf | e6 | 42 | 68 | 41 | 99 | 2d | 0f | b0 |

**Table 2: The offsets used in ShiftRows**

| Row | State offset $r_a^i$ | Key schedule offset $r_b^i$ |
|---|---|---|
| 0 | 0 | 0 |
| 1 | 1 | 2 |
| 2 | 2 | 4 |
| 3 | 3 | 5 |
| 4 | 5 | 6 |
| 5 | 6 | 8 |
| 6 | 7 | 9 |
| 7 | 8 | 10 |

We emphasize that the same matrix is used for both the state and key schedule. The multiplication is performed in GF($2^8$) defined with the irreducible polynomial $x^8 + x^4 + x^3 + x + 1$.

**AddRoundKey (AK).** The *1024-bit* subkey is xored to the state. The XOR can be seen as byte-wise, i.e.:

$$a_{i,j}^{new} = a_{i,j}^{old} \oplus b_{i,j}$$

**AddRoundConstant (AC).** A constant $C_i$ is xored to the subkey $K_i$ – in a similar fashion, it can be represented as a byte-wise operation. The value of the constants is dependent on the index *i*. It is defined as:

$$C_i = \begin{pmatrix} ff & ff & ff & ff & \dots & ff & ff & ff & ff \\ ff & ff & ff & ff & \dots & ff & ff & ff & ff \\ ff & ff & ff & ff & \dots & ff & ff & ff & ff \\ ff & ff & ff & ff & \dots & ff & ff & ff & ff \\ ff & ff & ff & ff & \dots & ff & ff & ff & ff \\ ff & ff & ff & ff & \dots & ff & ff & ff & ff \\ ff & ff & ff & ff & \dots & ff & ff & ff & ff \\ i\oplus 0 & i\oplus 1 & i\oplus 2 & i\oplus 3 & \dots & i\oplus c & i\oplus d & i\oplus e & i\oplus f \end{pmatrix}.$$

## 4.2 The Hash Function Gear

Once we have defined *C*- Gear, we use a standard approach to build a hash function based on this cipher. First, we define the compression function CF. It takes two inputs: *1024-bit* chaining value Hi and *1024-bit* message Mi, and produces *1024-bit* chaining value $H_{i+1}$ with Davies-Meyer mode of *C*- Gear, i.e.:

$$H_{i+1} = CF(H_i, M_i) = C\text{- Gear}\ (H_i, M_i) \oplus H_i$$

Further, we use this compression function to build a hash function with the Merkle-Damgård construction. Briefly, we fix an initial chaining value $H_0$ equal to the first *128 bytes* of the fractional part of π (see Table 3). We pad the message M (see below how the padding is performed), and split the expanded message into *1024-bits* chunks $M_i$. Next, we iterate all the message blocks using the compression function based on the Merkle-Damgård construction:

$$H_0 = IV$$

$$H_{i+1} = CF(H_i, M_i)$$

When the expanded message contains $l$ blocks, the output $H_{l+1}$ is used to produce the final hash based on truncation, i.e. the hash of M is $tr(H_{l+1})$, there tr(X) truncates the leftmost bits of X, depending on the hash size.

**Table 3: The initial chaining value $H_0$**

| |
|---|
| 24 3F 6A 88 85 A3 08 D3 13 19 8A 2E 03 70 73 44 |
| A4 09 38 22 29 9F 31 D0 08 2E FA 98 EC 4E 6C 89 |
| 45 28 21 E6 38 D0 13 77 BE 54 66 CF 34 E9 0C 6C |
| C0 AC 29 B7 C9 7C 50 DD 3F 84 D5 B5 B5 47 09 17 |
| 92 16 D5 D9 89 79 FB 1B D1 31 0B A6 98 DF B5 AC |
| 2F FD 72 DB D0 1A DF B7 B8 E1 AF ED 6A 26 7E 96 |
| BA 7C 90 45 F1 2C 7F 99 24 A1 99 47 B3 91 6C F7 |
| 08 01 F2 E2 85 8E FC 16 63 69 20 D8 71 57 4E 69 |

Thus, for *256-bit* digests, tr(X) outputs the *256* lefttmost (most significant) bits of X, while for *512-bit* digest this number is *512*. In general, for Gear −*n*, tr(X) outputs the *n* most significant bits of the last produced chaining value $H_{l+1}$.

**The padding.** This procedure produces expanded message $M_e$ from the original input message M. It assures that the length (in bits) of M is properly encoded into the expanded message $M_e$, and the length of $M_e$ is divisible by *1024*. To achieve this we use a trivial padding by attaching a required number of 0's to make the last message
block *1024 bits*, and always introduce an addition message block at the end that contains the length of M only.
Let M has *t bits*. Then from M, first we produce $M_{e^{\sim}}$ = M00...0, where the number of 0's is *1024−(t mod 1024)* when t is not divisible by *1024* – otherwise we do not attach any 0's. Next, we attach an additional *1024-bit* block that contains 1024 − 64 = 940 zeros, while the last *64 bits* are equal to *t*, i.e. the expanded message is defined as $M_e = M_{e^{\sim}}$00...0tbinary.

**Endian and mappings.** Our hash function is little endian oriented – it regards *64-bit* words as *8 bytes* in reverse order (with the least significant byte coming first). Furthermore, the mapping of byte sequence to matrix of the state (or the key schedule) is from left to right, and top row to bottom row. For example, the *128-byte* sequence a1, . . . , a128 is mapped to the matrix as follows:

$$\begin{matrix} a_1 & a_2 & a_3 & \dots & a_{16} \\ a_{17} & a_{18} & a_{19} & \dots & a_{32} \\ \dots & \dots & \dots & \dots & \dots \\ a_{113} & a_{114} & a_{115} & \dots & a_{128} \end{matrix}$$

# 5. PSEUDO CODE AND TEST VECTORS

The pseudo codes of state round, keyschedule round, *C*- Gear and Gear is given in Algorithm 1-4 respectively

**Algorithm 1** State Round(S, $K_i$)

S ← SubBytes(S)
S ← ShiftRows(S)
S ← MixColumns(S)
S ← AddRoundKey(S, Ki)

end

**Algorithm 2** KeySchedule Round($K_i$, *i*)

$K_{i+1}$ ← SubBytes($K_i$)
Ki+1 ← ShiftRows(Ki+1)
Ki+1 ← MixColumns(Ki+1)
Ki+1 ← AddRoundConstant(Ki+1, i)

end

**Algorithm 3** *C*− Gear *(P, K)*

S ← AddRoundKey(P, K) $K_0$ ← K
for *i* = 0 to 15 do
$K_{i+1}$ ← KeySchedule Round($K_i$, *i*)
S ← State Round(S) end for

end

**Algorithm 4** Gear (M)

$M_0|M_1|...|M_l$ ←padded(M) $H_0$ = IV
for *i* = 0 to l do
$H_{i+1}$ =*C*− Gear (Hi,Mi)⊕$H_i$
**end for**
output truncated($H_l$)

**end**

A list of test vectors in given in Table 4.

**Table 4: Test vectors for Gear -*512***

| Gear (“ ”) |
|---|
| 8798dbba48ffd3b62e239b549499c09b<br>3d4637273489f9061f5e1d8d214e31ae<br>1dc13d88a561c5594c9937ee864140e9<br>7f7b93ffd27e79251d4755a20eca60a4 |
| Gear ("The quick brown fox jumps over the lazy dog") |
| 9b182c6da0010a92e6df1dd67515764b<br>53a909aecc9be8dbf1c47bf876b4be42<br>7b96491fbf8e2e90453b4ac9cabf4b5d<br>73394019ca7801d11307e8d000eed3e2 |
| Gear ("The quick brown fox jumps over the lazy dag") |
| 257269675f2d432ba8dbece0b25d4ac9<br>a95450c9788a6ef65cee1d1e349b7ed4<br>a13e0302d0d8204f17832933896ac7e4<br>4b9709fd6ddb0f86732200955b51648e |

# 6. THE SECURITY OF THE MODE

In our design we use two widely applied techniques for construction of hash functions, the Davies-Meyer mode and the Merkle-Damgård construction.

The security of the single-block-length block cipher modes has been analyzed in [28, 6]. In particular, Black-Rogaway-Shrimpton have proved that the Davies-Meyer mode has asymptotically optimal bound for collision and preimage resistance, i.e. the number of queries to the underlying cipher with randomly chosen key (a black box access) to find collisions or a preimage is roughly as predicted by the generic bound. Thus this mode is secure against the standard attacks and shortcut attacks can be found only by exploiting a weakness in the block cipher (but not in the mode). Therefore Gear is secure against the traditional attacks as long as *C*- Gear is secure.

The Merkle-Damgård (MD) construction [12, 26] is an approach for building a collision resistant hash function from a collision resistant compression function. That is, if the hash function applies appropriate padding and the initial value is fixed, the hash function is collision resistant as long as the compression function has the same property. Note that in Gear, the initial value is fixed, and the padding is as required, thus the for collision resistance one only has to focus on the compression function.

## 6.1 The Wide Pipe Construction

The wide pipe construction proposed by Lucks [23] was developed to strengthen the security of the standard Merkle-Damgård against a variety of generic attacks. Most of these attacks use the fact that the standard single-pipe chaining value and internal state can be insufficient against attacks that target the intermediate chaining values. In particular:

- **Length extension attacks** – once the attacker has a single collision he can produce many more colliding message pairs. Assume $H(\cdot)$ is a single-pipe hash, and $M_1$, $M_2$ are such that $H(M_1) = H(M_2)$. Then for any M, $H(M_1|M) = H(M_2|M)$, thus the pair $(M_1|M, M_2|M)$ is also a colliding pair. However, for wide-pipe hash function (such as in Gear), in general this is not true. The initial message pair $M_1$, $M_2$ collides only on half of the bits – the other half is truncated, and not necessarily produces collisions. Thus, extending the colliding pair with additional message results in a different input chaining values for the last compression function, and most likely, different hash values.
- **Second preimage attack by Kelsey-Schneier[19]** – when the hashed message has *l* blocks (*l* invocations of compression functions), the complexity of finding a second preimage is $2^{n-l}$ instead of the generic $2^n$. This comes from the fact that if the attacker is able to find a second preimage of any of the intermediate chaining values, then he will succeed to find a preimage for the whole hash. Thus instead of one final target (the digest), he can aim any of the *l n-bit* values. However, as in Gear the intermediate chaining values have at least *2n bits*, the complexity of finding a second preimage for these values is at least $2^{2n}$ (instead of $2^n$ as in single-pipe). Thus, the wide-pipe Gear is resistant against this type of generic attacks.
- **Multicollisions by Joux[17]** - producing multicollisions (many different messages hash to the same value) has much lower complexity than the generic bound. Joux has shown that for a single-pipe MD hash function, one can produce $2^t$-collisions with only $t \cdot 2^{n/2}$ calls to the compression function. Joux's idea is very simple and original – he proposed creating sequentially collisions for the consecutive compression function calls. That is, first one finds a colliding message pair $(M_1^1, M_2^1)$ for the first compression function, then $(M_1^2, M_2^2)$ for the second (the input chaining value coincides with the output of the previous), and keeps repeating this procedure for all t compression function calls. Then, all $2^t$ messages $M_{i1}^1 | M_{i2}^2 | \ldots | M_{il}^1$, $i_j \in 1, 2$ hash to the same value. Again, to succeed with the above attack, one has to be able to find collisions (for the compression function), with a time complexity of finding collisions for the whole hash. However, in the double-pipe hash Gear, finding the intermediate collisions requires an effort of at least $2^n$ compression function invocations. Therefore, Joux's attack is not applicable to Gear.
- **Herding attack by Kelsey-Kohno[18]** – the attacker presents a digest h, and then for an arbitrary message M he is able to find $M_2$ such that $H(M|M_2) = h$. The idea behind the herding attacks is the production of aso-called diamond structure. In brief, the attack is based again on producing collisions for the intermediate chaining values. Same as above, in Gear this type of attack is prevented by the wide-pipe design.

## 6.2 The Wide Trail Strategy

The wide trail strategy [11] is one of the most popular approaches for designing block ciphers and cryptographic hash functions resistant against differential and linear attacks. Daemen and Rijmen noticed that the diffusion layer in SP ciphers can be chosen in a way that ensures a high number of differentially (or linearly) active S-boxes in any round-reduced characteristic. Two basic concepts are used for applying the wide trail: branch number and alternation of two different round transformations (which indeed can be combined into a single one). The branch number assures a minimal number of active S-boxes in any two-round characteristic. As in *C*- Gear, the diffusion layer is based on MDS code (see the matrix multiplication), the branch number is maximal and equals to 9 – that is, any two-round differential (or linear) characteristic has at least 9 active S-boxes. The alternating transformations are achieved with two different linear layers – in the case of *C*- Gear these are the ShiftRows and MixColumns operations. As ShiftRows moves each row of the matrix to a different position, by Theorem 2 from [11], we get that any four-round trail has $9 \cdot 9 = 81$ active S-boxes. Further in our analysis, we will use this lower bound to prove the resistance of Gear against various attacks.

### 6.2.1 Collision Attacks

The collision attacks on hash functions are based on finding differential trails with zero output difference. However, unlike differential distinguishers, where the probability can be as low as $2^{-n}$ for n-bit hash, the trails for collisions have to have at least $2^{-n/2}$ – otherwise, the generic collision finding algorithm (based for example on the Floyd's cycle finding algorithm) would have lower complexity. We will show further in our analysis that no differential trail exists for *C*- Gear with a probability higher than $2^{-n}$, which immediately allows to conclude that collision attacks based on differential trails are not applicable to Gear. Another

type of collision attacks are based on the use of weak modes for the compression function. However, as we have shown earlier, the mode of Gear is secure. We emphasize as well that the use of Merkle-Damgård construction assures that since our compression function is collision resistant, then the whole hash function Gear is collision resistant as well.

### 6.2.2 Preimage Attacks

The (second) preimage attacks for hash function based on secure modes usually exploit the weak message expansion, and in particular the low diffusion. Most of these attacks are based on the meet-in-the-middle (MITM) attack and the recent improvement in the form of splice and cut [1]. Although no sufficient conditions are currently available that ensure the compression function is secure against preimage attacks, a good rule of the thumb is to have a high diffusion in the message expansion. In Gear, the compression function is based on the cipher *C*- Gear that has a very high diffusion in the key schedule. Notice that in each round of the cipher, the whole key is used, and after only three rounds, the key schedule achieves a full diffusion of the bits of the key. Thus, it is expected that preimage attacks cannot be launched on very high number of rounds. The precise bound (or at least currently the best result) is achieved by taking into account the latest results on the similar hash function Grøstl [13]. Following the result on Wu et al. [29], it is clear that by using the partial matching technique and chunk separation, one can launch a pseudo-preimage attack on 8 rounds of Gear -*512*, with around $2^{507}$ time and memory complexity – we omit the details as the analysis is very similar to the one presented in [29]. We also note that shortcut attacks that exploit weak modes are discarded as well as the mode used in Gear is provably secure against preimage attacks.

### 6.2.3 Distinguishers

Non-trivial distinguishing attacks became increasingly popular during the SHA- 3 competition [27]. In this section we show the resistance of Gear against all possible known distinguishers for byte-oriented primitives.

#### *A. Differential and Linear Distinguishers*

Let us first examine the resistance of *C*- Gear against the two most popular forms of analysis: the differential [3] and linear cryptanalysis [24]. Here we want to emphasize one important point – the claimed security level of the examined cipher will be only in accordance to the application for the hash function. As the maximal output size of Gear is *512 bits* (all other versions have smaller output, thus generic attacks have lower complexity), we examine only the security of Gear -*512*. Thus, we need to prove that no differential and linear attacks on *C*- Gear exist with complexity lower than $2^{512}$. Although we do not claim higher security level for *C*- Gear, it is easy to extend the below analysis to reach such level – we omit the details as we use *C*- Gear only as an underlying cipher for *512-bit* hash.

- **Linear attacks**

We have seen that *C*- Gear follows the wide trail strategy; hence any 4-round trail has at least 81 active S-boxes. The best linear bias of the S-box used in *C*- Gear is $2^{-3}$, thus the probability of any 4-round linear trail is at most $2^{-3\cdot 81} = 2^{-243}$, while for any 12-round trail is at most $2^{3\cdot(-243)} = 2^{-729}$. Hence, *C*- Gear achieves the claimed security level of 512 against linear cryptanalysis. We point out as well that the low probability linear trail $2^{-729}$ requires an amount of approximately $2^{1458}$ pairs of plaintext-ciphertext which exceeds the whole codebook – thus the security level of the cipher against linear cryptanalysis is actually *1024 bits*.

- **Standard differential attacks**

First let us take a look at standard differential attacks and in particular single-key differential trails. When there is no difference in the key of *C*- Gear (which can be translated into no difference in the message block of Gear), the resistance against differential attacks comes straightforwardly from the wide trail strategy: 1) the maximal differential propagation probability of the S-box is $2^{-6}$, 2) any four-round differential trail has 81 active S-box. Thus, the probability of any four-round differential trail is $2^{-6\cdot 81} = 2^{-486}$, while the probability of any eight-round trails is $2^{-2\cdot 486} = 2^{-972}$. Obviously the low probability suffices to prove the claimed security bound of *512 bits*. Better bounds (lower probability trails) can be proven when trails are on 12 rounds – then the security level of *1024 bits* is achieved. We avoid this, as for Gear we need a security level of only *512 bits*.

Related-key differential attacks on *C*- Gear do not improve the complexity of the best attacks. This comes from the fact that the key schedule of *C*- Gear undergoes the same (or very similar) transformations. Thus the probability of any related-key differential characteristic, only in the key schedule, would be at most $2^{-972}$ for eight rounds. When *C*- Gear is used in the hash function mode (as in Gear), the attacker has the freedom to choose the key – thus let us further examine the possibility of message modification. For this purpose, we first obtain tighter bounds on probability. From the wide trail strategy it follows that any two-round trail has at least 9 active S-boxes and any four-round has 81 active. Hence, any six consecutive rounds have 90 active S-boxes and the probability of such differential trail is $2^{-6\cdot 90} = 2^{-540}$, i.e. it is lower than $2^{-512}$ (which we need as we work with *512-bit* hash). The attacker can use message modification and choose the value of the state and the subkey in order to pass some rounds for free. However, out of all 16 rounds, he has to pass 11 rounds with the modification. As both the state and the key schedule are highly complex, we believe that this is hard to achieve, and estimate that only 2-4 rounds can be passed for free with message modification. This brings the total number of attacked rounds to $7^{-9}$ (2,3,4 rounds for free + 5 rounds probabilistically).

- **Truncated differential attacks**

Truncated differentials [21] became increasingly popular as form of analysis of byte-oriented primitives after the invention of the Rebound attack [25] and Super S-boxes [14, 22]. These techniques have shown that the message modification combined with truncated differential can significantly increase the number of attacked rounds in frameworks such as known-key distinguishers for block ciphers or hash function attacks. Moreover, they stressed out that one cannot known in advance how many rounds can be passed for free when using message modification. In our analysis below we assume that this number is four as this is the state-of-the-art – we point out that further advancement in this field may bring up the number of rounds. However, the large security margin in Gear assures that only significant progress can influence security of our hash function.

Our design is similar to the hash function Grøstl [13], thus we follow the line of research given in [16] and show a truncated differential attack on 10 rounds of Gear -*512*. The differential is

given in Figure 1. The number of active S-boxes in the trail is as follow:

$64 \rightarrow 8 \rightarrow 1 \rightarrow 8 \rightarrow 64 \rightarrow 128 \rightarrow 64 \rightarrow 8 \rightarrow 1 \rightarrow 8 \rightarrow 64$

Using the technique from [16], we assume that the four middle rounds, i.e. $8 \rightarrow 64 \rightarrow 128 \rightarrow 64 \rightarrow 8$, are part of the inbound phase of the rebound attack, thus it is passed for free. The remaining six rounds, the first three, and the last three, are the outbound phase, and are passed probabilistically. The probability of this phase is $2^{2\cdot(-56)} = 2^{-112}$ – for each transition $8 \rightarrow 1$, it is $2^{-56}$, while the for the rest ($1 \rightarrow 8$, $8 \rightarrow 64$), the probability is 1.

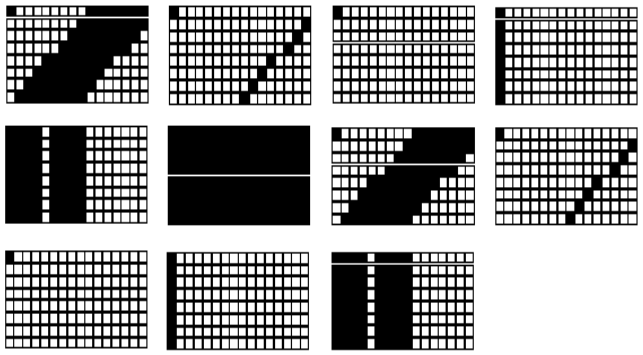

**Figure 1. The truncated differential for 10-round attack on Gear -*512***

The complexity of finding a conforming pair for the inbound phase, is $2^{280}$ time and $2^{64}$ memory (see [16] for details). Thus the total complexity of the attack is $2^{112}+2^{80} = 2^{392}$ time and $2^{64}$ memory.

### *B. Slide Attacks*

Slide attacks [4, 5] exploit rounds self-similarity and can be devastating for launching attacks on ciphers that use completely equal round transformations. To stop this type of attacks, round constants are introduced. *C*- Gear does not employ constants as part of the state transformations; however, the key schedule applies the AddRoundConstant operation which assures that each round of the key schedule is different (note that the round constants $C_i$ depend on the round index *i*). Any slid pair (with one or a few rounds apart), that is completely identical at the beginning, has to differ in the following round in at least 16 bytes of the subkey – the whole bottom row would be different as the round index is different. This leads to a very fast expansion of the key difference (between the elements of the slid pair) in the few consecutive rounds which in turn assures a high number of active S-boxes. Hence slide attacks could be possibly applied to Gear on few rounds only.

### *C. Integral Attacks*

Integral (or square) attack was first launched against the block cipher Square [10]. In general, it is applicable to any Rijndael-like cipher, and it exploits the fact that the S-boxes are invertible. Unlike for ciphers, where integral attacks lead to a key recovery, for hash functions, the additional rounds before and after the square property cannot be efficiently exploited. Thus as Gear is Rijndael-based hash function, integral property can be exploited and we expect that integral attacks for Gear can be launched on around three-five rounds.

### *D. Rotational Cryptanalysis*

Rotational attacks [20] follow the expansion through the rounds of the primitive of a pair of inputs where the second is a rotation of the first, i.e. each word (or possibly a byte or a column) of the second state, is produced by rotating the corresponding word of the first state. In general, rotational analysis is applicable to addition-rotation-xor primitives, however byte-oriented ciphers and hash function can be susceptible when the underlying transformations are rotational- friendly. The main method for achieving resistance against rotational attacks is the use of constants. In Gear, this is achieved by the AddRoundKey transformation. Note, that the key schedule assures that no rotational subkey pair can be produced in several consecutive rounds. Hence, we can conclude that rotational analysis is possibly applicable only to a few rounds of the compression function.

### *E. Resistance Against Other Dedicated Distinguishers*

The methods of analysis of byte-oriented primitives have been known for a while now. In the previous sections we have investigated all such methods. Further we present a dedicated approach that might be applicable only to our function and the underlying block cipher. In fact we show that we have taken the necessary steps to stop this type of attack. Note that the state and the key have the same size and use very similar transformations. A possible attack that might exploit this type of property is the one where the adversary switches the key and the plaintext and produces the same ciphertext, i.e. EK(P) = EP(K). However, to launch such attack the transformations should be the same, or at least similar – in this case the property might work for particular inputs only. The transformations in the state and in the key schedule differ at two places: ShiftRows and key/constant addition. If at the input of ShiftRows the state and the subkey have the same value, then at the output would be the same only if all the bytes within the row are equal. To achieve the same property for the addition, AddRoundKey and AddRoundConstant should be the same as well, i.e. the constant has to coincide with the subkey. However, it is clear that since in AddRoundConstant the last row byte constants are different, the output of the next application of ShiftRows will not produce equal values for the last row. Thus Gear resists this type of distinguisher.

## 7. CONCLUSION

We have presented a new cryptographic hash function Gear that supports digests of up to *512 bits*. Our proposal is based on the wide trail strategy and uses an underlying block cipher with *1024 bit* key and state. We use mode and construction with longstanding security analysis and provable against most of the generic attacks. We have shown that the hash function has a high security margin against all the known attacks available today. The claimed security level of Gear is given in Table 5.

**Table 5: The claimed security level of Gear and comparison to the level of an ideal hash function.**

| Hash | Collision | Preimage | Second Preimage | Distinguisher |
|---|---|---|---|---|
| **Gear-256**<br>Ideal-256 | $2^{128}$<br>$2^{128}$ | $2^{256}$<br>$2^{256}$ | $2^{256}$<br>$2^{256}$ | $2^{256}$<br>$2^{256}$ |
| **Gear-512**<br>**Ideal-512** | $2^{256}$<br>$2^{256}$ | $2^{512}$<br>$2^{512}$ | $2^{512}$<br>$2^{512}$ | $2^{512}$<br>$2^{512}$ |
| **Gear-n**<br>**Ideal-n** | $2^{n/2}$<br>$2^{n/2}$ | $2^{n}$<br>$2^{n}$ | $2^{n}$<br>$2^{n}$ | $2^{t}$<br>$2^{t}$ |

## 8. References


[1] K. Aoki and Y. Sasaki. Preimage attacks on one-block MD4, 63-step MD5 and more. In R. M. Avanzi, L. Keliher, and F. Sica, editors, Selected Areas in Cryptography, volume 5381 of Lecture Notes in Computer Science, pages 103–119. Springer, 2008.

[2] G. Bertoni, J. Daemen, M. Peeters, and G. V. Assche. On the indifferentiability of the Sponge construction. In N. P. Smart, editor, EURO- CRYPT, volume 4965 of Lecture Notes in Computer Science, pages 181– 197. Springer, 2008.

[3] Biham and A. Shamir. Differential cryptanalysis of DES-like cryptosystems. J. Cryptology, 4(1):3–72, 1991.

[4] A. Biryukov and D. Wagner. Slide attacks. In L. R. Knudsen, editor, FSE, volume 1636 of Lecture Notes in Computer Science, pages 245–259. Springer, 1999.

[5] A. Biryukov and D. Wagner. Advanced slide attacks. In B. Preneel, editor, EUROCRYPT, volume 1807 of Lecture Notes in Computer Science, pages 589–606. Springer, 2000.

[6] J. Black, P. Rogaway, and T. Shrimpton. Black-box analysis of the block- cipher-based hash-function constructions from PGV. In M. Yung, editor, CRYPTO, volume 2442 of Lecture Notes in Computer Science, pages 320– 335. Springer, 2002.

[7] A. Bogdanov, D. Khovratovich, and C. Rechberger. Biclique cryptanalysis of the full AES. In D. H. Lee and X. Wang, editors, ASIACRYPT, volume 7073 of Lecture Notes in Computer Science, pages 344–371. Springer, 2011.

[8] G. Brassard, editor. Advances in Cryptology - CRYPTO '89, 9th Annual International Cryptology Conference, Santa Barbara, California, USA, Au- gust 20-24, 1989, Proceedings, volume 435 of Lecture Notes in Computer Science. Springer, 1990.

[9] A. Canteaut, editor. Fast Software Encryption - 19th International Work- shop, FSE 2012, Washington, DC, USA, March 19-21, 2012. Revised Selected Papers, volume 7549 of Lecture Notes in Computer Science. Springer, 2012.

[10] J. Daemen, L. R. Knudsen, and V. Rijmen. The block cipher Square. In E. Biham, editor, FSE, volume 1267 of Lecture Notes in Computer Science, pages 149–165. Springer, 1997.

[11] J. Daemen and V. Rijmen. The wide trail design strategy. In B. Honary, editor, IMA Int. Conf., volume 2260 of Lecture Notes in Computer Science, pages 222–238. Springer, 2001.

[12] I. Damg˚ard. A design principle for hash functions. In Brassard [8], pages 416–427.

[13] P. Gauravaram, L. R. Knudsen, K. Matusiewicz, F. Mendel, C. Rechberger, M. Schlaffer, and S. S. Thomsen. Grøstl–a sha-3 candidate. Submission to NIST, 2008.

[14] H. Gilbert and T. Peyrin. Super-Sbox cryptanalysis: Improved attacks for AES-like permutations. In Hong and Iwata [15], pages 365–383.

[15] S. Hong and T. Iwata, editors. Fast Software Encryption, 17th International Workshop, FSE 2010, Seoul, Korea, February 7-10, 2010, Revised Selected Papers, volume 6147 of Lecture Notes in Computer Science. Springer, 2010.

[16] J. Jean, M. Naya-Plasencia, and T. Peyrin. Improved rebound attack on the finalist grøstl. In Canteaut [9], pages 110–126.

[17] A. Joux. Multicollisions in iterated hash functions. application to cascaded constructions. In M. K. Franklin, editor, CRYPTO, volume 3152 of Lecture Notes in Computer Science, pages 306–316. Springer, 2004.

[18] 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.

[19] J. Kelsey and B. Schneier. Second preimages on n-bit hash functions for much less than 2n work. In R. Cramer, editor, EUROCRYPT, volume 3494 of Lecture Notes in Computer Science, pages 474–490. Springer, 2005.

[20] D. Khovratovich and I. Nikolic. Rotational cryptanalysis of ARX. In Hong and Iwata, pages 333–346.

[21] L. R. Knudsen. Truncated and higher order differentials. In B. Preneel, editor, FSE, volume 1008 of Lecture Notes in Computer Science, pages 196–211. Springer, 1994.

[22] M. Lamberger, F. Mendel, C. Rechberger, V. Rijmen, and M. Schla¨ffer. Rebound distinguishers: Results on the full Whirlpool compression func- tion. In M. Matsui, editor, ASIACRYPT, volume 5912 of Lecture Notes in Computer Science, pages 126–143. Springer, 2009.

[23] S. Lucks. A failure-friendly design principle for hash functions. In B. K. Roy, editor, ASIACRYPT, volume 3788 of Lecture Notes in Computer Science, pages 474–494. Springer, 2005.

[24] M. Matsui. Linear cryptoanalysis method for DES cipher. In T. Helleseth, editor, EUROCRYPT, volume 765 of Lecture Notes in Computer Science, pages 386–397. Springer, 1993.

[25] F. Mendel, C. Rechberger, M. Schla¨ffer, and S. S. Thomsen. The Rebound attack: Cryptanalysis of reduced Whirlpool and Grøstl. In O. Dunkelman, editor, FSE, volume 5665 of Lecture Notes in Computer Science, pages 260–276. Springer, 2009.

[26] R. C. Merkle. One way hash functions and DES. In Brassard [8], pages 428–446.
[27] National Institute of Standards and Technology. Cryptographic hash al- gorithm competition. http://csrc.nist.gov/groups/ST/hash/sha-3/ index.html.
[28] B. Preneel, R. Govaerts, and J. Vandewalle. Hash functions based on block ciphers: A synthetic approach. In D. R. Stinson, editor, CRYPTO, volume 773 of Lecture Notes in Computer Science, pages 368–378. Springer, 1993.
[29] S. Wu, D. Feng, W. Wu, J. Guo, L. Dong, and J. Zou. (Pseudo) preimage attack on round-reduced Grøstl hash function and others. In Canteaut [9], pages 127–145.
[30] Wang, Xiaoyun, Hongbo Yu, and Yiqun Lisa Yin. "Efficient collision search attacks on SHA-0." *Advances in Cryptology–CRYPTO 2005*. Springer Berlin Heidelberg, 2005.
[31] Nandi, M. and S. Paul (2010). "Speeding up the wide-pipe: Secure and fast hashing." Progress in Cryptology-INDOCRYPT 2010: 144-162.
[32] Eli Biham and Orr Dunkelman, "A Framework for Iterative Hash Functions - HAIFA," Cryptology ePrint Archive, 2007. [Online]. http://eprint.iacr.org/2007/278