Cryptanalysis of WIDEA

Gaëtan Leurent(✉)

UCL Crypto Group, Louvain-la-Neuve, Belgium
Gaetan.Leurent@uclouvain.be

Abstract. WIDEA is a family of block ciphers designed by Junod and Macchetti in 2009 as an extension of IDEA to larger block sizes (256 and 512 bits for the main instances WIDEA-4 and WIDEA-8) and larger key sizes (512 and 1024 bits, respectively). WIDEA-w is composed of w parallel copies of the IDEA block cipher, with an MDS matrix to provide diffusion between them. An important motivation was to use WIDEA to design a hash function.

In this paper we present low complexity attacks on WIDEA based on truncated differentials. We show a distinguisher for the full WIDEA with complexity only 2^{65}, and we use the distinguisher in a key-recovery attack with complexity $w \cdot 2^{68}$. We also show a collision attack on WIDEA-8 if it is used to build a hash function using the Merkle-Damgård mode of operation.

The attacks exploit the parallel structure of WIDEA and the limited diffusion between the IDEA instances, using differential trails where the MDS diffusion layer is never active. In addition, we use structures of plaintext to reduce the data complexity.

Keywords: Cryptanalysis · Block cipher · Hash function · Truncated differential · IDEA · WIDEA · HIDEA

1 Introduction

Block ciphers are one of the most useful and versatile primitive in symmetric cryptography. Their basic use is to encrypt data and provide confidentiality, but they can also be used to build MAC algorithms (e.g. CBC-MAC), stream ciphers (e.g. in counter mode) and hash functions (e.g. using the Davies-Meyer or Matyas-Meyer-Oseas mode). Block ciphers are relatively well understood and we have well-established ciphers suitable for most uses such as DES, AES, IDEA, RC5, or Blowfish. However, there are still some new proposals to accommodate specific needs such as large block size, low resources, reduced leakage, or high speed on a particular platform. All these designs must be studied in depth before they can be trusted and used in actual products. In this paper we study the recent proposal WIDEA, which is based on IDEA.

IDEA. The "International Data Encryption Standard" (IDEA) is a block cipher designed by Lai and Massey in 1991 [12]. IDEA is a modification of their earlier "Proposed Encryption Standard" (PES) [11] and was initially called Improved PES (IPES). IDEA uses 8.5 rounds of the so-called Lai-Massey scheme [16], and mixes operations from incompatible structures ($\oplus$, $\boxplus$, and $\odot$). It is well-considered in the cryptographic community, and used in some products (e.g. in PGP), but its adoption has been limited by IP restrictions.

After years of cryptanalysis, most of the known cryptanalytic techniques have been used against IDEA: differential, linear, differential-linear, boomerang, impossible differentials, bicliques, weak-keys, related-keys, . . . Still, the best attacks in a block cipher scenario do not really affect the security of IDEA: attack with a significant margin only reach 6 rounds [1,3,10,14] and only marginal attacks have been shown on the full version [3,10]. On the other hand, the key schedule has been shown to be weak, and this gives classes of weak keys [4–6], related-key attacks [2], and attacks in various hashing modes [17].

WIDEA. At FSE 2009 Junod and Macchetti proposed to revisit the IDEA philosophy [9] in the light of modern CPU architectures. They gave a wordslice implementation of IDEA using the vector instructions available in many current CPU (*SSE* on x86, *Altivec* on PowerPC, *NEON* on ARM . . .) and design a new wide block cipher based on IDEA: WIDEA.

WIDEA-w is built from w parallel IDEA instances, using MDS matrices for the diffusion across the parallel instances. WIDEA is quite fast on CPU with vector instructions because the IDEA instances can be computed simultaneously. WIDEA was expected to retain the good security properties of IDEA because it follows the same design criteria: it mixes operations from incompatible structures ($\oplus$, $\boxplus$, $\odot$, and $\otimes$) and full diffusion is achieved after one round.

WIDEA-w has a blocksize of $64 \cdot w$ bits and a key size of $128 \cdot w$ bits. The main versions considered by the designers are WIDEA-4 and WIDEA-8; the large block size and key size are justified with the idea of using them to design a hash function.

Previous Analysis of WIDEA. Recently, Nakahara [7] and Mendel *et al.* [13] found weak keys for WIDEA, similar to the weak keys of IDEA [5]. Mendel *et al.* used the weak key property to create a free-start collision attack when WIDEA is used in hash function mode.

1.1 Our Results

In this paper, we study the security of WIDEA as a block cipher, and when used in a hashing mode. Our main result is a key recovery attack with complexity 2^{70} or 2^{71} which shows that WIDEA is very far from the expected strength of a 512-bit or 1024-bit cipher. The large gap between the security of IDEA and WIDEA is due to the insufficient diffusion across the parallel IDEA instances.

Table 1. Comparision of attacks on WIDEA

Attack	Version	Data	Time	Mem.	Ref, notes
CF collisions	HIDEA-512 ($w = 8$)		2^{14}		[13], free-start
Distinguisher	WIDEA-w	2^{65} CP	2^{65}	2^{64}	3
Success: 63 %	$w \geq 4$	2^{65} CP	2^{71}	2^{64}	5.1, Seq. M
		$5 \cdot 2^{65+t/2}$ ACP	$5 \cdot 2^{65+t/2}$	2^{64-t}	5.2
Key recovery	WIDEA-w	$w \cdot 2^{68}$ CP	$w \cdot 2^{68}$	2^{64}	4
	$w \geq 4$	$w \cdot 2^{68}$ CP	$w \cdot 2^{74}$	2^{64}	5.1, Seq. M
		$5w \cdot 2^{68+t/2}$ ACP	$5w \cdot 2^{68+t/2}$	2^{64-t}	5.2
Hash collisions	HIDEA-512 ($w = 8$)		2^{224}		6

We describe a simple truncated differential trail in Sect. 2, where the MDS diffusion layer is never active. This allows to keep a single IDEA instance active and to have a relatively high probability for the trail. We show how to build a distinguisher for WIDEA using structures of plaintext in Sect. 3. We give a full key recovery attack in Sect. 4, and we discuss some techniques to reduce the memory cost in Sect. 5. Finally, we study WIDEA used as a hash function, and give a collision attack based on the same differential trail in Sect. 6.

Attack Settings. A block cipher is expected to behave like a family of pseudo-random permutations: for an unknown key K, E_K should be indistinguishable from a truly random permutation. In this paper, we consider two different settings, and our results are listed in Table 1:

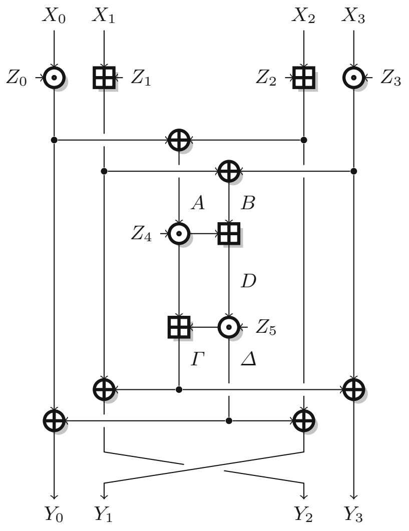
Chosen Plaintext Attack: The adversary builds a list of plaintext P_i, and receives the corresponding ciphertexts $C_i = E_K(P_i)$ under an unknown key K.

Adaptively Chosen Plaintext Attack: The adversary is given black-box access to a block cipher E_K with an unknown key K. He can ask for the encryption of any plaintext, and the choice of the plaintext can depend on previous answers.

1.2 Description of WIDEA

We give a brief description of WIDEA, but our attack is independent of most low-level details of the design. WIDEA, like IDEA, is a 16-bit oriented cipher, and combines operations from several algebraic structures of size 2^{16}. The elements of these structures are all mapped to 16-bit words, and the cipher uses the operations alternatively. We use the following notations:

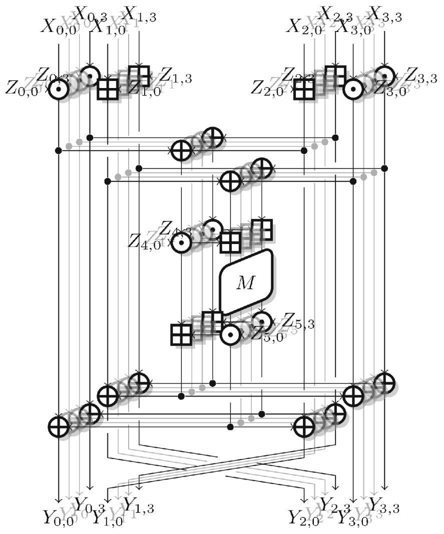
E	Block cipher	*Operations on 16-bit words:*	
P	Plaintext	$\boxplus$	Addition modulo 2^{16}
C	Ciphertext	$\odot$	Multiplication modulo $2^{16}+1$
X	State		(`0x0000` represents 2^{16})
K	Master Key	$\oplus$	Boolean exclusive or (xor)
Z_i	Round keys	$\otimes$	Multiplication in GF(2^{16})



$A \leftarrow (X_0 \odot Z_0) \oplus (X_2 \boxplus Z_2)$
$B \leftarrow (X_1 \boxplus Z_1) \oplus (X_3 \odot Z_3)$
$D \leftarrow (A \odot Z_4) \boxplus B$

$\Delta \leftarrow D \odot Z_5$
$\Gamma \leftarrow \Delta \boxplus (A \odot Z_4)$
$Y_0 \leftarrow (X_0 \odot Z_0) \oplus \Delta$
$Y_1 \leftarrow (X_2 \boxplus Z_2) \oplus \Delta$
$Y_2 \leftarrow (X_1 \boxplus Z_1) \oplus \Gamma$
$Y_3 \leftarrow (X_3 \odot Z_3) \oplus \Gamma$

Fig. 1. IDEA round function



```
for $0 \le i < w$ do
    $A_i \leftarrow (X_{0,i} \odot Z_{0,i}) \oplus (X_{2,i} \boxplus Z_{2,i})$
    $B_i \leftarrow (X_{1,i} \boxplus Z_{1,i}) \oplus (X_{3,i} \odot Z_{3,i})$
    $D_i \leftarrow (A_i \odot Z_{4,i}) \boxplus B_i$
end for
$D \leftarrow M \cdot D$          ▷ $M$ is an MDS matrix
for $0 \le i < w$ do
    $\Delta_i \leftarrow D_i \odot Z_{5,i}$
    $\Gamma_i \leftarrow \Delta_i \boxplus (A_i \odot Z_{4,i})$
    $Y_{0,i} \leftarrow (X_{0,i} \odot Z_{0,i}) \oplus \Delta_i$
    $Y_{1,i} \leftarrow (X_{2,i} \boxplus Z_{2,i}) \oplus \Delta_i$
    $Y_{2,i} \leftarrow (X_{1,i} \boxplus Z_{1,i}) \oplus \Gamma_i$
    $Y_{3,i} \leftarrow (X_{3,i} \odot Z_{3,i}) \oplus \Gamma_i$
end for
```

Fig. 2. WIDEA round function

The round functions of IDEA and WIDEA are given in Figs. 1 and 2. The important difference between the two is the multiplication by an MDS matrix M over the field $(\mathrm{GF}(2^{16}), \oplus, \otimes)$. This operation is similar to the AES MixColumn operation; it is used for diffusion between the parallel IDEA instances of WIDEA. WIDEA iterates 8 rounds (for all values of w) plus a final half-round for key-whitening:

$$C_{0,i} = X_{0,i} \odot K_{48,i} \quad C_{1,i} = X_{2,i} \odot K_{49,i} \quad C_{2,i} = X_{1,i} \odot K_{50,i} \quad C_{3,i} = X_{3,i} \odot K_{51,i}$$

The key schedule is described over $64 \cdot w$ words. The first 8 words are loaded with the master key K, and the expanded words are computed as:

$$K_i = \left(\left(\left((K_{i-1} \oplus K_{i-8}) \overset{16}{\boxplus} K_{i-5}\right) \overset{16}{\lll} 5\right) \lll 24\right) \oplus R_i,$$

where R_i are round constants. The round keys used for round r are $Z_{i,j} = K_{6r+i,j}$.

2 Truncated Differential Trail

Our attacks are based on a differential trail. We consider a pair of messages P, P' with a small difference encrypted through IDEA under the same key K, and we study the difference in the state X, X' after each round. However, we don't specify exactly the difference: we only specify for each word whether the difference is zero or non-zero, giving a truncated differential trail.

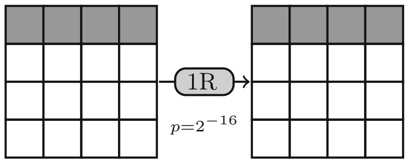
We start with a pair of states with only one IDEA instance active, such as (with instance 0 active):

$$\begin{array}{lllll} X_{0,0} \neq X'_{0,0} & X_{1,0} \neq X'_{1,0} & X_{2,0} \neq X'_{2,0} & X_{3,0} \neq X'_{3,0} & \\ X_{0,i} = X'_{0,i} & X_{1,i} = X'_{1,i} & X_{2,i} = X'_{2,i} & X_{3,i} = X'_{3,i} & \text{for } 1 \leq i < w \end{array}$$

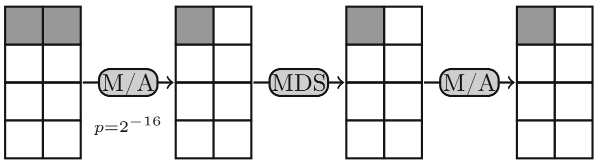
When we compute the round function, we have $D_i = D'_i$ for $i \neq 0$, and with probability 2^{-16}, we also have $D_0 = D'_0$. In this case, the input of the MDS matrix will be inactive, and the difference does not propagate to the other IDEA instances. This leads to:

$$\begin{array}{lllll} Y_{0,0} \neq Y'_{0,0} & Y_{1,0} \neq Y'_{1,0} & Y_{2,0} \neq Y'_{2,0} & Y_{3,0} \neq Y'_{3,0} & \\ Y_{0,i} = Y'_{0,i} & Y_{1,i} = Y'_{1,i} & Y_{2,i} = Y'_{2,i} & Y_{3,i} = Y'_{3,i} & \text{for } 1 \leq i < w \end{array}$$

Graphically, we can represent the state X as a matrix of 16-words, with active words in black and inactive words in white:

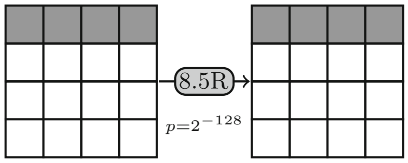


The trail inside the Multiply/Add/Diffuse box is:



The MDS matrix is applied to the right column, and all inputs are inactive.

We iterate this trail for 8 rounds of WIDEA, and the final half round does not affect which words are active. This gives a truncated differential trail with probability 2^{-128} for the full 8.5 rounds of WIDEA:



For a random permutation over $64 \cdot w$ bits, a pair would follow this trail with probability $2^{-64\cdot(w-1)}$. Therefore, we have an efficient distinguisher for WIDEA-w as soon as $w \geq 3$. If $w \geq 4$, the distinguisher is very strong, and we do not expect to have any false positives.

3 Distinguisher

To exploit this property, we use structures of 2^{64} plaintext, where one slice takes all possible values, and the other slices are fixed to a constant value. This structure gives $2^{64} \times (2^{64} - 1)/2 \approx 2^{127}$ pairs of plaintext; each pair has only one active slice and is a potential candidate for the differential trail. If we take two such structures, this gives about 2^{128} plaintext pairs, and with a probability of $1 - 1/e \approx 63\,\%$, at least one pair will follow the truncated trail.

We can efficiently test if such pairs are present by inserting all the ciphertexts in a hash table indexed by the slices that are expected to be inactive. This gives a chosen-plaintext distinguisher for WIDEA with complexity 2^{65} as shown in Algorithm 1.

Algorithm 1. Distinguish WIDEA from a random permutation

```
Input: $E$
  for $0 \leq t < 2$ do
      $T \leftarrow \varnothing$
      $X \leftarrow$ Rand()
      for all $X_{0,0}, X_{0,1}, X_{0,2}, X_{0,3}$ do
          $Y \leftarrow E(X)$
          $Y' \leftarrow Y_{1,0\ldots3} \| Y_{2,0\ldots3} \| \ldots \| Y_{w-1,0\ldots3}$
          if $Y' \in T$ then
              return WIDEA                              ▷ $(T\{Y'\}, X)$ is a right pair.
          end if
          $T\{Y'\} \leftarrow X$
      end for
  end for
  return Random
```

4 Key Recovery

We can turn this simple distinguisher into a full key recovery with some more effort.

4.1 First-Round Key

We consider a right pair (X, X'), and we study the internal state; we can express D_0, D'_0:

$$D_0 = \Big(((X_{0,0} \odot Z_{0,0}) \oplus (X_{2,0} \boxplus Z_{2,0})) \odot Z_{4,0} \Big) \boxplus ((X_{1,0} \boxplus Z_{1,0}) \oplus (X_{3,0} \odot Z_{3,0}))$$

$$D'_0 = \Big(((X'_{0,0} \odot Z_{0,0}) \oplus (X'_{2,0} \boxplus Z_{2,0})) \odot Z_{4,0} \Big) \boxplus ((X'_{1,0} \boxplus Z_{1,0}) \oplus (X'_{3,0} \odot Z_{3,0}))$$

Since the pair follows the trail, we have $D_0 = D'_0$, or equivalently:

$$\begin{aligned}\Big(\big((X_{0,0} \odot Z_{0,0}) \oplus (X_{2,0} \boxplus Z_{2,0})\big) \odot Z_{4,0}\Big) \boxminus \Big(\big((X'_{0,0} \odot Z_{0,0}) \oplus (X'_{2,0} \boxplus Z_{2,0})\big) \odot Z_{4,0}\Big) \\ = \big((X'_{1,0} \boxplus Z_{1,0}) \oplus (X'_{3,0} \odot Z_{3,0})\big) \boxminus \big((X_{1,0} \boxplus Z_{1,0}) \oplus (X_{3,0} \odot Z_{3,0})\big) \qquad (1)\end{aligned}$$

In this equation, the left hand side is a function of $Z_{0,0}, Z_{2,0}, Z_{4,0}$ only, while the right hand size is a function of $Z_{1,0}, Z_{3,0}$ only. We denote them as:

$$\begin{aligned}F_i(X, X', Z_{0,i}, Z_{2,i}, Z_{4,i}) &= \Big(\big((X_{0,i} \odot Z_{0,i}) \oplus (X_{2,i} \boxplus Z_{2,i})\big) \odot Z_{4,i}\Big) \\ &\quad \boxminus \Big(\big((X'_{0,i} \odot Z_{0,i}) \oplus (X'_{2,i} \boxplus Z_{2,i})\big) \odot Z_{4,i}\Big) \qquad (2) \\ G_i(X, X', Z_{1,i}, Z_{3,i}) &= \big((X'_{1,i} \boxplus Z_{1,i}) \oplus (X'_{3,i} \odot Z_{3,i})\big) \\ &\quad \boxminus \big((X_{1,i} \boxplus Z_{1,i}) \oplus (X_{3,i} \odot Z_{3,i})\big). \qquad (3)\end{aligned}$$

We can recover the key efficiently using a meet-in-the-middle technique. On the one hand, we compute $F_0(X, X', k_0, k_2, k_4)$ for all k_0, k_2, k_4, and on the other hand, we compute $G_0(X, X', k_1, k_3)$ for all k_1, k_3. Then we look for matches in the list because the correct key satisfies $F_0(X, X', Z_{0,0}, Z_{2,0}, Z_{4,0}) = G_0(X, X', Z_{1,0}, Z_{3,0})$ for a right pair X, X'.

In order to achieve a strong filtering, we use several right pairs $X^{(j)}, X'^{(j)}$ and we look for simultaneous matches between all the F's and G's, i.e. matches in the concatenations $||_{j=0}^{k} F_0(X^{(j)}, X'^{(j)}, k_0, k_2, k_4)$ and $||_{j=0}^{k} G_0(X^{(j)}, X'^{(j)}, k_1, k_3)$. Unfortunately, this filtering cannot distinguish the real key K, and the key K' where the most significant bit of $Z_{1,0}$ is flipped, because the effect of this bit on D is linear.

Each pair gives a 16-bit filtering and we are recovering 79 key bits, so we expect that 5 pairs would be sufficient. However, when implementing the attack, we found out that the filtering given by each pair is not independent, and we need more than 5 pairs; our experiments show that using $k = 8$ pairs is enough to isolate a single key pair most of time.

Therefore we can recover the correct value of $Z_{0\ldots4,0}$ (up to one bit) with complexity $2^{3\cdot16} = 2^{51}$ (we consider that the computation of 8 F and G functions costs about the same as one evaluation of WIDEA). We can also recover the keys $Z_{0\ldots4,i}$ used in the other IDEA instances in the same way: we just need different pairs following a path with another active slice.

4.2 Second-Round Key

We can now compute all the inputs to the MDS matrix in the first round, since we know the keys used in each IDEA instance. Then we compute the output of the MDS matrix, and we can again consider the parallel IDEA instances independently. First, we guess $Z_{5,i}$ in order to compute the state after the end of the first round. Then we apply the same meet-in-the-middle strategy as for the first round, in order to recover the second round key $Z_{6\ldots10,i}$. Finally, we know that the master key is $Z_{0...7}$ according to the key expansion algorithm. If several key candidates remain, we test them with one of the plaintext/ciphertext pairs. This would give a key-recovery with complexity $w \cdot 2^{16} \cdot 2^{48} = w \cdot 2^{64}$.

Missing Key Bits. In fact, we have 2^w key candidates for the first round, because the most significant bits of the $Z_{1,i}$'s can not be recovered by testing collisions in D. Instead of running the analysis for the second round with all these candidates, we use the fact that the unknown bits have a linear effect on the MDS operation. Moreover, the coefficients of the MDS matrix given in [9] for WIDEA-8 are all between 1 and 9; therefore any linear combination is between 0 and 15. For WIDEA-4, the coefficients are between 1 and 3; any linear combination is between 0 and 3. Instead of guessing the w missing bits of the key, we can guess the effect on the MDS output, which is of the form $t \otimes \texttt{0x8000}$, with $0 \leq t < 4$ for WIDEA-4 and $0 \leq t < 16$ for WIDEA-8 (i.e. a 2-bit guess and a 4-bit guess, respectively). Therefore the actual complexity of the key-recovery attack will be $4 \cdot 2^2 \cdot 2^{64} = 2^{68}$ for WIDEA-4, and $8 \cdot 2^4 \cdot 2^{64} = 2^{71}$ for WIDEA-8. The attack is described in Algorithms 2 and 3.

4.3 Complexity

We can slightly reduce the complexity using properties of the key schedule. More precisely, when $K_{6...10,0}$ has been recovered in the first IDEA instance, we can use the key scheduling algorithm to compute some bits of K_8, so that recovering the key of the next instance become negligible compared to the first key recovery. This reduces the complexity by a factor w.

Therefore, the analysis step has a complexity of only 2^{66} memory accesses to a table of size 2^{32} for WIDEA-4 (2^{68} accesses to a similar table for WIDEA8). The computation of the F's and G's will likely be negligible before the cost of memory accesses. As a rough estimation we can assume that a memory access to a table of size 2^{32} takes about the same time as the computation of the block cipher.

Data Complexity. The data complexity of the attack is $w \cdot 2^{68}$: we need $8 \cdot w$ right pairs, and each pair is found after 2^{65} chosen plaintexts. The data filtering step to isolate right pairs is actually the most expensive step of the attack: it requires $w \cdot 2^{68}$ memory accesses to a table of size 2^{64}.

5 Reducing the Memory Cost

Since the complexity of the key-recovery attacks on WIDEA is rather low, we briefly discuss practical aspects of the attack, in addition to the complexity figures which don't account for the cost of the memory. The bottleneck of the attack is the filtering of right pairs. If we use a hash table to find collisions in each structures as explained in Sect. 3, we need a random access memory of size 2^{64}, which is probably less practical than the time complexity of 2^{66} or 2^{68} for the analysis step.

Algorithm 2. Recover the Key from WIDEA

```
Input: $(X^{(i,j)}, X'^{(i,j)})$ right pairs with slice $i$ active        $0 \leq i < w$, $0 \leq j < k = 8$
  ▷ First step: recover $K_{0...4}$
  for $0 \leq i < w$ do
      $T \leftarrow \varnothing$
      for all $k_1, k_3$ do
          $G \leftarrow ||_{j=0}^{k} G_i(X^{(i,j)}, X'^{(i,j)}, k_1, k_3)$
          $T\{G\} \leftarrow (k_1, k_3)$
      end for
      for all $k_0, k_2, k_4$ do
          $F \leftarrow ||_{j=0}^{k} F_i(X^{(i,j)}, X'^{(i,j)}, k_0, k_2, k_4)$
          if $F \in T$ then
              $k_1, k_3 \leftarrow T\{F\}$
              $K_{0...4,i} \leftarrow k_0, k_1, k_2, k_3, k_4$
          end if
      end for
  end for
  ▷ Second step: recover $K_{5...10}$
  for $0 \leq i < w$ do
      for all $k_5, 0 \leq t < 16$ do
          $K_{5,i} \leftarrow k_5$
          for all $i, k$ do
              $Y^{\ i,k} \leftarrow \textsc{RoundTweak}(X^{\ (i,k)}, K, i, t \otimes \texttt{0x8000})$
              $Y'^{i,k} \leftarrow \textsc{RoundTweak}(X'^{(i,k)}, K, i, t \otimes \texttt{0x8000})$
          end for
          $T \leftarrow \varnothing$
          for all $k_1, k_3$ do
              $G \leftarrow ||_{j=0}^{k} G_i(Y^{(i,j)}, Y'^{(i,j)}, k_1, k_3)$
              $T\{G\} \leftarrow (k_1, k_3)$
          end for
          for all $k_0, k_2, k_4$ do
              $F \leftarrow ||_{j=0}^{k} F_i(Y^{(i,j)}, Y'^{(i,j)}, k_0, k_2, k_4)$
              if $F \in T$ then
                  $k_1, k_3 \leftarrow T\{F\}$
                  $K_{6...10,i} \leftarrow k_0, k_1, k_2, k_3, k_4$
              end if
          end for
      end for
  end for
```

Algorithm 3. WIDEA round with a tweak (ι, t) after the MDS step

```
▷ $t$ is the effect of the missing bits of $Z_1$ on $D_\iota$
function RoundTweak($X$, $Z$, $\iota$, $t$)
    for $0 \leq i < w$ do
        $A_i \leftarrow (X_{0,i} \odot Z_{0,i}) \oplus (X_{2,i} \boxplus Z_{2,i})$
        $B_i \leftarrow (X_{1,i} \boxplus Z_{1,i}) \oplus (X_{3,i} \odot Z_{3,i})$
        $D_i \leftarrow (A_i \odot Z_{4,i}) \boxplus B_i$
    end for
    $D \leftarrow M \cdot D$
    $D_\iota \leftarrow D_\iota \oplus t$
    for $0 \leq i < w$ do
        $\Delta_i \leftarrow D_i \odot Z_{5,i}$
        $\Gamma_i \leftarrow \Delta_i \boxplus (A_i \odot Z_{4,i})$
        $Y_{0,i} \leftarrow (X_{0,i} \odot Z_{0,i}) \oplus \Delta_i$
        $Y_{1,i} \leftarrow (X_{2,i} \boxplus Z_{2,i}) \oplus \Delta_i$
        $Y_{2,i} \leftarrow (X_{1,i} \boxplus Z_{1,i}) \oplus \Gamma_i$
        $Y_{3,i} \leftarrow (X_{3,i} \odot Z_{3,i}) \oplus \Gamma_i$
    end for
    return $Y$
end function
```

5.1 Sorting

A first way to avoid this problem is to store all the ciphertexts from a structure sequentially, and to run a sorting algorithm to find collisions. This still requires a memory of size 2^{64}, but we only make sequential accesses to this huge memory, and we can use disk or tape storage. The sorting algorithm increases the cost of the attack by a logarithmic factor, but the resulting attack will be much easier to carry out in practice.

The storage needed for the attack will be about 2^{64} elements of 16 bytes each (4 16-bit words for the active input slice, and the output can be restricted to 4 16-bit words if we use an extra pass to check that the rest of the state collides). This amounts to 2^{68} bytes, or 256 exabytes.

5.2 Time-Memory Trade-Offs

We can also use a time-memory trade-off to reduce the memory requirement of the attack. The filtering step of the attack is essentially a collision search for the function

$$\begin{aligned} \phi_r \colon \{0,1\}^{64} &\to \{0,1\}^{64\cdot(w-1)} \\ x &\mapsto \mathrm{Trunc}_{64\cdot(w-1)}(E_K(x\|r)) \end{aligned}$$

with a random $r \in \{0,1\}^{64\cdot(w-1)}$. If we truncate the output of ϕ_r to 64 bits, we can find collisions with a memory-less algorithm for a complexity of 2^{32}, using adaptively chosen inputs. However, we expect on average that 2^{63} collisions exists for this function, but only 0.5 collisions correspond to a right pair for the differential trail. Therefore the total complexity to find a right pair without memory will be 2^{96}.

More generally, we can store distinguished points so that finding N collisions costs less than $N \cdot 2^{32}$. Using the analysis of [15], we know that we can find a "golden collision" with a complexity of $2 \cdot 2.5 \cdot 2^{64+t/2}$ if we have a memory of size 2^{64-t} (with $0 < t < 64$).

6 Hash Function Collisions

One important use case of WIDEA as envisioned by the designers is to build a hash function. Hash function benchmarks are given in [9], and a more complete description of a hash function (named HIDEA) was presented in the ESC seminar [8]. HIDEA uses WIDEA to build a compression function with the Davies-Meyer mode, and iterates it with the Merkle-Damgård and HAIFA modes of operation. We note that the presentation of HIDEA in [8] suggests to use a 10.5-round WIDEA, instead of the 8.5-round version of [9].

To find collisions for HIDEA, we first look for a pair of messages M, M' so that the internal state X, X' reached after processing them satisfies $X_{0\ldots3,i} = X'_{0\ldots3,i}$ for $i \neq 0$. This is equivalent to finding a collision in a truncated function with an output of $64 \cdot (w-1)$ bits. For this step we can store the hash of $2^{32\cdot(w-1)}$ random messages, or use a memory-less collision finding algorithm. This step has a complexity of $2^{32\cdot(w-1)}$, i.e. 2^{224} for WIDEA-8. We note that we can just as easily have two pre-specified prefixes P and P' and look for M, M' such that the state X, X' reached after processing $P\|M$ and $P'\|M'$ satisfies $X_{0\ldots3,i} = X'_{0\ldots3,i}$ for $i \neq 0$.

We assume that $P\|M$ and $P'\|M'$ both have the same length, and this length is an integral number of blocks. When we append a random block N to $P\|M$ and $P'\|M'$, the compression function is computed as:

$$\begin{aligned} h(X,N) &= Y \oplus X, & Y &= E_N(X) \\ h(X',N) &= Y' \oplus X', & Y' &= E_N(X'). \end{aligned}$$

We have $X_{0\ldots3,i} = X'_{0\ldots3,i}$ for $i \neq 0$, and we know that with probability 2^{-128} (2^{-160} for a 10.5-round WIDEA), this gives $Y_{0\ldots3,i} = Y'_{0\ldots3,i}$ for $i \neq 0$. Additionally, we have $Y_{0\ldots3,0} \oplus X_{0\ldots3,0} = Y'_{0\ldots3,0} \oplus X'_{0\ldots3,0}$ with probability 2^{-64}. Therefore we have $h(X,N) = h(X',N)$ with probability 2^{-192} (2^{-224} for a 10.5-round WIDEA).

When combining both steps, we have a collision attack with complexity 2^{224} for WIDEA-8 with up to 10.5 rounds. The attack is described by Algorithm 4.

Surprisingly, this attack doesn't use any property of the key schedule, and can use arbitrary messages. This allows to build meaningful collisions easily. On the other hand, a few more rounds can be attacked using message modification techniques, if needed.

Algorithm 4. Find collisions for HIDEA-512

```
Input: $P, P'$ chosen prefix
  Find $M, M'$ with $\mathrm{Trunc}_{64(w-1)}(H(P\|M)) = \mathrm{Trunc}_{64(w-1)}(H(P'\|M'))$
  ▷ Complexity $2^{224}$
  repeat
      $N \leftarrow \mathsf{Rand}()$
  until $H(P\|M\|N) = H(P'\|M'\|N)$
  ▷ Complexity $2^{224}$
```

7 Conclusion

In this paper we show devastating attacks on the WIDEA block cipher. Our main result is a key-recovery attack with complexity $w \cdot 2^{68}$ for the WIDEA family with $w \geq 4$. In particular this affects the main instances considered in the WIDEA paper: WIDEA-4 (256-bit block and 512-bit key) and WIDEA-8 (512-bit block and 1024-bit key). We also show a collision attack when WIDEA is used to build a hash function, as was proposed by the designers. The collision attack affects instances with $w \geq 8$: we can build collisions for HIDEA-512 (based on WIDEA-8) with a complexity of 2^{224}.

The attacks exploit the limited diffusion between the IDEA instances by building trails where the MDS diffusion layer is never active. Since the input of the MDS layer is only 16-bit for one IDEA instance, such trails have a probability of $2^{-16 \cdot r}$ for an r-round WIDEA. In addition, we use structures of plaintext to reduce the data complexity of the block-cipher attacks. The attacks don't depend on low-level details of the design (such as the key schedule, the MDS matrix, or the exact computational graph of IDEA). The complexity is almost independent of the width w, and can even break extended version of WIDEA with more than 8.5 rounds.

We have implemented the key-recovery attack with a reduced WIDEA using 8-bit words, and all the steps of the attack worked as expected.

Acknowledgment. We would like to thanks the anonymous reviewers for very detailed comments. In particular, they noticed a mistake in our description of IDEA and WIDEA (our implementation of the attack used the correct algorithm, though).

The author is supported by the ERC project CRASH. Part of this work was done while the author was at the university of Luxembourg, supported by the AFR grant PDR-10-022 of the FNR.

References

1. Biham, E., Dunkelman, O., Keller, N.: A new attack on 6-round IDEA. In: Biryukov, A. (ed.) FSE 2007. LNCS, vol. 4593, pp. 211–224. Springer, Heidelberg (2007)
2. Biham, E., Dunkelman, O., Keller, N.: A unified approach to related-key attacks. In: Nyberg, K. (ed.) FSE 2008. LNCS, vol. 5086, pp. 73–96. Springer, Heidelberg (2008)

3. Biham, E., Dunkelman, O., Keller, N., Shamir, A.: New data-efficient attacks on 6-round IDEA. Cryptology ePrint Archive, Report 2011/417 (2011). http://eprint.iacr.org/
4. Biryukov, A., Nakahara Jr, J., Preneel, B., Vandewalle, J.: New weak-key classes of IDEA. In: Deng, R.H., Qing, S., Bao, F., Zhou, J. (eds.) ICICS 2002. LNCS, vol. 2513, pp. 315–326. Springer, Heidelberg (2002)
5. Daemen, J., Govaerts, R., Vandewalle, J.: Weak keys for IDEA. In: Stinson, D.R. (ed.) CRYPTO 1993. LNCS, vol. 773, pp. 224–231. Springer, Heidelberg (1994)
6. Hawkes, P.: Differential-linear weak key classes of IDEA. In: Nyberg, K. (ed.) EUROCRYPT 1998. LNCS, vol. 1403, pp. 112–126. Springer, Heidelberg (1998)
7. Nakahara Jr, J.: Differential and linear attacks on the full WIDEA-n block ciphers (under Weak Keys). In: Pieprzyk, J., Sadeghi, A.-R., Manulis, M. (eds.) CANS 2012. LNCS, vol. 7712, pp. 56–71. Springer, Heidelberg (2012)
8. Junod, P.: IDEA: past, present, and future. Early Symmetric Crypto (2010). https://www.cryptolux.org/esc2010/Pascal_Junod
9. Junod, P., Macchetti, M.: Revisiting the IDEA Philosophy. In: Dunkelman, O. (ed.) FSE 2009. LNCS, vol. 5665, pp. 277–295. Springer, Heidelberg (2009)
10. Khovratovich, D., Leurent, G., Rechberger, C.: Narrow-Bicliques: cryptanalysis of full IDEA. In: Pointcheval, D., Johansson, T. (eds.) EUROCRYPT 2012. LNCS, vol. 7237, pp. 392–410. Springer, Heidelberg (2012)
11. Lai, X., Massey, J.L.: A proposal for a new block encryption standard. In: Damgård, I.B. (ed.) EUROCRYPT 1990. LNCS, vol. 473, pp. 389–404. Springer, Heidelberg (1991)
12. Lai, X., Massey, J.L.: Markov ciphers and differential cryptanalysis. In: Davies, D.W. (ed.) EUROCRYPT 1991. LNCS, vol. 547, pp. 17–38. Springer, Heidelberg (1991)
13. Mendel, F., Rijmen, V., Toz, D., Varıcı, K.: Collisions for the WIDEA-8 compression function. In: Dawson, E. (ed.) CT-RSA 2013. LNCS, vol. 7779, pp. 162–173. Springer, Heidelberg (2013)
14. Sun, X., Lai, X.: The key-dependent attack on block ciphers. In: Matsui, M. (ed.) ASIACRYPT 2009. LNCS, vol. 5912, pp. 19–36. Springer, Heidelberg (2009)
15. van Oorschot, P.C., Wiener, M.J.: Parallel collision search with application to hash functions and discrete logarithms. In: Denning, D.E., Pyle, R., Ganesan, R., Sandhu, R.S. (eds.) ACM Conference on Computer and Communications Security, pp. 210–218. ACM (1994)
16. Vaudenay, S.: On the Lai-Massey scheme. In: Lam, K.-Y., Okamoto, E., Xing, Ch. (eds.) ASIACRYPT 1999. LNCS, vol. 1716, pp. 8–19. Springer, Heidelberg (1999)
17. Wei, L., Peyrin, T., Sokołowski, P., Ling, S., Pieprzyk, J., Wang, H.: On the (In)security of IDEA in various hashing modes. In: Canteaut, A. (ed.) FSE 2012. LNCS, vol. 7549, pp. 163–179. Springer, Heidelberg (2012)