Revisiting the IDEA Philosophy

Pascal Junod[1,2] and Marco Macchetti[1]

[1] Nagracard SA
CH-1033 Cheseaux-sur-Lausanne, Switzerland
[2] University of Applied Sciences Western Switzerland (HES-SO/HEIG-VD)
CH-1401 Yverdon-les-Bains, Switzerland

Abstract. Since almost two decades, the block cipher IDEA has resisted an exceptional number of cryptanalysis attempts. At the time of writing, the best published attack works against 6 out of the 8.5 rounds (in the non-related-key attacks model), employs almost the whole codebook, and improves the complexity of an exhaustive key search by a factor of only two. In a parallel way, Lipmaa demonstrated that IDEA can benefit from SIMD (Single Instruction, Multiple Data) instructions on high-end CPUs, resulting in very fast implementations. The aim of this paper is two-fold: first, we describe a parallel, time-constant implementation of eight instances of IDEA able to encrypt in counter mode at a speed of 5.42 cycles/byte on an Intel Core2 processor. This is comparable to the fastest stream ciphers and notably faster than the best known implementations of most block ciphers on the same processor. Second, we propose the design of a new block cipher, named WIDEA, leveraging on IDEA's outstanding security-performance ratio. We furthermore propose a new key-schedule algorithm in replacement of completely linear IDEA's one, and we show that it is possible to build a compression function able to process data at a speed of 5.98 cycles/byte. A significant property of WIDEA is that it closely follows the security rationales defined by Lai and Massey in 1990, hence inheriting all the cryptanalysis done the past 15 years in a very natural way.

Key words: IDEA block cipher, WIDEA compression function, Intel Core2 CPU, wordslice implementation

1 Introduction

Finding the proper balance between security and speed has always been a nontrivial challenge for designers of block ciphers and hash functions. One possibility consists in using low-footprint arithmetical operations, like simple Boolean operators (usually AND, OR or XOR), table lookups or modular additions, to build a rather fast round function. Since the strength of such round function is usually low in cryptographic terms, one is forced to iterate it a sufficient number of times to get a proper security level. Another approach consists in using more complicated arithmetical operations, like multiplications, for instance. The inherent larger cryptographic strength of such operations naturally comes with a slower speed.

AES [15,37] is maybe one of the most elegant balance between efficiency and security for a 128-bit block size. By using quite strong diffusion and confusion elements in a design having a high internal parallelism, Daemen and Rijmen have obtained a very fast cipher while keeping a reasonable security margin. However, as a matter of fact, it is interesting to note that several designs of hash functions submitted to the NIST SHA3 competition (e.g., Skein [19] and MD6 [41]) have deliberately chosen to use much simpler operations to build a "light" round and to iterate this round function a large number of times. This approach is preferred by some designers because it allows low-footprint hardware implementations as well as an easier cryptanalysis.

The IDEA block cipher [26, 27] was designed in the beginning of the 90's with the following philosophy in mind: mix three different and algebraically incompatible operations. As a result, a rather strong round function is iterated no more than 8 times to build a cipher with an outstanding security record: almost 20 years after its design, no faster attack than an exhaustive key search is known against its full flavor, despite a very intense cryptanalysis activity resulting in more than a dozen of academic papers discussing its security. However, as of today, IDEA has more been known for its security than for its speed, even if some fast implementations have been proposed by Lipmaa [29], exploiting the Intel MMX instruction set. Furthermore, it is well-known that the implementation of the so-called IDEA multiplication is rather delicate and, if not done properly, is prone to timing attacks [23].

Related Work How to increase the block size of IDEA has been studied by Nakahara and co-authors: they proposed the MESH ciphers [36], which are ciphers relying on the same operations than in IDEA, but having block sizes of up to 128 bits as well as a stronger key-schedule algorithm. Other variants of MESH ciphers, targeting 8-bit microcontrollers, were described in [32], always exploiting the same philosophy but this times with operations working on 8-bit variables.

Our Contributions Attacking the common belief that IDEA is rather a slow cipher, we show that, by properly exploiting modern instruction sets, it is one of the fastest block ciphers available on the market on the `x86` and `x86-64` architectures, beating AES by a large margin, and resulting in a formidable security-speed ratio supported by almost 20 years of unsuccessful cryptanalysis. In this paper, we revisit the IDEA philosophy (iterating only a modest number of times a relatively strong round function) at the light of the latest multimedia instruction sets SSE, SSE2 and SSE3 which are available today in virtually every PC. Our contributions are double fold: first, we exhibit a so-called *wordslice* implementation of eight parallel instances of IDEA able to encrypt in counter mode at a speed of 5.42 clock cycles per byte, according to the eSTREAM benchmarking framework, on an Intel Core2[3] CPU. Our implementation is notably more

[3] Precisely, the CPU belongs to family 6, model 23, stepping 6.

than 30% faster than the fastest known implementation of AES [22], measured at 7.81 cycles/byte on the same CPU, while it is able to handle as few as 64 bytes of data, compared to the 128 bytes of the implementation of Käsper and Schwabe. For the sake of completeness, we note that the fastest standard (i.e., non-bitslice) implementation of AES has been recently reported to encrypt at a rate of 10.57 cycles/byte on an Intel Core2 CPU [2]. Additionally, our implementation does not suffer from cache attacks [1, 39], is completely branch-free, and is therefore time-constant.

Our second contribution is the design of a new block cipher family named WIDEA. It relies on n parallel instances of IDEA mixed using a high-quality diffusion element. We discuss the rationales behind our design, its security and concretely specify the WIDEA-8 instance that operates on 512-bit data blocks. By applying the Davies-Meyer mode of operation, we turn WIDEA-8 into a compression function capable of processing data at 5.98 cycles/byte on the same Intel Core2 CPU.

2 The IDEA Block Cipher

In this section, we first recall the specifications of IDEA and we discuss the available literature dedicated to its security.

2.1 Overview of the Cipher

The IDEA block cipher handles 64-bit blocks of data under a 128-bit key. It consists of 8 identical rounds (Fig. 1 illustrates one round), each parametered by a 96-bit subkey, followed by a final key-whitening layer, often named *half round*. The r-th round transforms a 64-bit data input interpreted as a vector of four 16-bit words $(X_0^{(r)}, X_1^{(r)}, X_2^{(r)}, X_3^{(r)})$ to an output vector $(Y_0^{(r)}, Y_1^{(r)}, Y_2^{(r)}, Y_3^{(r)})$ having a similar shape. This process is keyed by six 16-bit subkeys denoted $Z_j^{(r)}$ with $0 \leq j \leq 5$ derived from the 128-bit master key according to a rather simple, bit-selecting (and therefore completely linear) key-schedule (see Fig. 2). The strength of IDEA is certainly due to an elegant design approach which consists in mixing three algebraically incompatible group operations: the addition over GF(2^{16}), denoted $\oplus$, the addition over $\mathbb{Z}_{2^{16}}$, denoted $\boxplus$, and the multiplication over $\mathbb{Z}^*_{2^{16}+1}$, denoted $\odot$, where 0 represents the value 2^{16}. Round r is defined by the following operations: one first computes two intermediate values

$$\alpha^{(r)} = \left(X_0^{(r)} \odot Z_0^{(r)}\right) \oplus \left(X_2^{(r)} \boxplus Z_2^{(r)}\right) \text{ and } \beta^{(r)} = \left(X_1^{(r)} \boxplus Z_1^{(r)}\right) \oplus \left(X_3^{(r)} \odot Z_3^{(r)}\right).$$

These two values form the input of the *multiplication-addition box* (MA-box) which results in

$$\delta^{(r)} = \left(\left(\alpha^{(r)} \odot Z_4^{(r)}\right) \boxplus \beta^{(r)}\right) \odot Z_5^{(r)} \text{ and } \gamma^{(r)} = \left(\alpha^{(r)} \odot Z_4^{(r)}\right) \boxplus \delta^{(r)}.$$

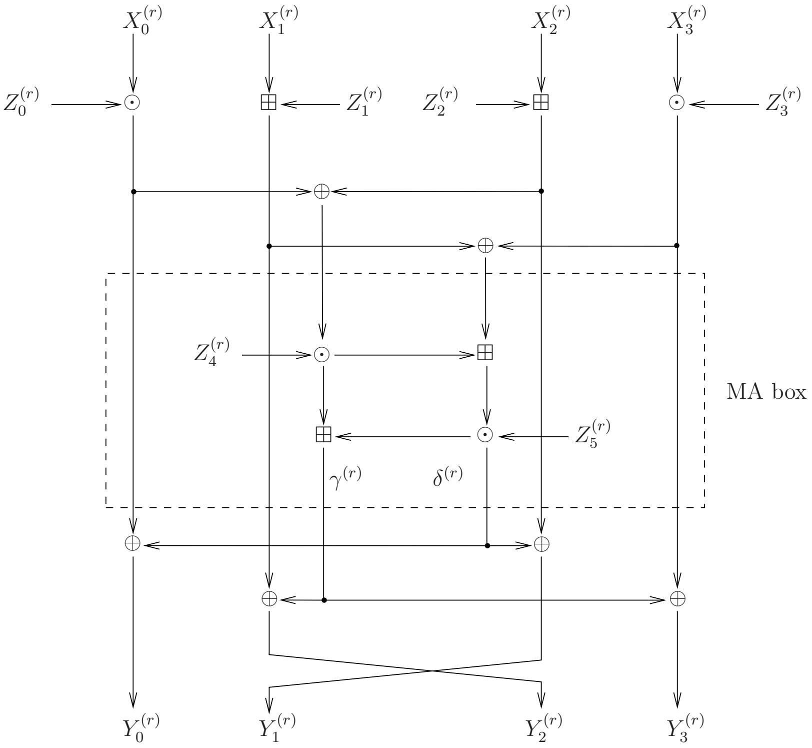

Fig. 1. Round r of IDEA

The output of i-th round is then obtained through

$$\begin{aligned} Y_0^{(r)} &= \left(X_0^{(r)} \odot Z_0^{(r)}\right) \oplus \delta^{(r)}, \quad Y_1^{(r)} = \left(X_2^{(r)} \boxplus Z_2^{(r)}\right) \oplus \delta^{(r)}, \\ Y_2^{(r)} &= \left(X_1^{(r)} \boxplus Z_1^{(r)}\right) \oplus \gamma^{(r)} \text{ and } Y_3^{(r)} = \left(X_3^{(r)} \odot Z_3^{(r)}\right) \oplus \gamma^{(r)} \end{aligned}$$

After the 8-th round, a final key-whitening layer is applied:

$$Y_0^{(9)} = X_0^{(9)} \odot Z_0^{(9)}, Y_1^{(9)} = X_2^{(9)} \boxplus Z_1^{(9)}, Y_2^{(9)} = X_1^{(9)} \boxplus Z_2^{(9)} \text{ and } Y_3^{(9)} = X_3^{(9)} \odot Z_3^{(9)}.$$

2.2 Cryptanalysis of IDEA

Designed to offer resistance to differential cryptanalysis [10], the IDEA block cipher has been subject to a very intense scrutiny by the cryptologic community since its publication in 1990. This is probably due to the fact that, at the time of writing, the best attack ever designed against IDEA in a classical scenario is able

Round i	$Z_1^{(i)}$	$Z_2^{(i)}$	$Z_3^{(i)}$	$Z_4^{(i)}$	$Z_5^{(i)}$	$Z_6^{(i)}$
1	$Z_{[0...15]}$	$Z_{[16...31]}$	$Z_{[32...47]}$	$Z_{[48...63]}$	$Z_{[64...79]}$	$Z_{[80...95]}$
2	$Z_{[96...111]}$	$Z_{[112...127]}$	$Z_{[25...40]}$	$Z_{[41...56]}$	$Z_{[57...72]}$	$Z_{[73...88]}$
3	$Z_{[89...104]}$	$Z_{[105...120]}$	$Z_{[121...8]}$	$Z_{[9...24]}$	$Z_{[50...65]}$	$Z_{[66...81]}$
4	$Z_{[82...97]}$	$Z_{[98...113]}$	$Z_{[114...1]}$	$Z_{[2...17]}$	$Z_{[18...33]}$	$Z_{[34...49]}$
5	$Z_{[75...90]}$	$Z_{[91...106]}$	$Z_{[107...122]}$	$Z_{[123...10]}$	$Z_{[11...26]}$	$Z_{[27...42]}$
6	$Z_{[43...58]}$	$Z_{[59...74]}$	$Z_{[100...115]}$	$Z_{[116...3]}$	$Z_{[4...19]}$	$Z_{[20...35]}$
7	$Z_{[36...51]}$	$Z_{[52...67]}$	$Z_{[68...83]}$	$Z_{[84...99]}$	$Z_{[125...12]}$	$Z_{[13...28]}$
8	$Z_{[29...44]}$	$Z_{[45...60]}$	$Z_{[61...76]}$	$Z_{[77...92]}$	$Z_{[93...108]}$	$Z_{[109...124]}$
9	$Z_{[22...37]}$	$Z_{[38...53]}$	$Z_{[54...69]}$	$Z_{[70...85]}$		

Fig. 2. Complete key-schedule of IDEA. $Z_{[0..15]}$ denotes the bits 0 to 15 (inclusive) of Z, $Z_{[117..4]}$ means the bits 117-127 and 0-4 of Z, and the leftmost bit of Z has the index 0.

to break only 6 out of the 8.5 rounds at a $2^{126.8}$ computational cost: breaking the full version of IDEA might be considered by certain cryptanalysts as a kind of “Holy Grail”. We now make a review of the available literature dedicated to the cryptanalysis of IDEA.

Classical Attacks Differential cryptanalysis [10] has been one of the first technique to be tried against IDEA by Meier [31] to break up to 2.5 rounds faster than an exhaustive search. Borst et al. [12] were able to break using a differential-linear attack and 3.5 rounds using truncated differentials. Biham et al. [5] used impossible differentials to break 4.5 rounds. Another approach to break IDEA, based on integral attacks, has been proposed by Nakahara et al. [33] against 2.5 rounds. The approach has first been pushed to 4 rounds by Demirci [17], and then to 5 rounds by Demirci et al. [18] in combination with meet-in-the-middle techniques. Inspired by a work of Nakahara et al. [35], Junod [21] presented several efficient attacks mixing the Biryukov-Demirci relation and square attacks against up to 3.5 rounds. More recently, Biham et al. [7] described a linear attack on 5-round IDEA improving the complexity of [18]. The same authors presented the first attack against 6 rounds in [8] employing almost the whole codebook and having a computational complexity of $2^{126.8}$ operations.

In the related-key setting, an attack against 6.5 rounds was proposed by Biham et al. in [6], 7.5 rounds were reached by the same authors in [8] and recently, an attack working against $4r$-round IDEA for any r has been described in [9].

Side-Channel Attacks A few attacks exploiting side-channel information potentially leaked by implementations of IDEA have been published so far. A practical timing attack against key-dependent implementations of the IDEA multiplication has been described by Kelsey et al. [23]. Lemke et al. [28] have discussed the application of DPA-oriented techniques to attack implementations of IDEA on an 8-bit microcontroller, while Clavier et al. [14] have considered fault attacks. Protection methods have also been studied in [38].

Simplified IDEA Variants Some authors have also attacked simplified versions of IDEA. For instance, Borisov et al. [11] have replaced all the $\boxplus$ operations by $\oplus$ ones, except for the two in the output transformation. The authors showed that for one key out of 2^{16}, there exists a multiplicative differential characteristic over eight rounds that holds with probability 2^{-32}. Raddum [40] considered at the light of differential cryptanalysis another version, called IDEA-X/2, where only half of the $\boxplus$'s in one round are changed to $\oplus$ operations, namely the $\boxplus$'s where $Z_2^{(r)}$ and $Z_3^{(r)}$ are inserted, while the MA-structure is left unchanged.

3 A Wordslice IDEA Implementation

Given the fact that IDEA does not contain S-boxes and it uses only 16-bit arithmetical operations, it is particularly suited to be optimized on those processor architectures that include SIMD multimedia extensions; nowadays practically every PC is built around the `x86-64` architecture, which supports these features via the SSEx instruction sets. Moreover, this trend is also significantly showing up in the context of embedded systems, see for instance the ARC VRaptor$^{\mathrm{TM}}$ multicore architecture, the ARM NEON$^{\mathrm{TM}}$ technology and the new Intel Atom$^{\mathrm{TM}}$ and VIA Nano$^{\mathrm{TM}}$ processors.

Previous work [29] by Lipmaa has shown that a 4.35× increase in speed is achievable on Pentium processors that support the MMX instruction set. We now push the approach further, showing that IDEA can be very conveniently implemented on all those processors that implement the SSE2 instruction set, leading to encryption speed records; we also show that the multiplication modulo $2^{16} + 1$ can easily be implemented in a time-constant way, thanks to the SSE2 packed-word comparison instructions. We think that our results wipe away two common misconceptions about the IDEA block cipher, once and for all: that it is slow and difficult to secure against timing attacks [23]. On the contrary, we show that IDEA is probably one of the fastest block cipher on current microprocessor architectures, and it is completely immune to both timing attacks and cache attacks [1, 39].

The SSE2 instruction set defines 144 instructions that operate on 128-bit words; the SSE2 integer arithmetic instructions operate on *packed* data, i.e. each 128-bit operand is seen as a vector of 8, 16, 32 or 64-bit words. Since IDEA is natively operating on 16-bit variables, it is clearly possible to write SSE2 code that carries out eight IDEA encryptions in parallel; we call this implementation *wordslice*, as bitslice implementations [3, 30] would similarly work at bit level on 128 IDEA encryptions in parallel. The main advantage of wordslice implementations over bitslice is that to reach significant speedups it is not necessary to operate on huge amount of data, and that the orthogonalization process is straightforward.

Since the multiplication is clearly the most complex operation in the IDEA cipher, and the most critical regarding timing analysis, it deserves special care. The piece of code of Fig. 3, written using SSE2 intrinsics, is an implementation of the wordslice IDEA multiplication; it contains 11 pseudo-instructions, requires a space of four 128-bit registers and performs eight IDEA multiplications in parallel. It leverages on the unsigned multiplication instruction `_mm_mulhi_epu16` , whose functionality is not available in the MMX instruction set, and the comparison instruction `_mm_cmpeq_epi16`, which essentially allows it to be branch-free (and thus time-constant).

The two operands are initially contained in the a and b variables. The t and c variables are used as temporary storage and c contains the final result (the initial values of a and b are not preserved through the computation); `XMM_0` is the 128-bit zero string. The algorithm takes inspiration from known efficient implementations [4, 29], but eliminates any need of comparison or branch point. The main idea behind it is that the two values that would be returned whether $a \cdot b = 0$ or not are calculated in parallel; the final choice is determined by the value of a mask, the value of a at line 8, which is also derived from the input data. The algorithm also uses the fact that the upper and the lower 16 bits of $a \cdot b$ are equal if and only if at least one of the operands is 0; this property was never observed before and can be easily checked with an exhaustive simulation. For the sake of clarity, a line-equivalent (but obviously inefficient) C implementation that performs one IDEA multiplication using unsigned integer 32-bit variables is also given. The rest of the wordslice IDEA algorithm can be implemented with packed unsigned 16-bit additions and 128-bit XORs; this part is quite straightforward to derive and will not be shown here.

After having written a complete implementation based on the SSE2 instruction set, we have proceeded to the performance tests to assess the speed level of this code. It is surely interesting to test the encryption speed in ECB mode, with pre-calculated round keys, as this gives an indication of the raw speed that can be reached. However, by running the word-slice IDEA in counter mode one can also obtain a quite efficient stream cipher. We have thus implemented some simple routines to realize a branch-free SSE2 implementation of the counter mode of operation, and obtained a stream cipher with 48-bit IV and 128-bit key.

Both codes have been compiled with the Intel compiler; speed has been measured by executing the function a high number of times and taking the average time spent by the processor. We have also integrated our code in the eSTREAM [13] benchmarking framework and found that the figures differ by no more than 1% with regards to the ones we obtained. The result is that plain ECB encryption runs on an Intel Core2 processor at 5.55 cycles per byte, while counter mode keystream generation runs at 5.42 cycles[4] per byte.

[4] It is noteworthy that a code with more instructions takes less time to execute. This fact is most likely due to micro-architectural optimizations automatically performed by the compiler. In other words, putting less stress on the pipeline sometimes allows a better scheduling of the micro-operations.

```
t  =  _mm_add_epi16    (a, b);      /* t  =  (a + b) & 0xFFFF;      */
c  =  _mm_mullo_epi16  (a, b);      /* c  =  (a * b) & 0xFFFF;      */
a  =  _mm_mulhi_epu16  (a, b);      /* a  = (a * b) >> 16;          */
b  =  _mm_subs_epu16   (c, a);      /* b  =  (c - a);               */
                                    /* if (b & 0x80000000) b = 0;  */
b  =  _mm_cmpeq_epi16  (b, XMM_0);  /* if (b == 0) b = 0xFFFF;      */
                                    /* else b = 0;                  */
b  =  _mm_srli_epi16   (b, 15);     /* b  =  b >> 15;               */
c  =  _mm_sub_epi16    (c, a);      /* c  =  (c - a) & 0xFFFF;      */
a  =  _mm_cmpeq_epi16  (c, XMM_0);  /* if (c == 0) a = 0xFFFF;      */
                                    /* else a = 0;                  */
c  =  _mm_add_epi16    (c, b);      /* c  =  (c + b) & 0xFFFF;      */
t  =  _mm_and_si128    (t, a);      /* t  =  t & a;                 */
c  =  _mm_sub_epi16    (c, t);      /* c  =  (c - t) & 0xFFFF;      */
```

Fig. 3. Eight parallel IDEA multiplications using the SSE2 instruction set

We think that the reached level of security vs. speed trade-off justifies additional research effort; the IDEA cipher appears as an extremely good building block to realize other cryptographic primitives that may be used to implement authenticated encryption and hash functions. As a first step, we proceed in the next Section with the definition of the WIDEA block cipher family.

4 The WIDEA Block Cipher Family

In this section, we first describe the rationales behind the WIDEA cipher family design, then we make short, preliminary cryptanalysis of our proposal, followed by a discussion on implementation issues and performance results.

4.1 Design Rationale

We now show how a computational skeleton composed of n IDEA instances computed in parallel can be transformed in a natural and efficient way into a $(n \cdot 64)$-bit block cipher. Basically, we need to define a minimal modification that provides diffusion over the new block size; the term *minimal* bears several meanings here:

- The modification must require minimal computational overhead.
- The modification must alter the skeleton in the least noticeable way, in particular it must not affect the achieved degree of parallelism and it must be elegant.
- The modification must follow and enforce all original IDEA design criteria.

The first problem is to define the way in which we provide full diffusion within one round; MDS matrices over $\mathrm{GF}(2^n)$ are regarded as an optimal and efficient way to solve the problem [42], and have been extensively used in well-known

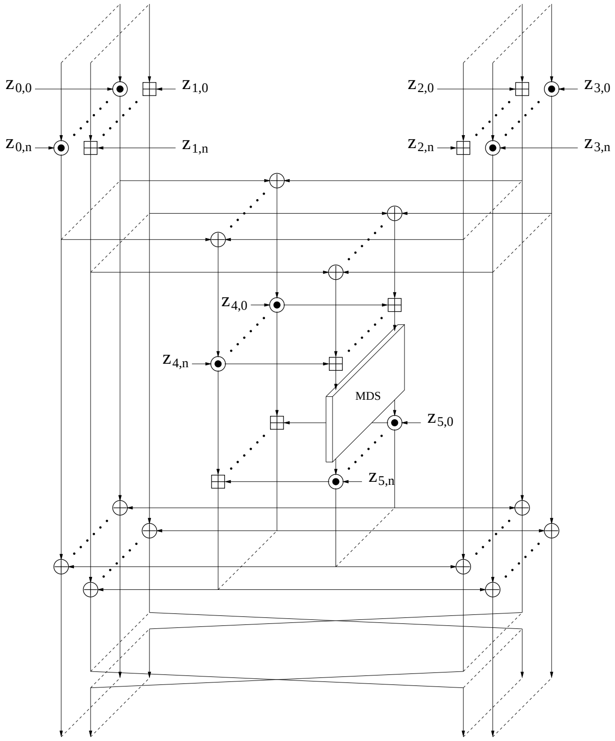

Fig. 4. The first round of WIDEA-n.

constructions [15, 16]: since the IDEA structure naturally operates on 16-bit words, we choose MDS matrices over $\mathrm{GF}(2^{16})$ as our diffusion primitive. A second problem is to identify the location in the IDEA round function to insert the diffusion block. The MA-box is an interesting place for two main reasons: it is already used to provide diffusion in the IDEA round function and it does not contain XOR operations. There are four arcs connecting the four operations inside the IDEA MA-box, but only a diffusion block inserted into the right arc would guarantee that full diffusion is still achieved. We therefore modify the n-way IDEA structure as shown in Fig. 4. The third dimension is used to represent the fact that n instances of IDEA can independently be computed and are tied together by the application of the MDS matrix.

The IDEA design criteria are effectively enforced; more specifically:

1. Full diffusion in the new block cipher is still obtained in one round, i.e. every round output bit depends on all $n \cdot 64$ input bits.
2. Every operation is still preceeded and followed by operations defined over algebraically incompatible groups.

```
b = _mm_and_si128 (a, XMM_0x8000);
a = _mm_slli_epi16 (a, 1);
b = _mm_cmpeq_epi16 (b, XMM_0x8000);
b = _mm_and_si128 (b, XMM_POLY);
a = _mm_xor_si128 (a, b);
```

Fig. 5. The wordslice Xtime operation

3. No dependence on arbitrary constants is introduced, the only choices being limited to the irreducible polynomial that defines the algebraic structure of GF(2^{16}) and the coefficients of the $n \times n$ MDS matrix that can be chosen in order to minimize the number of operations.

We call the new core of the round function *MAD-box* (standing for Multiply-Add-Diffuse) and we refer to the global $(n \cdot 64)$-bit construction as *WIDEA* (the name providing enough hints for a "wide" block IDEA cipher); the particular members of the family obtained by fixing the value of n are identified as WIDEA-n.

Compared with AES-like constructions operating on wide blocks (such as the W block cipher instanced in the Whirlpool hash function [20]), the WIDEA structure needs only one eighth of total MDS applications, thus keeping the computational cost of diffusion quite small. Variants with $n = 4$ (256-bit) and $n = 8$ (512-bit) block sizes are easily defined for instance by taking the same MDS matrices used in the AES and in the W ciphers, but defined over GF(2^{16}). These ciphers are very useful, as they can be used to build compression functions for 256-bit and 512-bit hash functions (this scenario also justifies the huge key sizes); as a reference, a complete specification of WIDEA-8 is given in the Appendix.

As we believe the new structure is not deviating substantially from IDEA, we think that it is not possible to exploit the bigger block size to mount attacks based on incomplete diffusion (such as integral attacks [24]) against more rounds than in IDEA, due to the fact that full diffusion is again obtained after one round. For this reason we keep the number of rounds of WIDEA at 8.5.

Instead, we focus our improvement effort on the key-schedule algorithm, which is significantly changed in order to remove the problem of weak keys and to render attacks that exploit the controllability of the key input more difficult. This is a valid scenario for related-key attacks and in case the block cipher is used to build a compression function (and a hash function). As in IDEA, we keep the key size equal to $(2n \cdot 64)$-bit (twice the block size), accepting the fact that not all key material can be used to key each round. However, to compensate for this we define a new key scheduling algorithm based on a non-linear feedback shift register, similarly to what is done in the MESH block ciphers [36]; we introduce non-linearity, diffusion and diversification in the WIDEA key scheduling, but always in a way to preserve the n-way parallelism already achieved in the cipher round function.

We denote with Z_i, $0 \leq i \leq 51$, the subkeys that are used in the 8.5 rounds of WIDEA-n; note that due to the n-way parallelism each subkey has a size of $n \cdot 16$ bits (thus each subkey Z_i can be split into the n slices $z_{i,0} \ldots z_{i,n}$). Moreover, denoting with K_i, $0 \leq i \leq 7$, the 8 words that represent the WIDEA master key, the new key scheduling algorithm is defined by the following equations:

$$\begin{aligned}
&Z_i = K_i && 0 \leq i \leq 7 \\
&Z_i = ((((Z_{i-1} \oplus Z_{i-8}) \overset{16}{\boxplus} Z_{i-5}) \overset{16}{\lll} 5) \lll 24) \oplus C_{\frac{i}{8}-1} && 8 \leq i \leq 51,\ 8 \mid i \\
&Z_i = ((((Z_{i-1} \oplus Z_{i-8}) \overset{16}{\boxplus} Z_{i-5}) \overset{16}{\lll} 5) \lll 24) && 8 \leq i \leq 51,\ 8 \nmid i
\end{aligned}$$

The symbols and the notation are explained more formally in the Appendix. Rotation by 5 bit positions is independently carried out on each 16-bit slice of Z_i, as suggested by the superscript; rotation by 24 bit positions (3 byte positions) is instead carried out globally on each $n \cdot 16$ bits word. The values of the constants $C_0 \div C_5$, that are injected every 8 iterations, should vary with the particular instance of the cipher.

The key scheduling is designed in a way such that it can be computed using a shift register of eight $n \cdot 16$-bit words and the same 16-bit arithmetical and logical operations used in the round function; the two rotation operations have been chosen as a practical and simple way to mix information between the n slices of the key schedule algorithm. Note that the byte level rotation is completely transparent in the context of 8-bit implementations. One could also think of fixing $n = 1$, basically obtaining IDEA with a strengthened key schedule, or $n = 2$ obtaining a cipher operating on the same block and key sizes as AES.

4.2 Preliminary Security Analysis

The fact that WIDEA is heavily based on the IDEA construction implies that all related cryptanalytic results may apply in a quite natural way, and therefore should be taken into consideration. We start this brief discussion by pointing out that considerable effort has been spent to strengthen the key-schedule part, as indicated above. Non-linearity is added by mixing XORs with integer addition, and different constants are injected every 8 iterations; thanks to this, it is very difficult to exploit repetitive patterns in the subkeys, or to find long sequences of subkeys characterized by low Hamming weight. We have verified with software simulations that thanks to the diffusion provided by the bit-level and byte-level rotations, coupled with integer additions, every bit of key material used starting from round 4 (non-linearly) depends on all the 1024 bits of the master key. For these reasons we expect that no weak keys can be found for WIDEA, and that the related-key attacks against IDEA cannot be transposed to WIDEA.

Regarding the classical attack scenarios, one may question if attacks can be based on the fact that the round function of WIDEA-n is based on n parallel instances of IDEA. Actually, the effect of the MDS diffusion matrix is to keep at 8 the number of full diffusions applied in the encryption process; to make a comparison, the AES block cipher applies a total of 5 to 7 full diffusions, depending on the key size. Recent proposals of big and efficient block ciphers, such as Threefish [19], are also quite conservatively designed to implement 7 or 8 full diffusions, depending on the digest size. Due to this property we do not expect that integral attacks, differential or linear attacks constitute a bigger threat for WIDEA than for IDEA. Obviously, independent cryptanalysis is needed to verify our claims, and we encourage further research in this direction.

4.3 WIDEA-8 Implementation Results

WIDEA-8 is certainly the most interesting member of the cipher family, since it can efficiently be computed using the XMM registers available in the `x86-64` architecture[5]. The coding of WIDEA naturally takes advantage of the optimizations discussed in §3; the same code is used to perform the wordslice IDEA multiplications. Concerning the MDS matrix multiplication, we show how to perform a wordslice GF(2^{16}) multiplication times 2 (this is equivalent to the AES `xtime` operation [37]). We use the `_mm_cmpeq_epi16` instruction to generate a polynomial mask to be XORed to the left-shifted input basing on the value of the MSB of each 16-bit slice of the input. The code using SSE intrinsics is given in Fig. 5; it contains 5 pseudo instructions.

The initial and final values are stored in a, while b is used as temporary storage; `XMM_0x8000` is a vector of eight 16-bit words with only the MSB set to 1 and `XMM_POLY` contains eight instances of the polynomial reduction mask. This `xtime` operation is used to compute the MDS matrix multiplication; the total number of `xtime` operations is determined by the maximum degree of the elements in the MDS matrix (this is equal to three for WIDEA-8). We also exploit the fact that the MDS matrix is circulant to optimize the number of computational steps; since this technique is highly dependent on the entries of the MDS matrix, it will not be discussed here.

Regarding the key-schedule algorithm, the operations are quite elementary and do not deserve special mention. The Intel SSE2 instruction set can be used to implement it easily using a bank of 9 XMM registers (8 to store the state of the non-linear feedback shift register plus one for temporary storage); if the SSE3 instruction set is supported by the target processor, which is the case of all Core2 CPUs, the `_mm_shuffle_epi8` byte shuffling instruction can be used in place of shift instructions to implement the byte-level rotation.

The WIDEA-8 cipher has been implemented by hand in Intel assembly language; this is done to exploit as much as possible the bank of XMM registers, as their number is increased from 8 to 16 when the code is executed in 64-bit mode. In this case it is possible to compute the key scheduling algorithm on-the-fly during encryption. Quite amazingly, in only one point in the code the space provided by the XMM register bank is not sufficient, and we need to save a variable in the cache memory; thus our optimized WIDEA code is almost completely acting solely on the processor register bank, and in a completely time-constant way.

[5] We note that in the future it may be possible to implement efficiently a WIDEA-16 instance, as Intel is planning to introduce in future micro-architectures the YMM registers, characterized by a size of 256 bits (however, only floating point instructions are planned so far for such operand size)

Hash Function	Speed (cycles per byte)
EDON-R 512	2.29
WIDEA-8	***5.98***
CubeHash8/32	6.03
Skein-512	6.10
Shabal-512	8.03
LUX	9.50
Keccak	10.00
BLAKE-64	10.00
Cheetah	13.60
Aurora	26.90
Grostl	30.45
ECHO-SP	35.70
SHAvite-3	38.20
Lesamnta	51.20
MD6	52.64
ECHO	53.50
Vortex	56.05
FUGUE	75.50

Fig. 6. Speed comparison of WIDEA used as a Davies-Meyer construction in the Merkle-Damgard mode and some SHA-3 candidates on the Intel Core2 CPU in 64-bit mode.

Having on-the-fly key-scheduling is important because we want to test the speed of WIDEA-8 in the cases when it is used to build a compression function using the Davies-Meyer construction. From our experiments, we have determined that such compression function is able to process data at the speed of 5.98 cycles per byte on a Intel Core2 processor. We anticipate that such cryptographic primitive can be used to define hash functions characterized by an outstanding security vs. speed trade-off; we do not offer the definition of a full hash function here, but we consider this as a very promising future work which will also benefit from the insights about hash modes of operation obtained during the SHA-3 competition. Anyway, speed comparison between some SHA3 candidates and our compression function used in a straightforward Merkle-Damgard mode is provided[6] in Table 6. The expiration of the patent protecting IDEA in a near future, and the fact that no intellectual property was applied for WIDEA, might also increase the interest in our work [7].

[6] All the data are the ones provided by their respective designers in the presentation slides of the First SHA-3 Candidate Conference for the Intel Core2 `x86-64` architecture.

[7] As a matter of fact, several SHA-3 submissions are re-using AES components; for instance, Vortex [25], is only about three times faster than the basic compression function we have devised here when implemented using the future Intel AES dedicated instructions.

Acknowledgments

We would like to thank Olivier Brique, Jérôme Perrine and Corinne Le Buhan Jordan for their kind support during this work.

References

1. D. Bernstein. Cache-timing attacks on AES. Available on `http://cr.yp.to/papers.html`, 2005.
2. D. Bernstein and P. Schwabe. New AES software speed records. To appear in the proceedings of Indocrypt'08. Available via `http://cr.yp.to/papers.html`, 2008.
3. E. Biham. A fast new DES implementation in software. In E. Biham, editor, *Fast Software Encryption: 4th International Workshop, FSE'97, Haifa, Israel, January 1997. Proceedings*, volume 1267 of *Lecture Notes in Computer Science*, pages 260–272. Springer-Verlag, 1997.
4. E. Biham. Optimization of IDEA. Technical report, `nes/doc/tec/wp6/026/1`, NESSIE Project, 2002. Available on `https://www.cryptonessie.org`.
5. E. Biham, A. Biryukov, and A. Shamir. Miss-in-the-middle attacks on IDEA and Khufru. In L. Knudsen, editor, *Fast Software Encryption: 6th International Workshop, FSE'99, Rome, Italy, March 1999. Proceedings*, volume 1636 of *Lecture Notes in Computer Science*, pages 124–138. Springer-Verlag, 1999.
6. E. Biham, O. Dunkelman, and N. Keller. Related-key boomerang and rectangle attacks. In 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*, pages 507–525. Springer-Verlag, 2005.
7. E. Biham, O. Dunkelman, and N. Keller. New cryptanalytic results on IDEA. In X. Lai and K. Chen, editors, *Advances in Cryptology – ASIACRYPT 2006: 12th International Conference on the Theory and Application of Cryptology and Information Security, Shangai China, December 3-7, 2006. Proceedings*, volume 4284 of *Lecture Notes in Computer Science*, pages 412–427. Springer-Verlag, 2006.
8. E. Biham, O. Dunkelman, and N. Keller. A new attack on 6-round IDEA. In A. Biryukov, editor, *Fast Software Encryption, 14th International Workshop, FSE 2007, Luxembourg, Luxembourg, March 26-28, 2007. Revised Selected Papers*, volume 4593 of *Lecture Notes in Computer Science*, pages 211–224. Springer-Verlag, 2007.
9. E. Biham, O. Dunkelman, and N. Keller. A unified approach to related-key attacks. In K. Nyberg, editor, *Fast Software Encryption, 15th International Workshop, FSE 2008, Lausanne, Switzerland, February 10-13, 2008. Revised Selected Papers*, volume 5086 of *Lecture Notes in Computer Science*, pages 73–96. Springer-Verlag, 2008.
10. E. Biham and A. Shamir. Differential cryptanalysis of DES-like cryptosystems. *Journal of Cryptology*, 4(1):3–72, 1991.
11. N. Borisov, M. Chew, R. Johnson, and D. Wagner. Multiplicative differentials. In J. Daemen and V. Rijmen, editors, *Fast Software Encryption: 9th International Workshop, FSE 2002, Leuven, Belgium, February 4-6, 2002. Revised Papers*, volume 2365 of *Lecture Notes in Computer Science*, pages 17–33. Springer-Verlag, 2002.

12. J. Borst, L. Knudsen, and V. Rijmen. Two attacks on reduced IDEA (extended abstract). In W. Fumy, editor, *Advances in Cryptology* – Eurocrypt *'97: International Conference on the Theory and Application of Cryptographic Techniques, Konstanz, Germany, May 1997. Proceedings*, volume 1233 of *Lecture Notes in Computer Science*, pages 1–13. Springer-Verlag, 1997.
13. C. De Cannière. eSTREAM testing framework. Available at `http://www.ecrypt.eu.org/stream/perf/`.
14. C. Clavier, B. Gierlichs, and I. Verbauwhede. Fault analysis study of IDEA. In T. Malkin, editor, *Topics in Cryptology - CT-RSA 2008, The Cryptographers' Track at the RSA Conference 2008, San Francisco, CA, USA, April 8-11, 2008. Proceedings*, volume 4964 of *Lecture Notes in Computer Science*, pages 274–287. Springer-Verlag, 2008.
15. J. Daemen and V. Rijmen. *The Design of Rijndael*. Information Security and Cryptography. Springer, 2002.
16. J. Damen, L. Knudsen, and V. Rijmen. The block cipher SQUARE. In E. Biham, editor, *Fast Software Encryption: 4th International Workshop, FSE'97, Haifa, Israel, January 1997. Proceedings*, volume 1267 of *Lecture Notes in Computer Science*, pages 149–165. Springer-Verlag, 1997.
17. H. Demirci. Square-like attacks on reduced rounds of IDEA. In K. Nyberg and H. Heys, editors, *Selected Areas in Cryptography: 9th Annual International Workshop, SAC 2002, St. John's, Newfoundland, Canada, August 15-16, 2002. Revised Papers*, volume 2595 of *Lecture Notes in Computer Science*, pages 147–159. Springer-Verlag, 2003.
18. H. Demirci, A. Selçuk, and E. Türe. A new meet-in-the-middle attack on the IDEA block cipher. In *Selected Areas in Cryptography: 10th Annual International Workshop, SAC 2003, Ottawa, Canada, August 2003. Revised Papers*, volume 3006 of *Lecture Notes in Computer Science*, pages 117–129. Springer-Verlag, 2004.
19. N. Ferguson, S. Lucks, B. Schneier, D. Whiting, M. Bellare, T. Kohno, J. Callas, and J. Walker. The Skein hash function family – version 1.1. NIST SHA-3 Submission. Available via `http://ehash.iaik.tugraz.at/wiki/The_eHash_Main_Page`, 2008.
20. ISO. *Information technology – Security techniques – Hash-functions – Part 3: Dedicated hash-functions*. ISO/IEC 10118-3:2004, International Organization for Standardization, Genï£¡ve, Switzerland, 2004.
21. P. Junod. New attacks against reduced-round versions of IDEA. In H. Gilbert and H. Handschuh, editors, *Fast Software Encryption, 12th International Workshop, FSE 2005, Paris, France, February 21-23, 2005. Revised Selected Papers*, volume 3557 of *Lecture Notes in Computer Science*, pages 384–397. Springer-Verlag, 2005.
22. E. Käsper and P. Schwabe. Faster and timing-attack resistant AES-GCM. IACR ePrint Archive Report 2009/129. Available on `http://eprint.iacr.org/2009/129`.
23. J. Kelsey, B. Schneier, D. Wagner, and C. Hall. Side channel cryptanalysis of product ciphers. *Journal of Computer Security*, 8(2/3), 2000.
24. L. Knudsen and D. Wagner. Integral cryptanalysis (extended abstract). In J. Daemen and V. Rijmen, editors, *Fast Software Encryption: 9th International Workshop, FSE 2002, Leuven, Belgium, February 4-6, 2002. Revised Papers*, volume 2365 of *Lecture Notes in Computer Science*, pages 112–127. Springer-Verlag, 2002.
25. M. Kounavis and S. Gueron. Vortex: A new family of one way hash functions based on Rijndael rounds and carry-less multiplication. NIST SHA-3 Submission. Available via `http://ehash.iaik.tugraz.at/wiki/The_eHash_Main_Page`, 2008.

26. X. Lai. *On the design and security of block ciphers*, volume 1 of *ETH Series in Information Processing*. Hartung-Gorre Verlag, 1992.
27. X. Lai and J. Massey. A proposal for a new block encryption standard. In I. Damgård, editor, *Advances in Cryptology – Eurocrypt'90: Workshop on the Theory and Application of Cryptographic Techniques, Aarhus, Denmark, May 1990. Proceedings*, volume 473 of *Lecture Notes in Computer Science*, pages 389–404. Springer-Verlag, 1991.
28. K. Lemke, K. Schramm, and C. Paar. DPA on n-bit sized Boolean and arithmetic operations and its application to IDEA, RC6, and the HMAC-construction. In M. Joye and J.-J. Quisquater, editors, *Cryptographic Hardware and Embedded Systems – CHES 2004: 6th International Workshop, Cambridge, MA, USA, August 11-13, 2004. Proceedings*, volume 3156 of *Lecture Notes in Computer Science*, pages 205–219. Springer-Verlag, 2004.
29. H. Lipmaa. IDEA: a cipher for multimedia architectures? In S. Tavares and H. Meijer, editors, *Selected Areas in Cryptography: 5th Annual International Workshop, SAC'98, Kingston, Ontario, Canada, August 1998. Proceedings*, volume 1556 of *Lecture Notes in Computer Science*, pages 248–263. Springer-Verlag, 1999.
30. M. Matsui and J. Nakajima. On the power of bitslice implementation on Intel Core2 processor. In P. Paillier and I. Verbauwhede, editors, *Cryptographic Hardware and Embedded Systems – CHES 2007: 9th International Workshop, Vienna, Austria, September 10-13, 2007. Proceedings*, volume 4727 of *Lecture Notes in Computer Science*, pages 121–134. Springer-Verlag, 2007.
31. W. Meier. On the security of the IDEA block cipher. In T. Helleseth, editor, *Advances in Cryptology – Eurocrypt'93: Workshop on the Theory and Application of Cryptographic Techniques, Lofthus, Norway, May 1993. Proceedings*, volume 765 of *Lecture Notes in Computer Science*, pages 371–385. Springer-Verlag, 1993.
32. J. Nakahara. Faster variants of the MESH block ciphers. In A. Canteaut and K. Viswanathan, editors, *Progress in Cryptology - INDOCRYPT 2004: 5th International Conference on Cryptology in India, Chennai, India, December 20-22, 2004. Proceedings*, volume 3348 of *Lecture Notes in Computer Science*, pages 162–174. Springer-Verlag, 2004.
33. J. Nakahara, P. Barreto, B. Preneel, J. Vandewalle, and Y. Kim. Square attacks on reduced-round PES and IDEA block ciphers. In B. Macq and J.-J. Quisquater, editors, *Proceedings of 23rd Symposium on Information Theory in the Benelux, Louvain-la-Neuve, Belgium, May 29-31, 2002*, pages 187–195, 2002.
34. J. Nakahara, B. Preneel, and J. Vandewalle. The Biryukov-Demirci attack on IDEA and MESH ciphers. Technical report, COSIC, ESAT, Katholieke Universiteit Leuven, Leuven, Belgium, 2003.
35. J. Nakahara, B. Preneel, and J. Vandewalle. The Biryukov-Demirci attack on reduced-round versions of IDEA and MESH block ciphers. In H. Wang, J. Pieprzyk, and V. Varadharajan, editors, *Information Security and Privacy: 9th Australasian Conference, ACISP 2004, Sydney, Australia, July 13-15, 2004. Proceedings*, volume 3108 of *Lecture Notes in Computer Science*, pages 98–109. Springer-Verlag, 2004.
36. J. Nakahara, V. Rijmen, B. Preneel, and J. Vandewalle. The MESH block ciphers. In K. Chae and M. Yung, editors, *Information Security Applications, 4th International Workshop, WISA 2003, Jeju Island, Korea, August 25-27, 2003. Revised Papers*, volume 2908 of *Lecture Notes in Computer Science*, pages 458–473. Springer-Verlag, 2004.
37. National Institute of Standards and Technology, U. S. Department of Commerce. *Advanced Encryption Standard (AES)*, NIST FIPS PUB 197, 2001.

38. O. Neisse and J. Pulkus. Switching blindings with a view towards IDEA. In M. Joye and J.-J. Quisquater, editors, *Cryptographic Hardware and Embedded Systems – CHES 2004: 6th International Workshop, Cambridge, MA, USA, August 11-13, 2004. Proceedings*, volume 3156 of *Lecture Notes in Computer Science*, pages 230–239. Springer-Verlag, 2004.
39. D. Osvik, A. Shamir, and E. Tromer. Cache attacks and countermeasures: the case of AES. In D. Pointcheval, editor, *Topics in Cryptology - CT-RSA 2006, The Cryptographers' Track at the RSA Conference 2006, San Jose, CA, USA, February 13-17, 2006. Proceedings*, volume 3860 of *Lecture Notes in Computer Science*, pages 1–20. Springer-Verlag, 2006.
40. H. Raddum. Cryptanalysis of IDEA-X/2. In T. Johansson, editor, *Fast Software Encryption, 10th International Workshop, FSE 2003, Lund, Sweden, February 24-26, 2003. Revised Papers*, volume 2887 of *Lecture Notes in Computer Science*. Springer-Verlag, 2003.
41. R. Rivest, B. Agre, D. Bailey, C. Crutchfield, Y. Dodis, K. Fleming, A. Khan, J. Krishnamurthy, Y. Lin, L. Reyzin, E. Shen, J. Sukha, D. Sutherland, E. Tromer, and Y. Yin. The MD6 hash function – a proposal to NIST for SHA-3. NIST SHA-3 Submission. Available via `http://ehash.iaik.tugraz.at/wiki/The_eHash_Main_Page`, 2008.
42. C. Schnorr and S. Vaudenay. Black box cryptanalysis of hash networks based on multipermutations. In A. De Santis, editor, *Advances in Cryptology – EUROCRYPT '94: Workshop on the Theory and Application of Cryptographic Techniques, Perugia, Italy, May 1994. Proceedings*, volume 950 of *Lecture Notes in Computer Science*, pages 47–57. Springer-Verlag, 1995.

Appendix: Specification of WIDEA-8

Notation WIDEA-8 is a block cipher having block size of 512 bits and fixed key size of 1024 bits, which is heavily based on the IDEA cipher. In the following we will indicate 128-bit words with capital letters and 16-bit words with lowercase letters. The input, state and output of the cipher can be seen as an array of four 128-bit words, where each 128-bit word can in turn be seen as an array of eight 16-bit words. Thus, each 16-bit word is indexed by two numbers: the index of the 128-bit word that contains it, followed by the index of its position in the 128-bit word (128-bit words are indexed by only one number). We adopt here a big-endian ordering, so that the index of the most significant part of a variable is equal to 0 and the index of its least significant part is the largest one. Thus, indicating with $\mathbb{X}$ the 512-bit input of the cipher, we have $\mathbb{X} = X_0 \| X_1 \| X_2 \| X_3$ and $X_0 = x_{0,0} \| x_{0,1} \| x_{0,2} \| x_{0,3} \| x_{0,4} \| x_{0,5} \| x_{0,6} \| x_{0,7}$. Different arithmetic and logic operations are used in WIDEA-8. The IDEA multiplication of two 16-bit words (multiplication over $\mathbb{Z}^*_{2^{16}+1}$ where the zero 16-bit string represents the number 2^{16}) is denoted with "$\odot$"; addition over $\mathbb{Z}_{2^{16}}$ is denoted with "$\boxplus$". Each 16-bit word can also be seen as an element of the finite field GF(2^{16}) defined with the following irreducible polynomial $P(x) = x^{16} + x^5 + x^3 + x^2 + 1$. Addition over GF($2^{16}$), as well as bitwise logical XOR, is denoted with "$\oplus$" while multiplication over the same field is denoted with "$\cdot$"; logical left rotation of n positions is denoted with "$\lll n$". The same operators may be applied in a vectorial way over the 128-bit variables, i.e. each 16-bit slice of the operand(s) undergoes the transformations above; in this case we place the superscript "16" over the operator symbol, to distinguish the operation from the one carried out over the full 128 bits.

The Key-Schedule The WIDEA-8 key $\mathbb{Z}$ has size equal to 1024 bits and can be seen as an array of eight 128-bit words $\mathbb{Z} = Z_0 \| Z_1 \| Z_2 \| Z_3 \| Z_4 \| Z_5 \| Z_6 \| Z_7$. This array is filled with key material starting from the most significant positions and proceeding toward the least significant ones. Once the key has been set, the subkeys can be generated. WIDEA-8, similarly to IDEA, uses a total of 52 128-bit subkeys; the first 8 are taken directly from the key, starting naturally from Z_0. The additional 44 128-bit subkeys Z_i with $8 \leq i \leq 51$ are generated using the following equations:

$$Z_i = ((((Z_{i-1} \oplus Z_{i-8}) \overset{16}{\boxplus} Z_{i-5}) \overset{16}{\lll} 5) \lll 24) \oplus C_{\frac{i}{8}-1} \qquad 8 \leq i \leq 51,\ 8 \mid i$$
$$Z_i = ((((Z_{i-1} \oplus Z_{i-8}) \overset{16}{\boxplus} Z_{i-5}) \overset{16}{\lll} 5) \lll 24) \qquad 8 \leq i \leq 51,\ 8 \nmid i$$

In practice, the same recurrence relation is used for each iteration, and a different constant is added whenever the subkey index is a multiple of 8. The six 128-bit constants are given below, in hexadecimal format:

$$C_0 = \texttt{1dea}\|\texttt{0000}\|\texttt{0000}\|\texttt{0000}\|\texttt{0000}\|\texttt{0000}\|\texttt{0000}\|\texttt{0000}$$
$$C_1 = \texttt{3825}\|\texttt{0000}\|\texttt{0000}\|\texttt{0000}\|\texttt{0000}\|\texttt{0000}\|\texttt{0000}\|\texttt{0000}$$
$$C_2 = \texttt{1dd7}\|\texttt{0000}\|\texttt{0000}\|\texttt{0000}\|\texttt{0000}\|\texttt{0000}\|\texttt{0000}\|\texttt{0000}$$
$$C_3 = \texttt{3ea4}\|\texttt{0000}\|\texttt{0000}\|\texttt{0000}\|\texttt{0000}\|\texttt{0000}\|\texttt{0000}\|\texttt{0000}$$
$$C_4 = \texttt{e57a}\|\texttt{0000}\|\texttt{0000}\|\texttt{0000}\|\texttt{0000}\|\texttt{0000}\|\texttt{0000}\|\texttt{0000}$$
$$C_5 = \texttt{f7ba}\|\texttt{0000}\|\texttt{0000}\|\texttt{0000}\|\texttt{0000}\|\texttt{0000}\|\texttt{0000}\|\texttt{0000}$$

Encryption WIDEA-8 encryption consists in 8 full rounds followed by one half round; every round uses some subkeys calculated using the key scheduling algorithm specified above. We add an apex to each variable to indicate the round number, starting from 1; thus $\delta_{0,0}^{(i)}$ is the value of $\delta_{0,0}$ in the i-th round. The input of the i-th round is denoted as $\mathbb{X}^{(i)} = X_0^{(i)} \| X_1^{(i)} \| X_2^{(i)} \| X_3^{(i)}$ and the output $\mathbb{Y}^{(i)} = \mathbb{X}^{(i+1)}$ is calculated, for a full round, as follows. First, the inputs of the MAD-box are calculated as:

$$A^{(i)} = \left(X_0^{(i)} \overset{16}{\odot} Z_{6(i-1)}\right) \oplus \left(X_2^{(i)} \overset{16}{\boxplus} Z_{6(i-1)+2}\right)$$
$$B^{(i)} = \left(X_1^{(i)} \overset{16}{\boxplus} Z_{6(i-1)+1}\right) \oplus \left(X_3^{(i)} \overset{16}{\odot} Z_{6(i-1)+3}\right)$$

Then, the MAD-box calculation is carried out, resulting in:

$$\Delta^{(i)} = \mathrm{MDS}\left(\left(A^{(i)} \overset{16}{\odot} Z_{6(i-1)+4}\right) \overset{16}{\boxplus} B^{(i)}\right) \overset{16}{\odot} Z_{6(i-1)+5}$$
$$\Gamma^{(i)} = \left(A^{(i)} \overset{16}{\odot} Z_{6(i-1)+4}\right) \overset{16}{\boxplus} \Delta^{(i)}$$

The MDS operation is a left-multiplication over GF(2^{16}) of a 128-bit string with a fixed matrix, its elements defined over GF(2^{16}), and is defined as follows:

$$Y = \begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \\ y_7 \end{pmatrix} = \begin{pmatrix} 1\,1\,4\,1\,8\,5\,2\,9 \\ 9\,1\,1\,4\,1\,8\,5\,2 \\ 2\,9\,1\,1\,4\,1\,8\,5 \\ 5\,2\,9\,1\,1\,4\,1\,8 \\ 8\,5\,2\,9\,1\,1\,4\,1 \\ 1\,8\,5\,2\,9\,1\,1\,4 \\ 4\,1\,8\,5\,2\,9\,1\,1 \\ 1\,4\,1\,8\,5\,2\,9\,1 \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{pmatrix} = \mathrm{MDS}(X)$$

Thus, we could equivalently write a set of equations looking as this:

$$y_0 = x_0 \oplus x_1 \oplus 4 \cdot x_2 \oplus x_3 \oplus 8 \cdot x_4 \oplus 5 \cdot x_5 \oplus 2 \cdot x_6 \oplus 9 \cdot x_7$$

The output of the round is finally obtained combining the outputs of the MAD-box Δ and Γ with A and B as follows:

$$\begin{aligned} Y_0^{(i)} &= \left(X_0^{(i)} \overset{16}{\odot} Z_{6(i-1)}\right) \oplus \Delta^{(i)} & \qquad Y_1^{(i)} &= \left(X_2^{(i)} \overset{16}{\boxplus} Z_{6(i-1)+2}\right) \oplus \Delta^{(i)} \\ Y_2^{(i)} &= \left(X_1^{(i)} \overset{16}{\boxplus} Z_{6(i-1)+1}\right) \oplus \Gamma^{(i)} & \qquad Y_3^{(i)} &= \left(X_3^{(i)} \overset{16}{\odot} Z_{6(i-1)+3}\right) \oplus \Gamma^{(i)} \end{aligned}$$

On the other hand, a half round contains less operations and is defined as follows:

$$\begin{aligned} Y_0^{(i)} &= X_0^{(i)} \overset{16}{\odot} Z_{6(i-1)} & \qquad Y_1^{(i)} &= X_2^{(i)} \overset{16}{\boxplus} Z_{6(i-1)+1} \\ Y_2^{(i)} &= X_1^{(i)} \overset{16}{\boxplus} Z_{6(i-1)+2} & \qquad Y_3^{(i)} &= X_3^{(i)} \overset{16}{\odot} Z_{6(i-1)+3} \end{aligned}$$

Decryption WIDEA-8 decryption also consists in 8 full rounds followed by one half round; every round uses some subkeys calculated using the key scheduling algorithm specified above. The definitions of the full and half rounds for decryption is the same as that given above for encryption; the only difference is that the subkeys (previously inverted with respect to the proper law group) must be used in the inverse order. Note that the WIDEA key schedule algorithm is designed to be easily invertible, thus one may also apply on-the-fly inverse key scheduling for decryption, where the master key contains the last 8 subkeys derived for encryption.

Test vectors

```
PLAINTEXT

0000 0011 0022 0033 0044 0055 0066 0077
0088 0099 00aa 00bb 00cc 00dd 00ee 00ff
```

```
ff00 ee00 dd00 cc00 bb00 aa00 9900 8800
7700 6600 5500 4400 3300 2200 1100 0000

KEY

0000 0001 0002 0003 0004 0005 0006 0007
0008 0009 000a 000b 000c 000d 000e 000f
0000 0010 0020 0030 0040 0050 0060 0070
0080 0090 00a0 00b0 00c0 00d0 00e0 00f0
0000 0100 0200 0300 0400 0500 0600 0700
0800 0900 0a00 0b00 0c00 0d00 0e00 0f00
0000 1000 2000 3000 4000 5000 6000 7000
8000 9000 a000 b000 c000 d000 e000 f000

CIPHERTEXT

c28c 1bcf b923 65f9 d8a0 2d77 417c 3da8
f6ed 06ba 961e 3948 4162 ccaa a62a da5b
d6f2 b750 ecfb 22ce 71a3 3380 c8ef aa90
1424 67da 51fd 1d38 0978 cccc c99a 5f5a
```