Cryptography is Feasible on 4-Bit Microcontrollers - A Proof of Concept

Markus Vogt*, Axel Poschmann†, Christof Paar*
*Horst Görtz Institute for IT Security
Embedded Security Group
44801 Bochum, Germany
vogt@et.rub.de, cpaar@crypto.rub.de
† Division of Mathematical Sciences
School of Physical and Mathematical Sciences
Nanyang Technological University
Singapore 639798
aposchmann@ntu.edu.sg

Abstract

The RFID technology in combination with cryptographic algorithms and protocols is discussed widely as a promising solution against product counterfeiting. Usually the discussion is focussed on passive low-cost RFID-tags, which have harsh power constraints. 4*-Bit microcontrollers have very low-power characteristics (5-60 μA) and are therefore an interesting platform for active and passive low-cost RFID-tags. To the best of our knowledge there are no implementations of cryptographic algorithms on a* 4*-bit microcontroller published so far. Therefore, the main contribution of this work is to demonstrate that cryptography is feasible on these ultra-constrained devices and to close this gap. We chose* PRESENT *[1] as the cryptographic algorithm, because contrary to many other ciphers,* PRESENT *uses a 4x4 S-Box. Our implementation draws a current of* $6.7\mu A$ *at a supply voltage of* $1.8V$ *and a frequency of* 500 *KHz and requires less than* 200 *ms for the processing of one data block.*

I. Introduction

According to the U.S. Chamber of Commerce "counterfeiting and product piracy cost the U.S. economy between \$200 billion and \$250 billion per year and a total of 750.000 American Jobs" [2, p.26]. Combined with other sources, [3] estimates the global market size of counterfeited goods with US-\$527 billion[1]. Radio Frequency IDentification (RFID) is widely discussed as a promising solution for the counterfeiting issues in the literature [5], [6], [7], [8]. Many of the proposed authentication protocols use a Pseudo Random Number Generator (PRNG), a hash function, or symmetric key encryption [9], [10], [11], [12], [13], [14], [15], [16], [17]. Block ciphers can be used as a hash function in Davies-Meyer mode or as a PRNG in Output Feedback Mode and therefore serve as an integral part of many secure identification systems.

In the literature often the term RFID focuses on passive low-cost RFID-tags, but actually the general term RFID denotes a variety of radio-based identification technologies [18]. Besides hardware implementations also microcontrollers may be used. 4-Bit microcontroller are already deployed in a very broad range of everyday life items, ranging from watches, micro-ovens and washing machines to security critical applications such as car tire sensors or one-time PIN generators. To the best of our knowledge there are no implementations of cryptographic algorithms on a 4-bit microcontroller published so far. Therefore, the main contribution of this work is to demonstrate that cryptography is feasible on ultra-constrained 4-bit microcontrollers and to close this gap. We chose the ATAM893$-$D microcontroller of ATMEL's MARC4 family due to two reasons. Firstly, it has a low power consumption of below 1 mA [19], which makes it a very interesting candidate

Axel Poschmann performed most of the work described in this contribution as a researcher with the Embedded Security Group.

[1]Note that the value of global drug trade is estimated with US-\$321.6 billion in 2005 [4, p.127].

for active low-cost RFID-tags. Secondly, it is very hard to obtain 4-bit microcontrollers together with a suitable development kit. We contacted several different manufacturers of 4-bit microcontrollers, but only ATMEL responded and kindly provided us with a complete development kit.

A natural choice for the cryptographic algorithm would have been the Advanced Encryption Standard (AES [20]), because it is the most widespread and probably the best scrutinized symmetric encryption algorithm. The AES uses a substitution table with 8-bit input and 8-bit output (so-called 8x8 S-Boxes), which is perfect for 8-bit microcontrollers and well suited for CPUs with larger datapaths. However, since our goal is to demonstrate cryptography on 4-bit microcontrollers, we chose PRESENT as the cryptographic algorithm, because contrary to many other ciphers PRESENT uses a 4x4 S-Box (i.e. a substitution table with 4-bit input and 4-bit output). PRESENT is a recently published ultra-lightweight block cipher, which was designed especially with hardware implementations for ultra-constrained devices such as passive RFID-tags in mind [1]. Furthermore, its hardware implementation results of 1000 gate equivalents [21] are the smallest of any symmetric cipher published so far. It is worth noticing that also stream ciphers such as TRIVIUM [22] or GRAIN [23] that are designed with a low hardware footprint in mind require more hardware resources [24]. However, software implementation results for PRESENT are not published so far, except for a bit-sliced implementation in a publication of Grabher *et al.* [25] that is based on a student project at Ruhr University Bochum. Again we would like to point out that to the best of our knowledge there are no implementations of cryptographic algorithms on 4-bit microcontrollers published so far.

The remainder of this work is organized as follows: in Section II we first recall the PRESENT algorithm and briefly describe its building blocks. Then in Section III we describe the used development environment for programming the ATAM893−D. Subsequently, in Section IV we describe our PRESENT implementation in detail and present our results. Finally this article is concluded in Section V.

II. The PRESENT Algorithm

PRESENT is a recently published ultra-lightweight block cipher, which was specially designed with hardware implementations for ultra-constrained devices such as passive RFID-tags in mind [1]. It is a substitution-permutation network with 64-bits block size and 80 or 128-bits of key (from here on referred to as PRESENT-80 or PRESENT-128, respectively). Though PRESENT is a relatively new algorithm, already some cryptanalytic results have been published [26], [27], [28], [29]. However, none of the proposed attacks on reduced-round versions if of PRESENT is practically feasible or better than brute-force if extended to the full amount of 32 rounds. In fact these results rather underline the strength of the cipher PRESENT. In the remainder of this article we therefore focus on PRESENT-80 rather than AES-128, because 80-bits provide a security level which is sufficient for many RFID-ish applications. PRESENT has 31 regular rounds and a final round that only consists of the key mixing step. One regular round consists of a key mixing step, a substitution layer, and a permutation layer. Figure 1 depicts the top-level algorithmic description of PRESENT.

Contrary to other block ciphers such as the AES PRESENT uses a substitution table that only has 4-bit input and 4-bit output (so-called 4x4 S-Boxes). The state is split into 16 4-bit chunks that are then each substituted by a single 4x4 S-Box. The S-Box is given in hexadecimal notation according to Table I. It is worth to mention that the substitution layer consists of 16 *identical* 4x4 S-Boxes and also the key schedule uses the same 4x4 S-Box. This constitutes a great advantage compared to other block ciphers, because it significantly lowers area and memory requirements for hardware and software implementations. Actually, this was the main reason to chose PRESENT for implementation on a 4-bit microcontroller.

The bit permutation used in PRESENT is given by Table II. Bit i of STATE is moved to bit position P(i).

The key schedule of PRESENT-80 consists of a 61-bit left rotation, an S-Box, and an XOR with a round counter. Note that PRESENT uses the same S-Box for the data-path and the key schedule, which allows to share resources.

TABLE I. S-Box Layer of PRESENT

x	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
S[x]	C	5	6	B	9	0	A	D	3	E	F	8	4	7	2	1

TABLE II. Permutation layer of PRESENT

i	P(i)	i	P(i)	i	P(i)	i	P(i)
0	0	16	4	32	8	48	12
1	16	17	20	33	24	49	28
2	32	18	36	34	40	50	44
3	48	19	52	35	56	51	60
4	1	20	5	36	9	52	13
5	17	21	21	37	25	53	29
6	33	22	37	38	41	54	45
7	49	23	53	39	57	55	61
8	2	24	6	40	10	56	14
9	18	25	22	41	26	57	30
10	34	26	38	42	42	58	46
11	50	27	54	43	58	59	62
12	3	28	7	44	11	60	15
13	19	29	23	45	27	61	31
14	35	30	39	46	43	62	49
15	51	31	55	47	59	63	63

```
generateRoundKeys()
for $i = 1$ to 31 do
    addRoundKey(STATE,$K_i$)
    sBoxLayer(STATE)
    pLayer(STATE)
end for
addRoundKey(STATE,$K_{32}$)
```

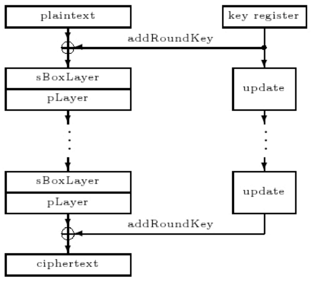
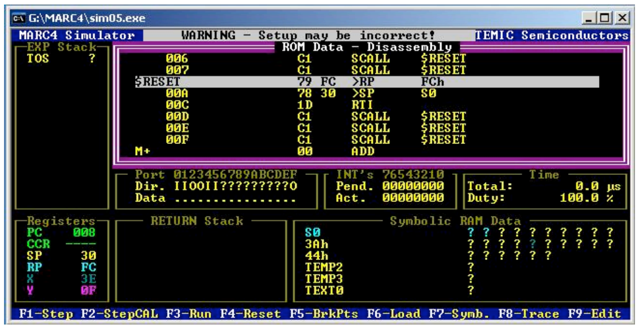


Fig. 1. A top-level algorithmic description of PRESENT**.**



```
G:\MARC4\sim05.exe
MARC4 Simulator      WARNING - Setup may be incorrect!      TEMIC Semiconductors
-EXP Stack-                        ROM Data - Disassembly
 TOS      ?        006                  C1     SCALL   $RESET
                   007                  C1     SCALL   $RESET
                 $RESET                 79 FC  >RP     FCh
                   00A                  78 30  >SP     S0
                   00C                  1D     RTI
                   00D                  C1     SCALL   $RESET
                   00E                  C1     SCALL   $RESET
                   00F                  C1     SCALL   $RESET
                 M+                     00     ADD
              Port 0123456789ABCDEF   INT's 76543210          Time
              Dir. IIOOII?????????O   Pend. 00000000   Total:      0.0 µs
              Data ................   Act.  00000000   Duty:     100.0 %
-Registers-     RETURN Stack                Symbolic RAM Data
 PC     008                     S0                ? ? ? ? ? ? ? ? ? ?
 CCR    ----                    3Ah               ? ? ? ? ? ? ? ? ? ?
 SP      30                     44h               ? ? ? ? ? ?
 RP      FC                     TEMP2             ?
 X       3E                     TEMP3             ?
 Y       0F                     TEXT0             ?
F1-Step F2-StepCAL F3-Run F4-Reset F5-BrkPts F6-Load F7-Symb. F8-Trace F9-Edit
```

Fig. 2. Screenshot of the MARC4 Simple Core Simulator

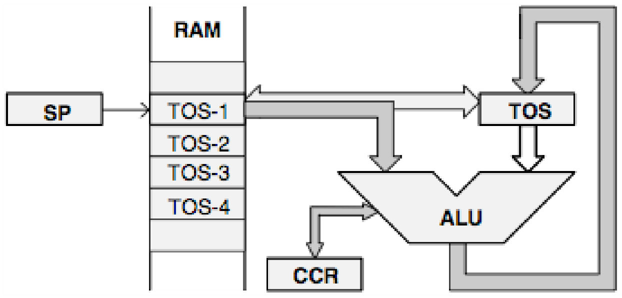


Fig. 3. Arithmetic Logic Unit - ALU of the MARC4, *source:[19]*

The user-supplied key is stored in a key register and its 64 most significant (i.e. leftmost) bits serve as the round key. The key register is rotated by 61 bit positions to the left, the left-most four bits are passed through the PRESENT S-Box, and the round counter value i is exclusive-ored with bits $\mathtt{k}_{19}\mathtt{k}_{18}\mathtt{k}_{17}\mathtt{k}_{16}\mathtt{k}_{15}$ of K with the least significant bit of the round counter on the right. For further details, the interested reader is referred to [1].

III. 4-Bit Target Platform

The ATAM893−D, member of Atmel's MARC4 family of 4-bit single-chip microcontrollers inherits a RISC[2] core and contains EEPROM, RAM, parallel input/output ports, two 8-bit programmable multifunction counters/timer and an on-chip clock generation with integrated RC-, 32-kHz and 4-MHz crystal oscillators. The unique combination of high data throughput and low current consumption makes the ATAM893−D a perfect candidate for wireless applications such as remote keyless entry, immobilizer systems, wireless sensors and other data control applications needing cryptographic security routines.

Atmel's MARC4 microcontroller family is based on a low-power 4-bit CPU core. The modular MARC4 architecture is high-level language oriented, consuming still below 1 mA in active mode [19]. Programming of MARC4 microcontrollers is supported by a personal computer based software development system with a high-level language *qForth* compiler and a real-time core simulator (see Fig. 2).

The CPU is based on the Harvard architecture with physically separate program memory and data memory. Three independent buses (instruction-, memory- and I/O-bus) are used for parallel communication between ROM, RAM and peripherals. This enhances the program execution speed by allowing both instruction prefetching and a simultaneous communication to the on-chip peripheral circuitry. The integrated interrupt controller, with eight prioritized interrupt levels, supports fast processing of hardware events. The core contains 4 KByte program memory (ROM), 256x4-bit data memory (RAM), arithmetic-logic-unit (ALU) (see Fig. 3), Program Counter (PC), RAM address register, instruction decoder and interrupt controller. The RAM is used for the Expression Stack, the Return Stack and as data memory for variables and arrays. It can be addressed by any of the four 8-bit wide RAM Address Registers `SP`, `RP`, `X` and `Y`. These registers allow access to any of the 256 RAM nibbles.

All arithmetic, I/O and memory reference operations take their operands *from*, and return their result *to* the Expression Stack (EXP) which is addressed by `SP`. The MARC4 performs the operations with the top of stack items (`TOS` and `TOS-1`). The `TOS` register contains the top element of EXP and works in the same way as an accumulator. This stack is also used for passing parameters between subroutines, and as a scratchpad area for temporary storage of data (cf. Fig. 3). The 12-bit wide Return Stack (RET) is addressed by the Return Stack Pointer (`RP`). It is used for storing return addresses of subroutines, interrupt routines and for keeping loop-index counters. It

[2]reduced instruction set computing

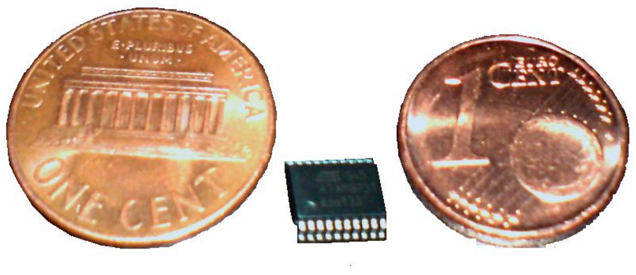


Fig. 4. MARC4 Microcontroller actual size comparison

can also be used as a temporary storage area.

The instruction set supports the exchange of data between the top elements of the Expression Stack and the Return Stack. The two stacks within the RAM have a user-definable maximum depth. The MARC4 controller has six programmable registers and one condition code register. The Program Counter (`PC`) is a 12-bit register that contains the address of the next instruction to be fetched from the ROM. Instructions currently being executed are decoded in the instruction decoder to determine the internal micro-operations. For linear code (no calls or branches), the program counter is incremented with every instruction cycle. If a branch, call, return instruction or an interrupt is executed, the program counter is loaded with a new address. The `PC` is also used with the table instruction to fetch 8-bit wide ROM constants.

The development system used is the *MARC4 Starter Kit* which contains five samples of the *Atmel ATAM893* μC (see Fig. 4), the target application board *T4xCx92*, an *E-Lab ICP V24 Portable* programmer and corresponding MARC4 programming board (see Fig. 5), as well as the software development software (see Fig. 2).

IV. 4-Bit Implementation of the Block Cipher PRESENT

Since the MARC4 inherits a stack-based architecture (which is similar to a zero-address machine) and its instructions contain only the operation to be performed without any source or destination, please note that it was arduous to implement the PRESENT block cipher on the target platform. In fact it turned out to be similar to the *Tower of Hanoi* mathematical game.

To fully understand the code and its effects presented in this section, it is required that the reader is familiar with the Reverse Polish Notation (RPN) as well as the MARC4's stack-based architecture (detailed in [30]).

All variables have to be initialized with a specific value for the MARC4 hardware to function correctly, since no variables are automatically assigned to zero. The 64-bit plaintext (or ciphertext, respectively) input to be encrypted (decrypted) is stored in the nibbles `TextF` to `Text0`, where `TextF` represents the 4 most-significant-bits (MSB) of the input. Hence, we refer to their order from '0' to '9' followed by alphabetical means until the letter 'F'. Naming congruates with the 80 bits of the key variables `KeyJ` to `Key0`. In order to implement the PRESENT round counter the `2VARIABLE` construct is used. It allocates RAM space for storage of one double length (8-bit) value.

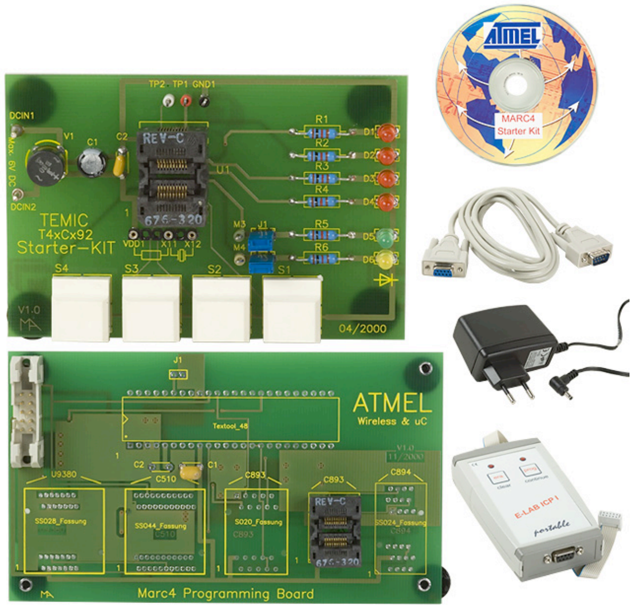


Fig. 5. Atmel MARC4 Starter Kit

```
2VARIABLE  Round
```

The 5 most significant bits of the 8-bit variable `Round` represent the current number of rounds performed. This design decision was made to efficiently implementing the exclusive-or operation (XOR) in the key scheduling function.

A. Round Structure

PRESENT consists of an initial key addition (see Section IV-B1) followed by 31 iterations of :SBOX (see Section IV-B2), :PBOX (see Section IV-B3), :KEYSCHED (see Section IV-B4) and :KEYXOR (see Section IV-B1). Also the 8-bit counter `Round` has to be increased in every iteration.

```
: ENCRYPT                        \ This is a COMMENT
      KEYSCHED
      Round 2@ 8 M+ Round 2!     \ increase Round by 8
```

```
        SBOX PBOX KEYXOR
;
```

The required 31 iterations of :ENCRYPT are performed in a $10 * 3 + 1$ fashion. The word :ENCLOOP applies 10 iterations of :ENCRYPT and is executed 3 times, resulting in 30 iterations of one round of PRESENT. Completing the encryption :ENCRYPT is once again executed, applying the last iteration of the PRESENT encryption.

```
: ENCLOOP
  11 BEGIN                        \ DO 10 times ENCRYPT
        1- DUP   0 > WHILE        \ ...WHILE TOS > 0
        ENCRYPT REPEAT  DROP      \ drop loop-counter=0
;
\ ----------------ENCRYPTION -------------------------
 : INT1                           \  INT1 = encryption
        KEYXOR                    \ Initial KeyXOR
        4        BEGIN            \ DO 3 times ENCLOOP
                 1-  DUP
                 0 > WHILE
                 ENCLOOP          \ DO-END
        REPEAT   DROP
        ENCRYPT                   \  31st Round of PRESENT
;
```

B. Encryption

Encrypting one block of 64 bits with a given 80-bit key can be divided into 4 procedures coinciding with the blocks found in Fig. 1. These procedures are 1) Adding the Key (addRoundKey), 2) Substitution Table (sBoxLayer), 3) Permutation Layer (pLayer) and 4) Deriving the Key (update) and will be detailed in the following.

1) Adding the Key (:KeyXOR): This sub-routine starts with a rather trivial function, the Key XOR. `TextF` places the 8-bit RAM−address of the variable `TextF` as two 4-bit values on to the stack (low nibble is top element). **@** copies the 4-bit value at a specified memory location via the two topmost nibbles onto the top of the stack (`TOS`). **!** stores a 4-bit value at a specified memory location. The two topmost nibbles represent the 8-bit RAM−address where the value being the 3rd element on the stack, is to be stored.

```
: KEYXOR
  TextF @        \ load value of TextF onto stack
  KeyJ @         \ load value of KeyJ onto stack
  XOR TextF !    \  XOR and save back to TextF
  [...]
                 \ all this can also be written in ...
  Text0 @ Key4 @ XOR Text0 !     \  ...one line
;
```

2) Substitution Table (:SBOX): The substitution table consisting of 16 values featured as nibbles stored in the ROM of the MARC4 microcontroller, beginning at ROM−address `0x360h` and ending at `0x36Fh`. This is done with `ROMCONST` which saves 4-bit values in an array at consecutive ROM addresses. Though the MARC4 instruction set contains the function `DTABLE@` which fetches an 8-bit constant from a `ROMCONST` array referring a 12-bit ROM address, a performance gain about 20% is achieved using `ROMBYTE@` instead. Since no carry can occur, the first address (12 bits) of the array `sTable` can be placed onto the Expression Stack, only the lowest 4 bits of the address need to be added up to generate the substitution table lookup-address.

```
ROMCONST sTable 12 , 5 , 6 , 11 , 9 , 0 , 10 , 13 ,
        3 , 14 , 15 , 8 , 4 , 7 , 1     , 2 , AT 360h

  sTable                  \ lookup table base address
  Text0 @ +               \ lookup table index by adding..
                          \ ..value of Text0
 ROMBYTE@                 \ sTable substitute
  Text0 !                 \ save sTable[Text0] to Text0
 DROP                     \ drop high nibble
  [...]
;
```

3) Permutation Layer (:PBOX) : Given the permutation layer's characteristics, it can be implemented using only 16 bits as temporary memory, which are stored in the variables `Temp3` to `Temp0`. The way most efficient approach of implementing the permutation layer was to shift each single bit out of the actual variable, and into its new location. This way of "filling" the variables equates *filo-queueing* and it is therefor necessary to insert the subsequent MSB into the LSB position!

Commands used to *shift into* and *rotating out of* the carry-bit are `SHL, SHR, ROL, ROR` (cf. [31]).

```
Text0 @ SHR Temp0 @ ROR Temp0 !
SHR     Temp1 @ ROR     Temp1 !
SHR     Temp2 @ ROR     Temp2 !
SHR     Temp3 @ ROR     Temp3 ! \ Text0 "emptied"
DROP            \ Drop "empty"  Text0 from stack
```

This way all 16x4 bits of `TextF` to `Text0` are consecutively processed to assign the bits to their new position. Each of these 16 iterations "empties" one 4-bit variable which is subsequently used as the next temporary variable maintaining memory-efficiency.

4) Deriving the Key (:KeySched): Efficient rotation of all 80 key-bits by 61 positions to the left in the Key Schedule is performed by moving 20 bits to the right and then rotate one position to the left. Since 20 is a multiple of the 4-bit architecture, the first operation would actually be only re-addressing memory pointers. As mentioned before the MARC4 hardware is a zero-address machine, therefore re-addressing can only be accomplished by **copying** the respective values *into* their new positions (i.e. variables).

```
: KEYSCHED
Key4@ Key3@ Key2@ Key1@ Key0@ KeyJ@ KeyI@ KeyH@ KeyG@
KeyF@ KeyE@ KeyD@ KeyC@ KeyB@ KeyA@ Key9@ Key8@ Key7@
  Key6 @ Key5@     \ Pushed Keys onto stack   >>20..

SHL      Key0      !      \ Rotate through Carry BEGIN
ROL      Key1      !      \ ..  <<1  matches >>19 ergo
        <<69
 [...]
ROL      KeyJ      !
0111b CCR @ <             \ if Carry, insert 0001b into
      Key0
IF Key0 @ 1 XOR Key0 !
ELSE Key0 @ 0 XOR         \ non-varying RunTime
  Key0 ! NOP NOP NOP      \ --> timing-attack resistance
THEN                      \ Rotate through Carry END
```

The single bits 79 to 76 oblige one SBOX lookup as shown in Section IV-B2, and the 8 bits of the counter `Round` are added to the bits 19 to 15 of the key resulting in the new actual key.

```
Round 2@ Key3@ XOR Key3! Key4@ XOR Key4!  \ Add Counter
```

C. Decryption

Due to MARC4's constrained hardware specifications in terms of available memory, no pre-computing and storing of the round-keys for decryption can be achieved. Therefor before decrypting, the 31st key is pre-computed, and an inverse key scheduling routine is implemented. Though performance takes a slight hit from the extra amount of pre-computing the 31st key, this was the most efficient way to implement the decryption routine without the need for additional external memory.

1) Key Schedule for Decryption and Round Structure: The differences in decrypting data, instead of encrypting, are that instead of the regular S-Box and permutation layer their respective inverses are used in the reverse order of appearance. Furthermore the ***first decryption*** **key** (i.e. the ***last encryption*** **key**) has to be pre-computed, i.e., the complete encryption key-schedule has to be finished, first.

The word `LASTKEY` computes the first decryption key and is therefore iterated $30 + 1$ times. Now that the first decryption key is stored in `KeyJ` to `Key0` the decryption can be started by setting interrupt number seven (`INT7`). The decryption code again coincides with encryption (cf. Section IV-A), except for the counter `Round` being decreased and changing order of the inverse substitution table with the inverse permutation layer as described in the following subsections.

2) Adding the Key (:iKeyXOR): Adding the key whilst decrypting coincides with encryption and can be found in Section IV-B1.

3) Substitution Table (:iSBOX): The inverse Substitution table (:ISBOX) of the decryption, is stored at ROM−addresses `0x370h` and ending at `0x37Fh`. The same characteristics of the :SBOX (see Section: IV-B2) also apply here, except for the inverted substitution table `IsTable`.

```
ROMCONST IsTable 5 , 14 , 15 , 8 , 12 , 1 , 2 , 13 ,
            11 , 4 , 6 , 3 , 0 , 7 , 9 , 10 , AT 370h
```

4) Inverse Permutation Layer (:iPBOX): The code of the inverse substitution layer (:iPBOX) logically equals the code of the :PBOX (see Section IV-B3) with only minor changes of sorting.

5) The Inverse Key Schedule (:iKEYSCHED): Presumable the *last key* is already stored in the variables `KeyJ` to `Key0`, the inverse key scheduling shares most of its code with the key scheduling routine of the encryption (cf. Section IV-B4), just the other way round. It starts with the exclusive-or with `Round` and the inverse substitution table lookup (see Section IV-C3) of the bits 79 to 76. The way of shifting all key-bits 61 positions to the right is again done the same way as explained in Section IV-B4, only changed in its order to acquire the inverse shifting direction.

PRESENT_FINAL:	**ENCRYPTION**	**DECRYPTION**
ROM (lines of code)	841	945
Max. Stack Depth (EXP / RET)	25 / 4	25 / 4
Initialization	230 cycles	230 cycles
Equals (@ 2 MHz)	27.9 ms	32.8 ms
Equals (@ 500 kHz)	111.5 ms	131.1 ms
Equals (@ 16 kHz)	3,483 ms	4,098 ms
Throughput (@ 2 MHz)	2,297 bits/sec	1,952 bits/sec
Throughput (@ 500 kHz)	574 bits/sec	488 bits/sec
Throughput (@ 16 kHz)	18.4 bits/sec	15.6 bits/sec

TABLE III. Implementation results of PRESENT-80 **on the ATAM893-D 4-Bit microcontroller.**

D. Results

The results refer to the optimized and most efficient implemented version. Since this is the first state-of-the-art block cipher on a 4-bit microcontroller there are no figures for comparison available. The following table presents a comparison of the achieved performance of the PRESENT-80. The pursued target of implementing PRESENT-80 on a 4-bit microcontroller was to achieve the shortest possible execution time *while* requiring as less resources as possible. Therefor the code-size and maximum growth of both stacks (EXP & RET) are listed as well as the achieved throughput. As one can see from Table III the decryption routine requires 100 lines of code more compared to the encryption routine which is due to the additional key-scheduling and the additional inverse S-Box. The stack growths and also the time for initialization were similar for both encryption and decryption. The encryption of 64 bit plaintext requires $55,734$ cycles which is equivalent to 13.9 ms at 4 MHz. In the theoretical lowest possible cycle rate of 2 KHz this would be equivalent to $27,8$ seconds. The decryption of one data block of 64 bits requires $65,574$ cycles which is equivalent to 16.4 ms at 4 MHz or 32.8 seconds at 2 KHz, respectively.

1.8V was applied as the supply voltage for measurements to obtain the current consumption of the microcontroller. A *Keithley 2001 DMM*[3] [32] was used for measuring while encrypting data on the μC. Table IV shows the power consumption for different frequencies. Please note that these numbers also resemble the power consumption of the decryption respectively. Furthermore,

[3]digital multi-meter

measurements have also been conducted for a supply voltage of 5V, but the measured current consumption was reciprocal proportional to the operating frequency. Since we could not find an explanation for this phenomenon, we decided to remove these figures from Table IV. The microcontroller was either clocked by an external crystal (XTAL) oscillator or using the internal RC-oscillator (cf. table IV, column 2). The following clock-speeds were measured using a 32 kHz XTAL: *16 kHz, 2 kHz, SLEEP*. The 2 MHz frequency was generated using the internal RC-oscillator. Finally, in order to reduce power-consumption an external 4 MHz XTAL was used generating the 500 kHz. We selected the different frequencies in order to show the time-power/energy trade-off that is possible with our PRESENT implementation. While 2 MHz and 2 KHz are the maximum and minimum frequencies of the ATAM893-D we also wanted to provide figures for an implementation that requires less then $200ms$ for encryption/decryption of one block. It turned out that 500 KHz is the lowest possible frequency to reach this goal.

As one can see from Table IV at a supply voltage of 1.8V the current consumption of the microcontroller is below 10 μA when clocked at 2 KHz, 16 KHz and 500 KHz. This is an interesting result, because it indicates that this implementation can also be used for passively powered low-cost RFID-tags, which typically require such harsh power constraints. At 500 KHZ is the best energy per bit ratio, which is interesting for active devices in order to maximize the lifespan of the battery.

ENCRYPTION	CLK	Current Consumption	Energy per Block	Energy per Bit
1,8V @ SLEEP	ext	0.86 μA	n/a	n/a
1.8V @ 2 kHz	ext	6.09 μA	305.8 μJ	4.8μJ
1.8V @ 16 kHz	ext	9.2 μA	57.7μJ	0.9μJ
1.8V @ 500 kHz	ext	6.7 μA	1.3 μJ	**0.02** μJ
1.8V @ 2 MHz	int	79.3 μA	4.0 μJ	**0.07** μJ

TABLE IV. Measurements results of PRESENT-80 **on the ATAM893-D 4-Bit microcontroller.**

V. Conclusion

4-Bit microcontrollers are deployed in a very broad range of everyday life items, ranging from watches, micro-ovens and washing machines to security critical applications such as car tire sensors or one-time PIN generators. Due to the low-power nature of the ATAM893-D microcontroller it is also thinkable that 4-Bit microcontroller will be employed in low-cost RFID-tags. To the best of our knowledge up to now there are no implementation results of cryptographic algorithms for 4-Bit microcontrollers published. In this work we have closed this gap and provided the first implementation results of this kind. We therefore presented a proof-of-concept that state-of-the-art cryptography is feasible on ultra-constrained 4-Bit microcontrollers. Our implementation draws a current of 6.7μA at a supply voltage of 1.8V and a frequency of 500 KHz. Together with the observation that the processing of one data block requires less than 200 ms we conclude that this implementation is interesting for passively powered RFID-tags. Future work is to strengthen this implementation against side-channel attacks such as Differential Power Analysis [33].

Acknowledgement

The authors would like to thank ATMEL Corp. for their support and help on many questions.

References

[1] A. Bogdanov, G. Leander, L. R. Knudsen, C. Paar, A. Poschmann, M. J. Robshaw, Y. Seurin, and C. Vikkelsoe, "PRESENT - An Ultra-Lightweight Block Cipher," in *Proceedings of CHES 2007*, ser. LNCS, no. 4727. Springer-Verlag, 2007, pp. 450 – 466. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-74735-2_31

[2] T. J. Donohue, "The state of american business 2007," United States Chamber of Commerce, Tech. Rep., 2007.

[3] "Havocscope illicit market news," Available online via http://www.havocscope.com, January 2008.

[4] N.A., "World drug report 2005," United Nations Office on Drugs and Crime, Tech. Rep., June 2005, available online via http://www.unodc.org/pdf/WDR_2005/volume_1_web.pdf.

[5] T. Staake, F. Thiesse, and E. Fleisch, "Extending the EPC network: the potential of RFID in anti-counterfeiting," *Proceedings of the 2005 ACM symposium on Applied computing*, pp. 1607–1612, 2005.

[6] P. Tuyls and L. Batina, "RFID-tags for Anti-Counterfeiting," *Topics in Cryptology-CT-RSA*, vol. 3860, pp. 115–131, 2006.

[7] A. Juels, "Rfid security and privacy: a research survey," *Selected Areas in Communications, IEEE Journal on*, vol. 24, no. 2, pp. 381–394, Feb. 2006.

[8] M. Lehtonen, T. Staake, F. Michahelles, and E. Fleisch, "From identification to authentication - a review of RFID product authentication techniques," Printed handout of Workshop on RFID Security – RFIDSec 06, Ecrypt, Graz, Austria, July 2006.

[9] D. Molnar, A. Soppera, and D. Wagner, "A scalable, delegatable, pseudonym protocol enabling ownership transfer of RFID tags," Handout of the Ecrypt Workshop on RFID and Lightweight Crypto, Ecrypt, Graz, Austria, July 2005.

[10] T. Dimitriou, "A lightweight RFID protocol to protect against traceability and cloning attacks," in *International Conference on Pervasive Computing and Communications – PerCom 2006*, IEEE. Pisa, Italy: IEEE Computer Society Press, March 2006.

[11] M. Feldhofer, S. Dominikus, and J. Wolkerstorfer, "Strong authentication for RFID systems using the AES algorithm," in *Workshop on Cryptographic Hardware and Embedded Systems – CHES 2004*, ser. Lecture Notes in Computer Science, M. Joye and J.-J. Quisquater, Eds., vol. 3156, IACR. Boston, Massachusetts, USA: Springer-Verlag, August 2004, pp. 357–370.

[12] D. Bailey and A. Juels, "Shoehorning security into the EPC standard," in *International Conference on Security in Communication Networks – SCN 2006*, ser. Lecture Notes in Computer Science, R. De Prisco and M. Yung, Eds., vol. 4116. Maiori, Italy: Springer-Verlag, September 2006, pp. 303–320.

[13] M. Feldhofer, "An authentication protocol in a security layer for RFID smart tags," in *The 12th IEEE Mediterranean Electrotechnical Conference – MELECON 2004*, vol. 2. Dubrovnik, Croatia: IEEE, May 2004, pp. 759–762.

[14] S. Dominikus, E. Oswald, and M. Feldhofer, "Symmetric authentication for RFID systems in practice," Handout of the Ecrypt Workshop on RFID and Lightweight Crypto, Ecrypt, Graz, Austria, July 2005.

[15] S. Lee, T. Asano, and K. Kim, "RFID mutual authentication scheme based on synchronized secret information," in *Symposium on Cryptography and Information Security*, Hiroshima, Japan, January 2006.

[16] K. Rhee, J. Kwak, S. Kim, and D. Won, "Challenge-response based RFID authentication protocol for distributed database environment," in *International Conference on Security in Pervasive Computing – SPC 2005*, ser. Lecture Notes in Computer Science, D. Hutter and M. Ullmann, Eds., vol. 3450. Boppard, Germany: Springer-Verlag, April 2005, pp. 70–84.

[17] X. G. Gao, Z. A. Xiang, H. Wang, J. Shen, J. Huang, and S. Song, "An approach to security and privacy of RFID system for supply chain," in *Conference on E-Commerce Technology for Dynamic E-Business – CEC-East'04*, IEEE. Beijing, China: IEEE Computer Society, September 2005, pp. 164–168.

[18] H.-J. Chae, D. J. Yeager, J. R. Smith, and K. Fu, "Maximalist cryptography and computation on the WISP UHF RFID tag," in *Proceedings of the Conference on RFID Security*, July 2007. [Online]. Available: http://prisms.cs.umass.edu/~kevinfu/papers/chae-RFIDSec07.pdf

[19] A. Corp., "Flash Version for ATAR080 ATAR090/890 ATAR092/892 and ATAM893-D ," available online via http://www.atmel.com/dyn/resources/prod_documents/doc4680.pdf, 2005.

[20] "Advanced Encryption Standard (AES)," National Institute of Standards and Technology (NIST), Federal Information Processing Standards (FIPS) Publication 197, November 2001, available via csrc.nist.gov.

[21] C. Rolfes, A. Poschmann, G. Leander, and C. Paar, "Ultra-lightweight implementations for smart devices - security for 1000 gate equivalents," in *Proceedings of the 8th Smart Card Research and Advanced Application IFIP Conference – CARDIS 2008*, ser. LNCS, vol. 5189. Springer-Verlag, 2008, pp. 89–103.

[22] C. de Cannière and B. Preneel, "Trivium," Available via www.ecrypt.eu.org/stream.

[23] M. Hell, T. Johansson, and W. Meier, "Grain - A Stream Cipher for Constrained Environments," Available via www.ecrypt.eu.org/stream.

[24] T. Good and M. Benaissa, "Hardware Results for Selected Stream Cipher Candidates," State of the Art of Stream Ciphers 2007 (SASC 2007), Workshop Record, pp. 191 – 204, February 2007, available via www.ecrypt.eu.org/stream.

[25] P. Grabher, J. Großschädl, and D. Page, "Light-weight instruction set extensions for bit-sliced cryptography," in *Cryptographic Hardware and Embedded Systems — CHES 2008*. Springer Verlag LNCS 5154, August 2008, pp. 331–345. [Online]. Available: http://www.cs.bris.ac.uk/Publications/Papers/2000890.pdf

[26] M. Albrecht and C. Cid, "Algebraic Techniques in Differential Cryptanalysis," in *Proceedings of Fast Software Encryption 2009 – FSE 2009*, ser. LNCS. Springer-Verlag, to appear., 2009. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-74619-5_13

[27] B. Collard and F.-X. Standaert, "A Statistical Saturation Attack against the Block Cipher PRESENT," in *Proceedings of CT-RSA 2009*, to appear.

[28] M. Wang, "Differential Cryptanalysis of Reduced-Round PRESENT," in *Proceedings of AFRICACRYPT 2008*, ser. LNCS, no. 5023. Springer-Verlag, 2008, pp. 40 – 49. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-68164-9_4

[29] M. R. Z'Aba, H. Raddum, M. Henricksen, and E. Dawson, "Bit-Pattern based integral attack," in *Fast Software Encryption: 15th International Workshop, FSE 2008, Lausanne, Switzerland, February 10-13, 2008, Revised Selected Papers*, ser. LNCS, vol. 5086. Springer-Verlag, 2008, pp. 363–381.

[30] M. Vogt, "Effcient Software Implementation of Lightweight Cryptography on a 4-Bit Microcontroller," available online via http://www.crypto.rub.de, 2008.

[31] T. S. Corp., "MARC4 4-Bit Microcontrollers - Programmers Guide," available online via http://www.atmel.com/dyn/resources/prod_documents/doc4747.pdf, 2004.

[32] K. Instruments, "7,5-Digit High Performance Multimeter," available online via http://www.keithley.com/data?asset=361, 2005.

[33] P. Kocher, J. Jaffe, B. Jun, *et al.*, "Differential Power Analysis," in *Advances in Cryptology-CRYPTO99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings*, vol. 1666, pp. 388–397.