Small Scale Variants Of The Block Cipher PRESENT

Gregor Leander

DTU Mathematics
Technical University of Denmark

1 Introduction

In this note we define small scale variants of the block cipher PRESENT [1]. The main reason for this is that the running time of some recent attacks (e.g. [2, 3]) remain unclear as they are based on heuristics that are hard or even impossible to verify in practice. Those attacks usually require the full code bock of present to be available and they work only if some independence assumptions hold in practice. While those assumptions are clearly wrong from a theoretical point of view, the impact on the running times of the attacks in question is not clear. With versions of PRESENT with smaller block size it might be possible to verify how those attacks scale for those versions and hopefully learn something about PRESENT itself. In the next section, all details of the toy ciphers are specified, with test vectors given in the appendix.

2 The small scale variants SmallPresent-[n]

The toy ciphers SmallPresent-[n] are based on PRESENT-80 the 80 bit key version of PRESENT. The design is as close to PRESENT as possible while the block size is reduced to $4n$ bits. In particular, SmallPresent-[16] is actually PRESENT-80. SmallPresent-[n] is an SP-network with a sBoxLayer consisting of n copies of the original PRESENT Sbox and a simple bit permutation as the linear pLayer. A key scheduling algorithm produces $4n$ bit round keys from an 80 bit master key. Thus the overall structure of the algorithm, as depicted in Figure 1 is the same as for the original PRESENT. As the purpose of these toy versions is to understand how the running time of certain attacks increases with the number of rounds, we do not specify the number of rounds for any of those toy versions. Moreover, we do not make any restrictions on the number of Sboxes n, however we anticipate that $n = 8$ might be the most interesting case.

The details of the individual functions are described below. Throughout we number bits from zero with bit zero on the right of a block or word.

addRoundKey. This step consists of a simple xor of the current state with the round key. More precisely, given round key $K_i = \kappa^i_{4n-1} \dots \kappa^i_0$ for round

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

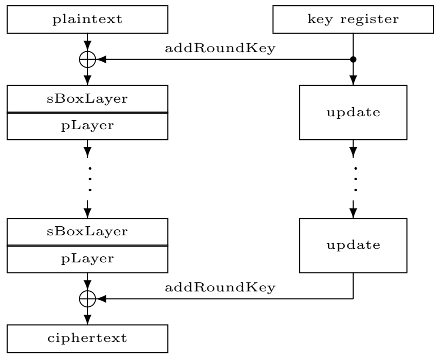

Fig. 1. A top-level algorithmic description of SMALLPRESENT.

i and current STATE $b_{4n-1} \ldots b_0$, addRoundKey consists of the operation for $0 \le j \le 4n-1$,

$$b_j \rightarrow b_j \oplus \kappa_j^i.$$

sBoxlayer. The S-box used in SMALLPRESENT is the 4- to 4-bit S-box $S : \mathbb{F}_2^4 \rightarrow \mathbb{F}_2^4$ already used in PRESENT. The action of this box in hexadecimal notation is given by the following table.

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	1	2

For sBoxLayer the current STATE $b_{4n-1} \ldots b_0$ is considered as n 4-bit words $w_{15} \ldots w_0$ where $w_i = b_{4*i+3}||b_{4*i+2}||b_{4*i+1}||b_{4*i}$ for $0 \le i \le n-1$ and the output nibble $S[w_i]$ provides the updated state values in the obvious way.

pLayer. The bit permutation used in SMALLPRESENT-[n] is given by the following function. Bit i of STATE is moved to bit position $P(i)$ where

$$P(i) = \begin{cases} n \times i \bmod (4n-1) & \text{for } 0 \le i < 4n-1 \\ 4n-1 & \text{for } i = 4n-1 \end{cases}.$$

As $4n = 1 \bmod 4n-1$ its inverse can be described as

$$P^{-1}(i) = \begin{cases} 4 \times i \bmod (4n-1) & \text{for } 0 \le i < 4n-1 \\ 4n-1 & \text{for } i = 4n-1 \end{cases}.$$

We note that for $n = 16$ this is exactly the linear transformation used in PRESENT. Moreover, it is not hard to see that for any n the corresponding bit permutation ensures an optimal diffusion, i.e. each bit of the state depends on each input bit after $\lceil \log_4(n) \rceil + 1$ rounds. The action of the pLayer is also depicted in Figure 2.

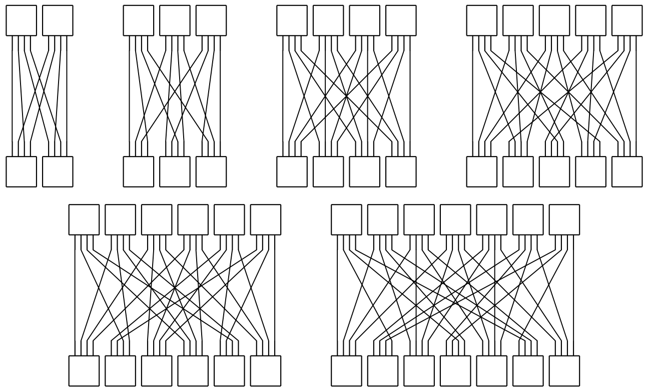

Fig. 2. The pLayer for $n = 2$ to $n = 7$.

The key schedule. The key scheduling of SmallPresent is identical with the key scheduling of PRESENT with the only difference that the round key consists only of the $4n$ rightmost bits of corresponding round key of present-80 (the 64 leftmost bits of the current contents of the key register are used in present). This is done to simplify implementing SmallPresent given an existing implementations of present. For details of the key scheduling we refer to [1].

References

1. A. Bogdanov, L.R. Knudsen, G. Leander, C. Paar, A. Poschmann, M.J.B. Robshaw, Y. Seurin, and C. Vikkelsoe. PRESENT: An ultra-lightweight block cipher. *Lecture Notes in Computer Science*, 4727:450, 2007.
2. J.Y. Cho. Linear Cryptanalysis of Reduced-Round PRESENT. In *Topics in Cryptology-CT-RSA 2010, The Cryptographers' Track at the RSA Conference 2010, San Francisco, CA, USA, March 1-5, 2010*. Springer, 2010.
3. B. Collard and F.X. Standaert. A statistical saturation attack against the block cipher PRESENT. In *proceedings of CT-RSA*. Springer, 2009.

A Testvectors

Below we list test vectors for SmallPresent-[n] for $n \in \{2, 4, 8, 16\}$. For all versions we used the all zero key and the all zero plain text.

Table 1. $n = 2$ (plaintext= 0 key=0)

round	state	key	state xor key	S(state xor key)
0	00	00	00	cc
1	f0	00	f0	2c
2	58	01	59	0e
3	54	01	55	00
4	00	62	62	a6
5	9c	2a	b6	8a
6	c4	33	f7	2d
7	59	5b	02	c6
8	b4	4c	f8	23
9	0d	84	89	3e
10	5e	55	0b	

Table 2. $n = 4$ (plaintext= 0 key=0)

round	state	key	state xor key	S(state xor key)
0	0000	0000	0000	cccc
1	ff00	0000	ff00	22cc
2	33c0	0001	33c1	bb45
3	c3cd	0001	c3cc	4b44
4	4b44	0062	4b26	986a
5	d238	002a	d212	7656
6	0fda	0033	0fe9	c21e
7	9952	005b	9909	eece
8	ffd0	064c	f99c	2ee4
9	67e0	0284	6564	a0a9
10	b0a1	0355	b3f4	

Table 3. $n = 8$ (plaintext= 0 key=0)

round	state	key	state xor key	S(state xor key)
0	00000000	00000000	00000000	cccccccc
1	ffff0000	00000000	ffff0000	2222cccc
2	0f0ff000	00000001	0f0ff001	c2c22cc5
3	a6a75801	03000001	a5a75800	f0fd03cc
4	b3b3a4b4	01400062	b2f3a4d6	862bf97a
5	9d4a7b1e	0180002a	9cca7b34	e44fd8b9
6	9ff8921b	02c00033	9d389228	e7b3e663
7	a8ceff71	3240005b	9a8eff2a	ef31226f
8	c1c3ef71	1400064c	d5c3e93d	704b1eb7
9	16a5979b	1a800284	0c25951f	c460e052
10	88ea2902	2f400355	a7aa2a57	

Table 4. $n = 16$ (plaintext= 0 key=0)

round	state	key	state xor key	S(state xor key)
0	0000000000000000	0000000000000000	0000000000000000	cccccccccccccccc
1	ffffffff00000000	c000000000000000	3fffffff00000000	b2222222cccccccc
2	80ff00ffff008000	5000180000000001	d0ff18ffff008001	7c22532222cc3cc5
3	4036c837b7c88c09	60000a0003000001	2036c237b4c88c08	6cba46bd894334c3
4	73c2cd26b6192359	b0000c0001400062	c3c2c126b759233b	4b46456a8d0e6bb8
5	41d7be58531e4446	900016000180002a	d1d7a858529e446c	757df30306e199a4
6	182ef861ad62fd1c	0001920002c00033	182f6a61afa2fd2f	5362afa5f2f62762
7	0ea0a5b67effc5a4	a000a0003240005b	aea005b64cbfc5ff	f1fcc08a94824022
8	bba0b848a113e080	d000d4001400064c	6ba06c48b513e6cc	a8fca493805b1a44
9	fa943423a9142338	30017a001a800284	ca954e23b39421bc	4fe0916b8be96584
10	69f2e22d63684d54	e01926002f400355	89ebc42d4c284e01	