

Hardware attacks against SM4 in practice


hardware.io
NETHERLANDS 2022

Who are we ?

Sylvain PELISSIER

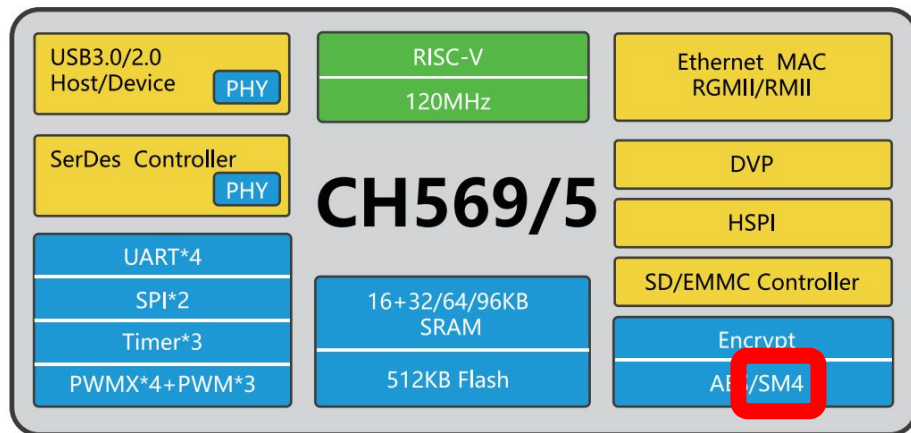
- Security researcher
- Applied Cryptography
- Hardware attacks
- CTF player
-  @Pelissier_S

Nicolas OBERLI

- Embedded systems evaluation
- Hardware attacks
- BlackAlps co-organizer
- Hydrabus
-  @baldanos

Once upon a time

- Discussed with a friend about a new chip
- More interested in security features



SM4 ?

- Wikipedia: “ block cipher used in the Chinese National Standard [...]”
- Never heard about it

- Started to look for implementation and known attacks

- Some papers exist, but no published tools

Me vs math

- Not very good at crypto
 - Base algorithm is fine, thanks to C implementations
- Those attack papers are all Chinese to me
 - Literally ;)
- Need a crypto guy !

2.5. 由 $\Delta A_{32} = \Delta X_{32} \oplus \Delta X_{33} \oplus \Delta X_{34} \oplus \Delta rk_{32} = \Delta X_{32}$ 知 ΔA_{32} 中只包含有一个非零字节 e_{32} , 其对应的字节位置为 j , 即有 $\Delta a_{j,32} = e_{32}$.

2.6. 经过 S 盒变换后 ΔB_{32} 中只包含有一个非零字节, 其位置为 j , 即为 $\Delta b_{j,32}$.

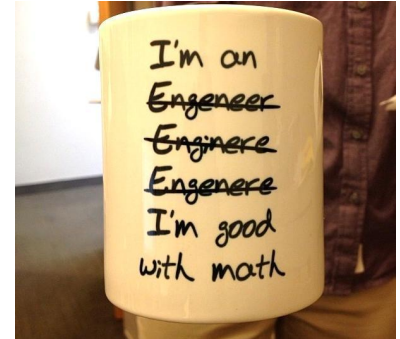
经过 L 变换后有

$$\begin{aligned}\Delta C_{32} &= (\Delta b_{0,32} \Delta b_{1,32} \Delta b_{2,32} \Delta b_{3,32}) \oplus \\ &\quad ((\Delta b_{0,32} \Delta b_{1,32} \Delta b_{2,32} \Delta b_{3,32}) \lll 2) \oplus \\ &\quad ((\Delta b_{0,32} \Delta b_{1,32} \Delta b_{2,32} \Delta b_{3,32}) \lll 10) \oplus \\ &\quad ((\Delta b_{0,32} \Delta b_{1,32} \Delta b_{2,32} \Delta b_{3,32}) \lll 18) \oplus \\ &\quad ((\Delta b_{0,32} \Delta b_{1,32} \Delta b_{2,32} \Delta b_{3,32}) \lll 24) \\ &= \Delta X_{35}.\end{aligned}$$

分析 L 变换中的左移运算, 可见有

SM4 block cipher

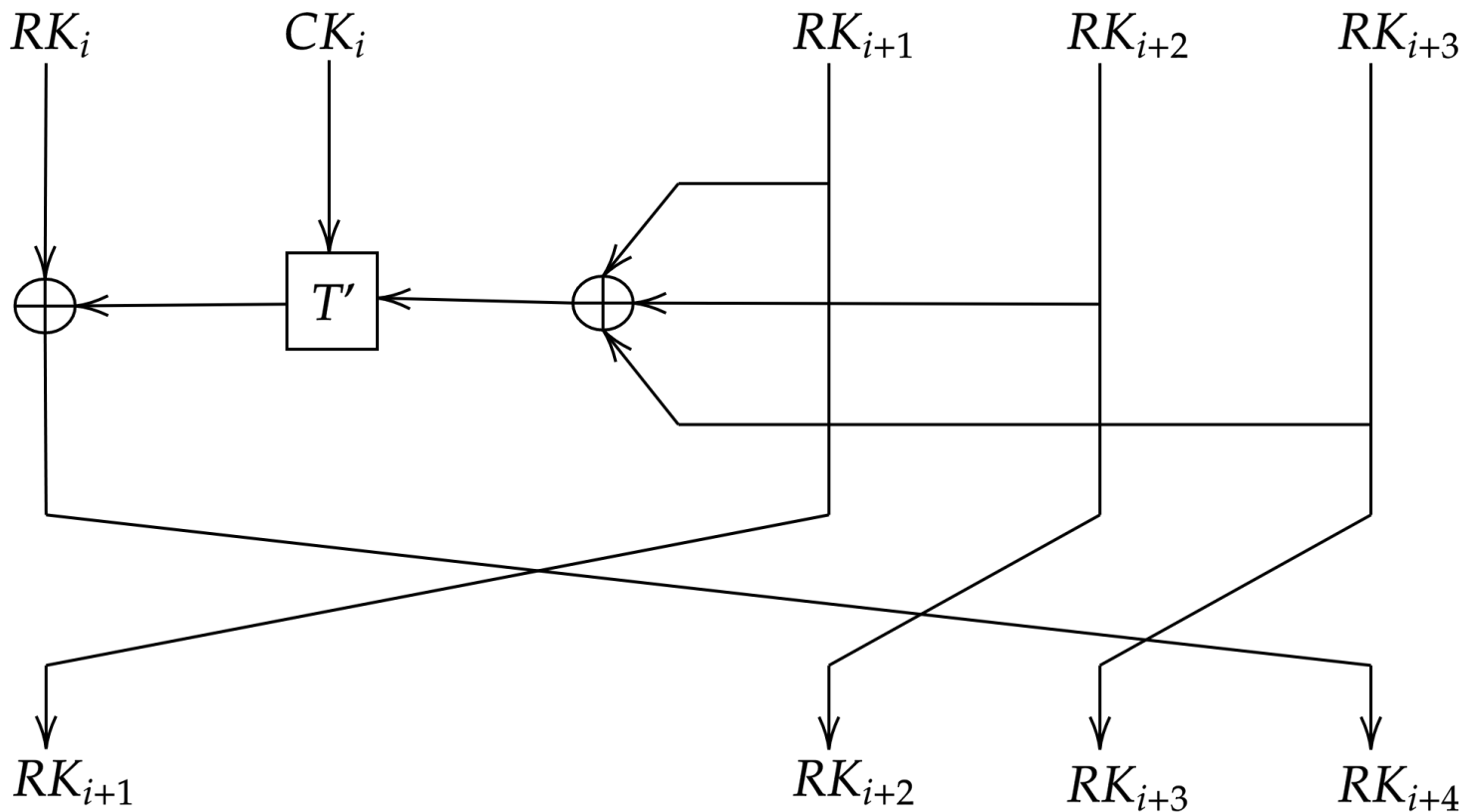
- Block cipher by Standardization Administration of the PRC (GB/T32907-2016).
- Draft IETF available in English [1].
- Key size and a block size of 128 bits.
- Encryption or decryption of one block of data is composed of 32 rounds.
- The algorithm works on 32-bits word.
- Used in Arm v8.4-A and RISC-V and many hardware accelerators.
- Mandatory for some product deployments.



SM4 key schedule

- An invertible key schedule is used to produce the 36 round keys (words) RK.
- Computed from the secret key and constants CK and FK.
- The initial four RKs are the secret key XORed with constant FKs.
- Possibility to identify round keys in memory with their relations like for AES.

SM4 key schedule round



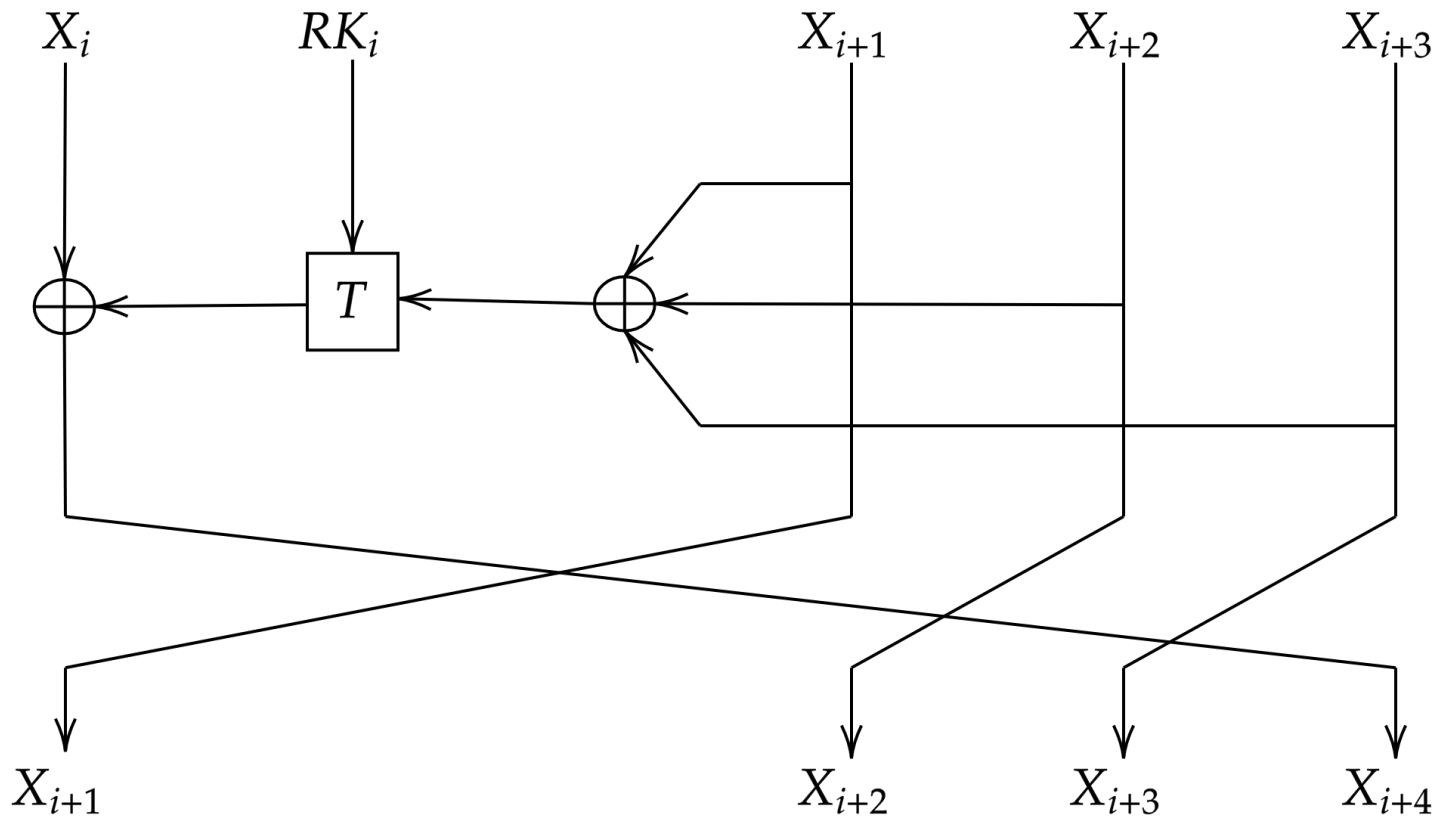
SM4 round

- SM4 encryption or decryption rounds work on a 128-bit state.
- For encryption, the first state is the plaintext:

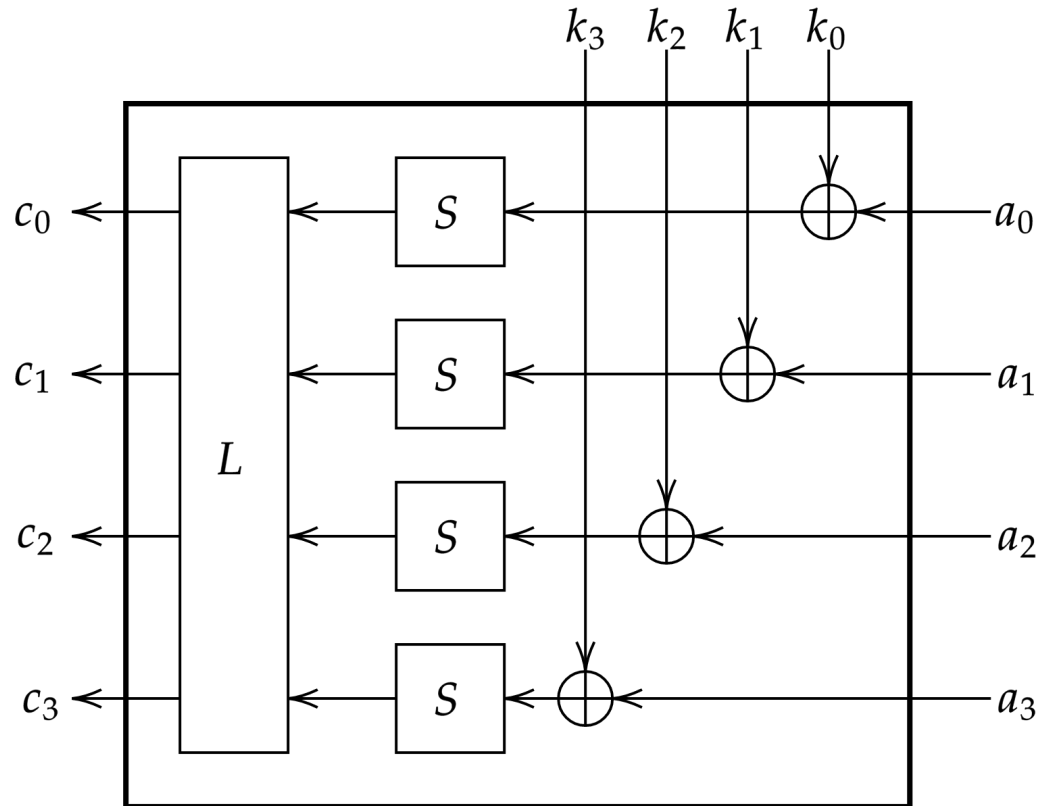
$$(X_0, X_1, X_2, X_3) = (P_0, P_1, P_2, P_3)$$

- The only difference between encryption and decryption is the order of the round keys.

SM4 round



T function



T function

- L is a linear transformation with 32-bit input and 32-bit output. Used for diffusion.
- S are S-Box with 8-bit input and 8-bit output. As AES Sbox, based on inverse and affine transformations.
- Final output of the cipher is reversed:

$$(C_0, C_1, C_2, C_3) = R(X_{32}, X_{33}, X_{34}, X_{35}) = (X_{35}, X_{34}, X_{33}, X_{32})$$

SM4 constants

```
└─┬─┘ ~ yara -s ~/software/rules/crypto/crypto_signatures.yar sm4  
SM4_FK sm4  
0x54990:$c0: C6 BA B1 A3 50 33 AA 56 97 91 7D 67 DC 22 70 B2  
SM4_CK sm4  
0x54910:$c0: 15 0E 07 00 31 2A 23 1C 4D 46 3F 38 69 62 5B 54
```

SM4 round key search

```
[0x00000000]> /ca sm4
```

```
Searching 1 byte in [0x0-0x1ff]
```

```
hits: 1
```

```
0x000000ff hit0_0 f98621f1612b6641db28e44757dbe32c
```

SM4 hardware attacks

- Two possible approaches
 - Side channel analysis (SCA)
 - Differential Fault Analysis (DFA)
- Two people, two approaches, we have a plan !

- First step: software implementation
- Second step: hardware implementation

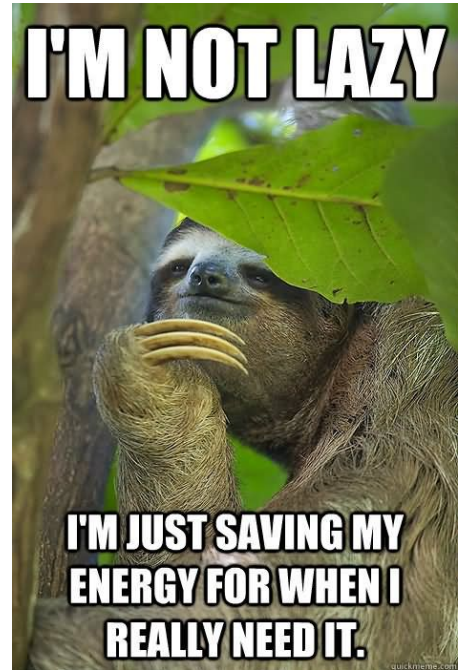
Side-Channel analysis

Side channel analysis

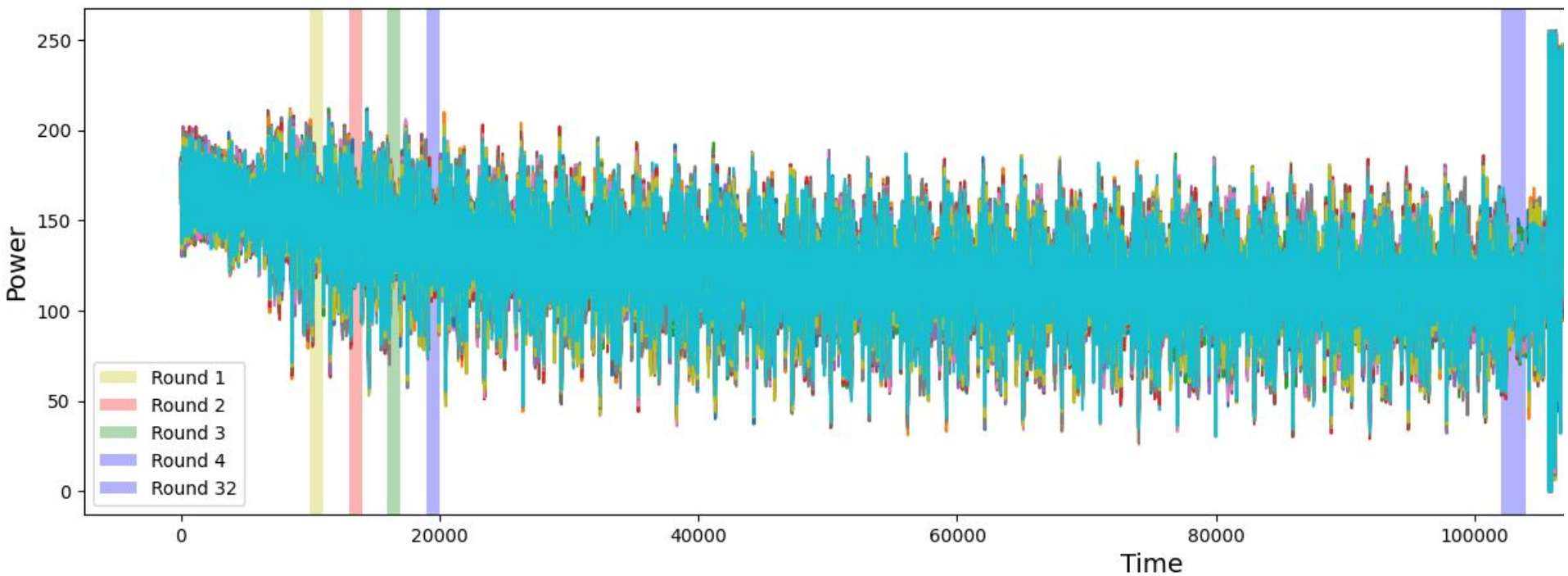
- Recover secret key based on some kind of *leakage*
 - Power consumption / EM emissions / timing / ...
- Multiple papers/tools for various cryptographic algorithms available
 - AES is widely attacked using side-channel analysis
- Some papers about SM4 side-channel analysis
 - Mostly in chinese :(
 - No available tool

Traces

- Need traces to perform analysis
- Software implementation of SM4 in C
- Target : ESP32-C3
- Acquisition method : LISN
 - One set of traces, two talks : ~~Optimization~~ laziness
- Random plaintext
- 50'000 averaged traces



Example traces



SCA library

- Instead of creating a new library, use an existing one
- SCARed by eShard
 - Clean / open-source codebase
 - Good documentation
 - In Python

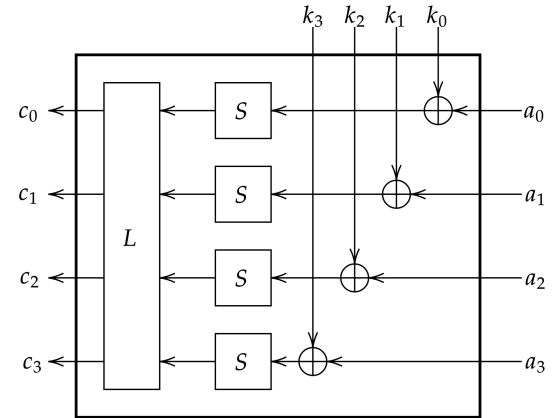
- Started by adding a SM4 implementation
 - Using same inputs/outputs as the AES module
 - Create helper functions
 - Inverse transforms

Selection functions

- Functions used to modelize the hypothesis
- Based on input data and key hypothesis, generate a dataset containing hypothetical values
- Used to perform the actual CPA attack

Sbox selection function

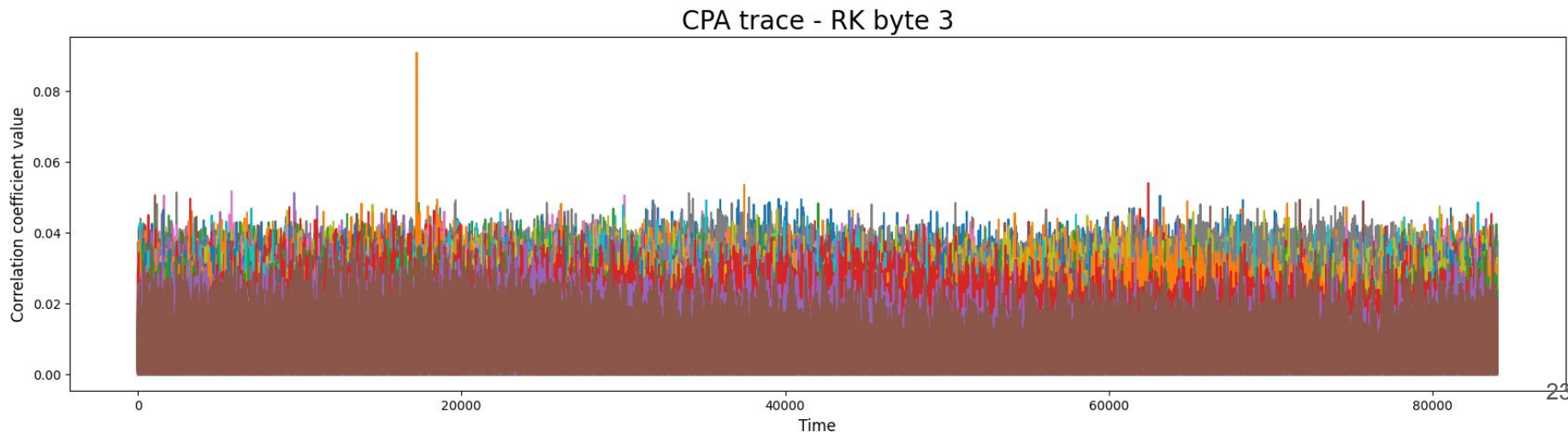
- Sbox is performed per byte
 - Good for CPA
- On first round, Sbox output is dependant on plaintext and round key
 - plaintext is known, round key is our guess
- Helper function takes 4 input words and outputs 4 bytes
- Result is XORed with our guess
- Calculate Sbox output and search for correlation



Sbox selection function results

- Looking great !
- Only one guess arises during computation
 - And is the correct one !

```
[10]: cpa_attack = scared.CPAAttack(  
        selection_function=sm4.selection_functions.encrypt.FirstSubBytes(),  
        model=scared.HammingWeight(),  
        discriminant=scared.maxabs  
    )  
  
    round_container = scared.Container(thz, frame=slice(10000,11000))  
  
    cpa_attack.run(round_container)  
  
[11]: for kb in range(4):  
        KEY_BYTE = kb  
        plt.title('CPA trace - RK byte '+str(KEY_BYTE), fontsize=20)  
        plt.xlabel('Time', fontsize=12)  
        plt.ylabel('Correlation coefficient value', fontsize=12)  
        plt.plot(abs(cpa_attack.results[:, KEY_BYTE, :].T))  
        plt.show()  
  
    cpa_round_key = np.argmax(cpa_attack.scores, axis=0)  
    k1 = int(cpa_round_key.astype(np.uint8).tobytes()[::-1].hex(), 16)  
    print(f"K1 = {hex(k1)}")
```



Recovering round keys

- CPA allows to recover one round key
- Key schedule is invertible, but we need 4 consecutive round keys

- Since we've found one word, we can compute the next round value
- Apply the same attack on next round to get the next round key
- Repeat two more times to get 4 round keys

```
[12]: @scared.attack_selection_function
def second_sbox(plaintext, guesses):
    res = np.empty((plaintext.shape[0], len(guesses), 4), dtype='uint8')
    data = sm4.round_forward(sm4.arr_to_words(plaintext), k1)
    for i, guess in enumerate(guesses):
        res[:, i, :] = sm4.sbox(np.bitwise_xor(sm4.sm4lt(data), guess))
    return res

round_container = scared.Container(ths, frame=slice(13000,14000)) 24
```


Recover the master key

- From 4 consecutive round keys, revert the algorithm to retrieve RK[0-3]
- Apply FK constant to retrieve the key

```
[27]: l = np.array([k1, k2, k3, k4], dtype=np.uint32)

#Last round key is the fourth. Counting from 3 to 0 to recover RK[0-3]
for i in range(3, -1, -1):
    l = sm4.inv_key_schedule(l, i)
# Apply FK to retrieve the master key
MK = sm4.master_key(l)
for i in range(4):
    print(f"MK{i} = 0x{MK[i]:08x}")
```

MK0 = 0x01234567

MK1 = 0x89abcdef

MK2 = 0x12345678

MK3 = 0x9abcdef0

Last round SBOX

- Last round SBOX can be retrieved from the ciphertext as well.
- Knowing ciphertext, possible to retrieve the sbox values and guess last round key
- Don't forget that ciphertext words are shifted !

```
@scared.attack_selection_function
def last_sbox(ciphertext, guesses):
    res = np.empty((ciphertext.shape[0], len(guesses), 4), dtype='uint8')
    for i, guess in enumerate(guesses):
        res[:, i, :] = sm4.sbox(np.bitwise_xor(sm4.inv_sm4lt(
            np.flip(sm4.arr_to_words(ciphertext), axis=1)), guess))
    return res

round_container = scared.Container(th, frame=slice(102000,104000))
```

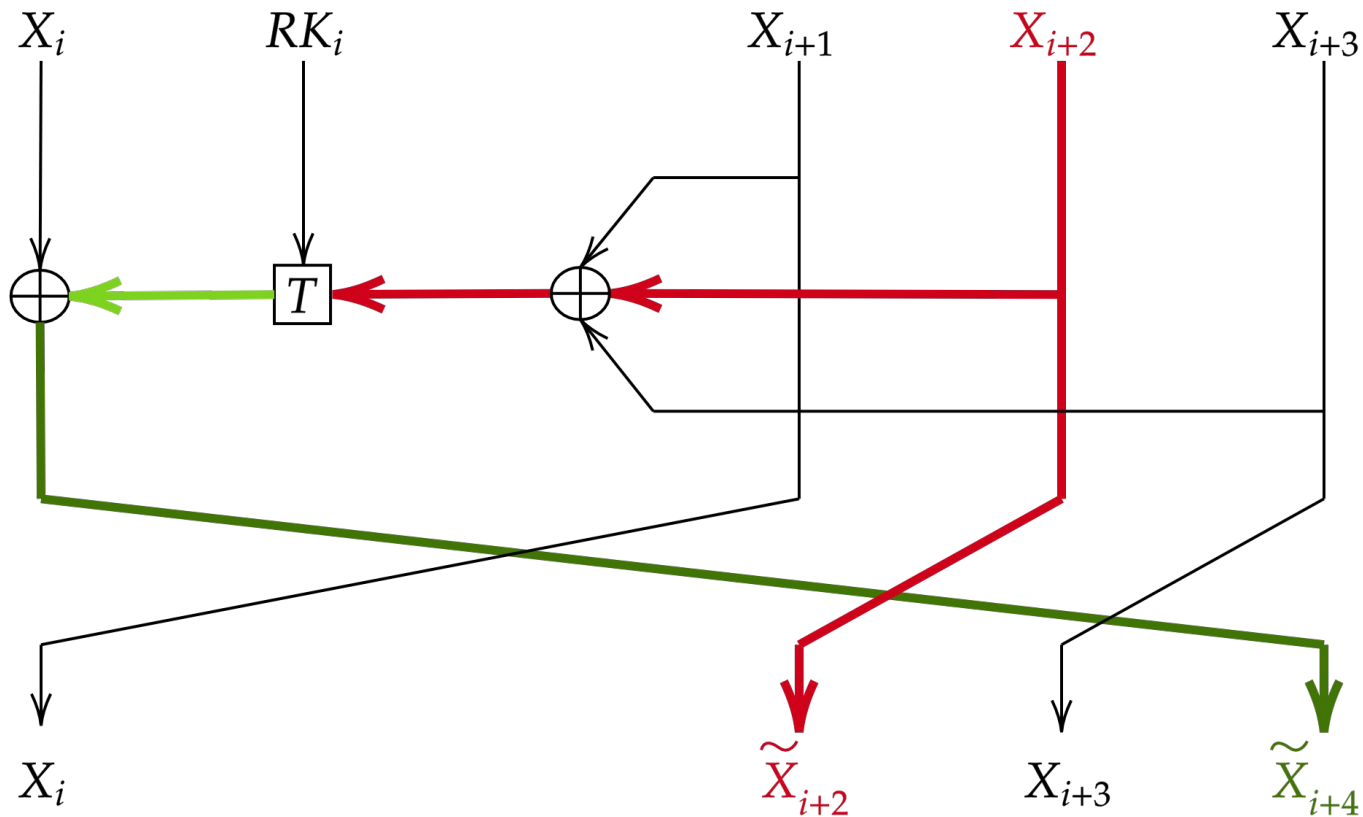
Demo time !

Differential Fault Analysis

Fault attacks on SM4

- First fault attack was introduced in 2006 by Zhang and Wu (paper in Chinese language) [1].
- All further fault attacks are based on this one.
- One byte fault in a word before the last round.
- No tools available from the paper but attack was later implemented by Guojun Tang:
 - https://github.com/guojuntang/sm4_dfa

First fault attack on SM4



First fault attack on SM4

- The faulted output are easily distinguishable:

```
for c in faults:  
    print(color_diff(c, ref))
```

```
2db4b4990000000000000000000000000000000000000000e5  
2d2d674a000000000000000000000000000000000000009100  
e40000000000000000000000000000000000000000000000  
7373acdf00000f00000000000000000000000000000000  
0f0fcfc000008900000000000000000000000000000000
```

First fault attack on SM4

For the round input we have a fault $\tilde{x}_{i+2} = x_{i+2} \oplus \alpha$ in the second byte of X_{i+2} :

$$S(x_{i+3} \oplus \tilde{x}_{i+2} \oplus x_{i+1} \oplus k) \oplus S(x_{i+3} \oplus x_{i+2} \oplus x_{i+1} \oplus k) = S(x) \oplus S(x \oplus \alpha)$$

For the output we have:

$$L^{-1} \left(\tilde{X}_{i+4} \oplus X_{i+4} \right) = (0, \beta, 0, 0)$$

So we are searching x for given known α and β such that:

$$S(x) \oplus S(x \oplus \alpha) = \beta$$

First fault attack on SM4

- We can create a static table T such that

$$T[\alpha][\beta] = \{x : S(x) \oplus S(x \oplus \alpha) = \beta\}$$

- We collect faulted ciphertexts.
- We compute α and β for each faulted ciphertext.
- The entry $T[\alpha][\beta]$ give us the list of possible round key byte candidates.
- Statistically for each faulted ciphertext we would obtain 2 candidates.
- We need 8 different faults to the full round key.

First fault attack on SM4

- After a round key is recovered we can decrypt the last round and apply again the attack.
- With 4 round keys, we can invert the key schedule and recover the full secret key.
- We need on average 32 faults in total.

Test

- First Test on simulated faults.
- Test ARM binary using a C implementation of SM4
- Used our fault simulation tool based on radare2
 - Fault model : instruction skip
- Fixed key, fixed plaintext
 - Sequentially skip one instruction and print result



Fault simulation output

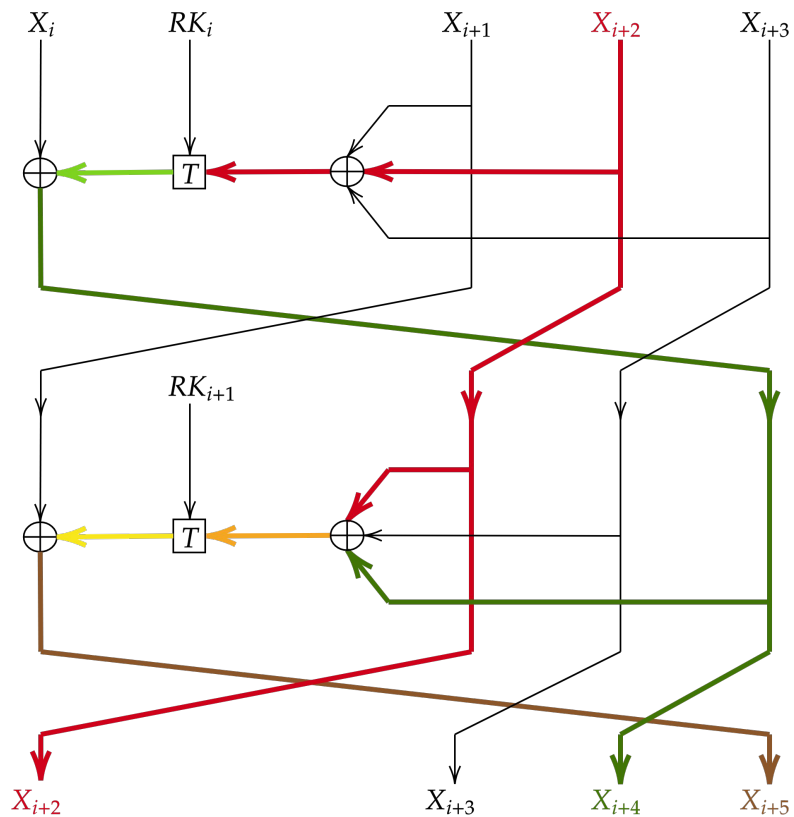
```
simulated_faults
1912 Skip adds r3, r7, r4 @ 0x8222(count=17680) 923a700b4a033411275beb17ce749e3e
1913 Skip ldr r3, [r7, 4] @ 0x8226(count=17682) 888152334a033411275beb17ce749e3e
1914 Skip lsls r3, r3, 0x18 @ 0x8228(count=17683) 54fe2d904a033411275beb17ce749e3e
1915 Skip strb r2, [r3, 3] @ 0x822e(count=17686) b061b2eb4a033411275beb17ce749e3e
1916 Skip ldrb r3, [r3] @ 0x8232(count=17688) 5f88f8ed4a033411275beb17ce749e3e
1917 Skip movs r0, r3 @ 0x8234(count=17689) fefc2d384a033411275beb17ce749e3e
1918 Skip bl sym.sm4Sbox @ 0x8236(count=17690) 9a4affea4a033411275beb17ce749e3e
1919 Skip push {r7, lr} @ 0x81bc(count=17691) e4b8c3d8bfb97223fdc128a55687fe8a
1920 Skip sub sp, 0x10 @ 0x81be(count=17692) fdc128a55687fe8a32374dc40101e3fe
1921 Skip movs r2, r0 @ 0x81c2(count=17694) fefc2d384a033411275beb17ce749e3e
1922 Skip adds r3, r7, 7 @ 0x81c4(count=17695) 87b0180d4a033411275beb17ce749e3e
1923 Skip strb r2, [r3] @ 0x81c6(count=17696) 87b0180d4a033411275beb17ce749e3e
1924 Skip ldr r3, [pc, 0x20] @ 0x81c8(count=17697) b242dfca4a033411275beb17ce749e3e
1925 Skip adds r3, r7, 7 @ 0x81cc(count=17699) 253832274a033411275beb17ce749e3e
1926 Skip ldr r2, [r7, 0xc] @ 0x81d0(count=17701) b840d7c24a033411275beb17ce749e3e
1927 Skip movs r1, 0xb @ 0x81d4(count=17703) b1be20354a033411275beb17ce749e3e
1928 Skip adds r3, r7, r1 @ 0x81d6(count=17704) 87b0180d4a033411275beb17ce749e3e
```

Demo time !

Extended fault attack on SM4

- First DFA is not applicable in some cases.
- Attack proposed in 2007 by Li and Gu [2].
- Faults happening one round earlier, corrupt all the bytes of the following word.
- We can apply the first fault attack in parallel and recover all the 4 bytes of the round key.
- It transforms a byte fault model into a word fault model.

Extended fault attack on SM4



Extended fault attack on SM4

- We can fault a round before and recover two round keys only with two faults.
- Only four faults allow to recover the secret key completely.



Extended fault attack on SM4

- The faulted output are still distinguishable:

```
for c in faults:  
    print(color_diff(c, ref))
```

```
afe2c4f6cfa2c99e5555015400000000  
716b04fada5a63b72ea7892e00170000  
6fa33644a18e8e2f0000000a00000000  
e9ba5693010404050000001100000000  
8925521706d10c04fe43bdfe00000000
```

- Does not work for some faulted words.

Going further

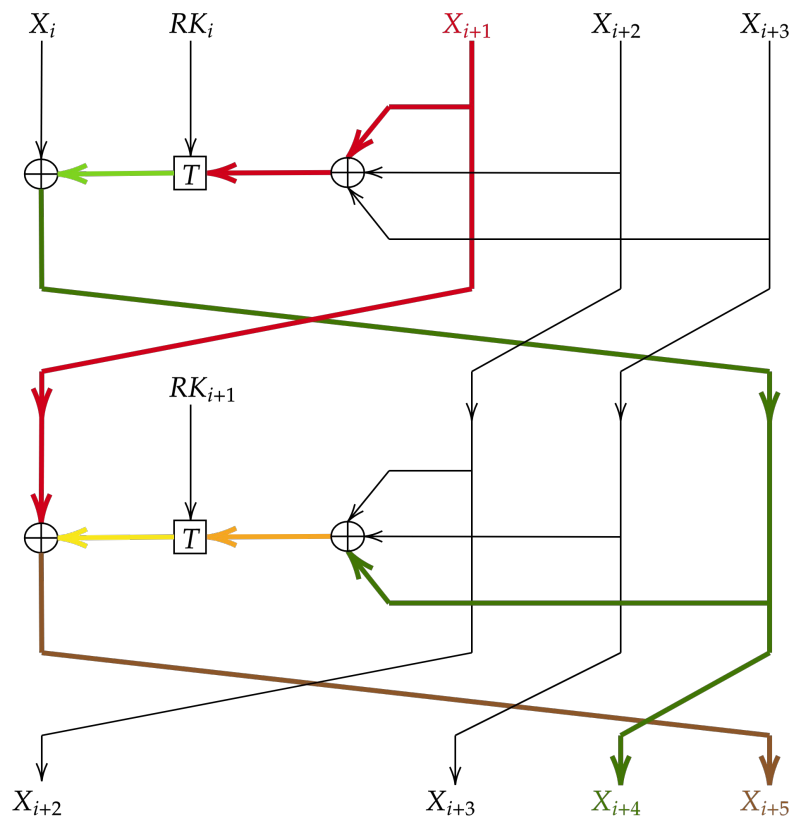
- Attack proposed by Li *et al.* in 2010 [3]
- Attack one round further and bruteforce DFA until the correct round keys are found.
- A single fault is necessary.
- Useful when few faults are available.

```
20 for c in faults:
21     print(color_diff(c, ref))
```

```
67ea114ae838bf59924e71c1a527c762
29bd7d124098b16014cc0cc7f1729263
0cd773fed6508709ef255c4b2e01e1cf
5da97ee4b8d595989d82a197ede60be6
93e8334fdb662ca5469d7ac9e2fc25c
```



Still some room for improvement



Fault attack implementation

- Combine two first DFA attacks and some improvements.
- Inverse the round once the round key is found and continue the attack.
- Included in a tool similar to phoenixAES from Side-Channel Marvels.
- Available as a single Python package:

```
pip install phoenixSM4
```

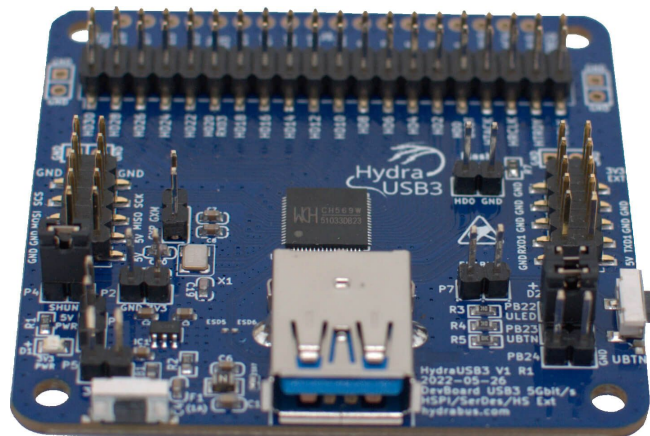


Demo time !

Hardware implementation

Target

- In the meantime, Benjamin released HydraUSB3
- Based on CH569w from WCH
 - RISC-V microcontroller with hardware SM4 IP
- Time to test on real hardware



Firmware

- Simple C firmware
 - Read 16 bytes of plaintext
 - Raise GPIO
 - Perform SM4
 - Reset GPIO
 - Write 16 bytes of ciphertext

```
while(1)
{
    read_buf(plaintext, 16);
    bsp_uled_on();
    ECDC_SingleRegister((uint32_t *)plaintext,
                       (uint32_t *)ciphertext);

    bsp_uled_off();
    phex(ciphertext, 16);
    printf("\r\n");
}
```


Side-Channel analysis

Setup

- Tried shunt and LISN
 - Correlation on plaintext, ciphertext but nothing in between
 - T-test does not show anything interesting

- Maybe my setup is not fast enough

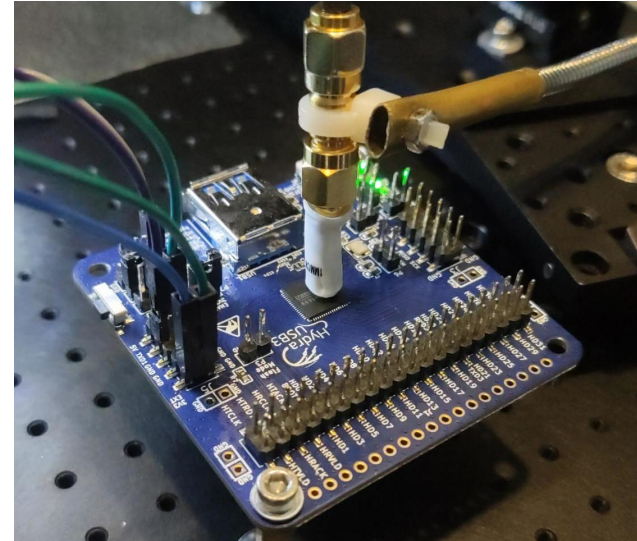
- Work in progress



Fault injection

Setup

- Same board and firmware as with side channel
- EM fault injection with NewAE's ChipShouter
 - 400V / 150ns pulse
 - Stock clockwise coil



Results

- ~1400 faults generated (283 unique ciphertexts)
- Running the DFA tool recovers the key

Round key 32 found:

FE1866AB

Round key 31 found:

5D7F2319

Round key 30 found:

AFC4C1D1

Round key 29 found:

4D2037C3

[2875595006, 421756765, 3519136943, 3275169869]

Master Key found:

0123456789abcdef123456789abcdef0

Summary

- SM4 is another block cipher
- More and more deployed, thanks to standardization
- We provide multiple open-source tools to perform attacks on SM4
 - In-memory key schedule finder included in radare2
 - Power analysis library to be included into SCARed
 - Differential fault analysis tool PhoenixSM4 to be included in the Side channel Marvels
- To see fewer talks like this one, it's easy: **Publish your code !**



Thank you !

References

- [1] Ronald Henry Tse, Wong Wai Kit and Markku-Juhani O. Saarinen, *The SM4 Blockcipher Algorithm And Its Modes Of Operations*, Internet Engineering Task Force, 2018
- [2] Zhang Lei and Wu Wen-Ling, *Differential Fault Analysis on SMS4*, Chinese Journal of Computers, 2006.
- [3] Wei Li and Dawu Gu, *An Improved Method of Differential Fault Analysis on the SMS4 Cryptosystem*, The First International Symposium on Data, Privacy, and E-Commerce (ISDPE 2007), 2007,