Evaluating Cryptanalytical Strength of Lightweight Cipher PRESENT on Reconfigurable Hardware

Jan Pospíšil, Martin Novotný
Department of Digital Design
Czech Technical University in Prague, Faculty of Information Technology
Prague, Czech Republic
Email: pospij17@fit.cvut.cz, novotnym@fit.cvut.cz

***Abstract*—The PRESENT cipher is a symmetric block cipher with 64 bits of data block and 80 (or 128) bits of key. It is based on Substitution-permutation network and consists of 31 rounds. PRESENT is intended to be implemented in small embedded and contactless systems, thus its design needs only small amount of chip area and consumes low power.**

In this work we evaluate the resistance of PRESENT against time-memory trade-off attack. Specifically Rainbow Tables method is used. We determine the computational demand of this type of attack conducted on special parallel reconfigurable hardware COPACOBANA consisting of array of FPGA chips with custom design.

***Keywords*-reconfigurable hardware, FPGA, COPACOBANA, time-memory trade-off, TMTO attack, PRESENT, cryptanalysis, lightweight cipher**

I. Introduction

Lightweight cryptography is a special part of cryptography dealing with security of such products like tickets for public transport, RFID tags identifying the goods in shop, car immobilizers or door opening systems. As many of those systems are powered from electromagnetic field, they must satisfy very strong design constrains on area and power consumption.

Ciphers like Crypto1, KeeLoq or Hitag2 used in lightweight cryptography have been already broken either by means of brute-force attack, algebraic attack or side-channel analysis [1], [2], [3], [4] or [5]. Surprisingly, although these ciphers are insecure either for their short key or for their wrong design, they are still used in many systems.

PRESENT [6] is a symmetric block cipher which was designed as a replacement of above mentioned insecure ciphers. The goal of this work is exploration of its resistance against time-memory trade-off attack. We implement the breaker performing the time-memory trade-off attack based on Rainbow Tables method [7] on reconfigurable platform COPACOBANA, we measure the performance of the breaker and based on these measurements we evaluate the complexity of this attack. We focus only on computation part of the attack (implemented in the COPACOBANA). Data processing part of this attack (storing, sorting and retrieving data) is out of the scope of this work.

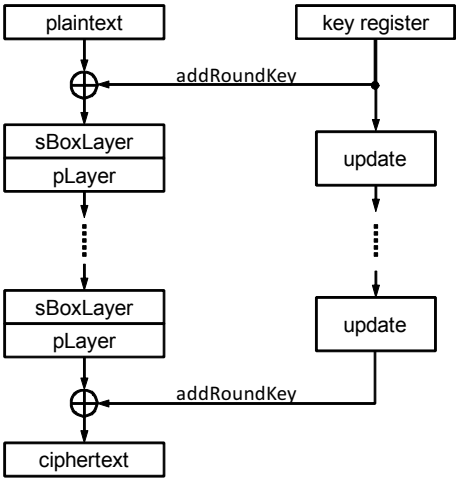

Figure 1. Structure of PRESENT

Table I
VALUES OF S-BOX FUNCTIONS (IN HEXADECIMAL NOTATION)

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

II. PRESENT Cipher

PRESENT is a symmetric block cipher with 64-bit block and 80-bit (or 128-bit) key. The topmost scheme of the cipher can be seen in Figure 1. It is based on Substitution-permutation network which consists of 31 rounds. Each round has three parts: bite-wise XOR of data with the round key, S-box modification of data and permutation of data. Along with the data modification, the round key is modified too (it is done by S-box and XOR with actual number of round).

In the design there are two predefined structures in use: linear bitwise permutation, and S-box. S-box is a non-linear substitution function $S : \mathbb{F}_2^4 \rightarrow \mathbb{F}_2^4$ of 4 bit input and 4 bit output according to Table I. There are 16 identical S-boxes in each substitution layer.

The permutation is a function $P : \mathbb{F}_2^{64} \rightarrow \mathbb{F}_2^{64}$ expressed by Equation 1.

$$\forall i \in \langle 0; 15 \rangle, \forall j \in \langle 0; 3 \rangle : \\ P_{out}(16 \cdot j + i) = P_{in}(j + 4 \cdot i) \quad (1)$$

As all 16 S-boxes in the substitution layer are identical, the design of PRESENT allows trading area for speed in a wide range—from fast pipelined designs down to small integrated circuits suitable for lightweight cryptography.

III. Implementation Platform — COPACOBANA

The COPACOBANA (Cost-Optimized Parallel Code Breaker) machine [8] is a high-performance, low-cost cluster consisting of 120 Xilinx Spartan3-XC3S1000 FPGAs. Currently, COPACOBANA and its successor RIVYERA appear to be the only such reconfigurable parallel FPGA machines optimized for code breaking tasks reported in open literature. Depending on the actual algorithm, the parallel hardware architecture can outperform conventional computers by several orders of magnitude. COPACOBANA has been designed under the assumptions that (i) computationally costly operations are parallelizable, (ii) parallel instances have only a very limited need to communicate with each other, (iii) the demand for data transfers between host and nodes is low due to the fact that computations usually dominate communication requirements and (iv) typical crypto algorithms and their corresponding hardware nodes demand very little local memory which can be provided by the on-chip RAM modules of an FPGA. Considering these characteristics COPACOBANA appeared to be perfectly tailored for our attack.

The structure of COPACOBANA is shown in Figure 2. Via its controller card COPACOBANA is connected to the host computer that controls the attack. The host computer can communicate with each individual FPGA—it can send data or command to the FPGA, it can monitor the status of the FPGA and, upon success, it also obtains found key. The host computer is also able to broadcast data and/or commands to all FPGAs in parallel.

IV. Related Work

Several attacks on PRESENT have already been implemented [10], [11], however, none of them is based on time-memory trade-off (TMTO) tables. Existing mathematical attacks have been mounted on reduced cipher, mostly with round count up to 25. Overview of these attacks can be seen in [10]. COPACOBANA was a target platform for TMTO attacks on DES [8], [12] and A5/1 [8].

TMTO in connection with cryptanalysis was first published by Martin Hellman in 1980 [13]. In his work he introduced the method of precomputed chains stored in memory only by its start point and end point. By doing this, chosen plaintext should be encrypted by every possible key. By the number of chains, its length and the number of tables

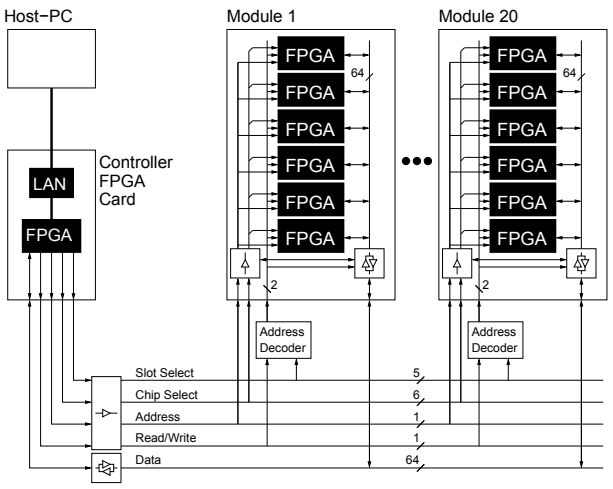

Figure 2. COPACOBANA inner structure (source: [9], modified)

generated in offline phase, attacker can affect the memory needed to store chains and the time needed to find match afterwards in online phase. Later, Ronald L. Rivest introduced modification of the Hellman's method called Distinguished Points [14]. This method utilizes the simply determinable property of some keys to reduce memory accesses in the online phase. In 2003 Philippe Oechslin introduced the new TMTO method, called Rainbow Tables [7]. In our work we are using this method, which is described in the next section.

Another work concerning PRESENT cipher and its implementation recently appeared in [15]. In this work authors present hardware implementation of several lightweight ciphers and compare the results in terms of chip size and power consumption. Target platform used in this work was 90 nm CMOS standard-cell ASIC (with 0.9 V core voltage).

V. TMTO Attack — Rainbow Tables

The time-memory trade-off (TMTO) attacks make trade-off between two extreme approaches, i.e. the brute-force attack and the lookup table one. The brute-force attack utilizes computation power to simply verify every possible key. This method consumes huge amount of time and needs only small amount of memory in general. The lookup table attack is based on precomputed "database" of pairs of keys and corresponding ciphertexts, where all ciphertexts were obtained by encryption of *certain chosen plaintext* with all possible keys. When sorted by ciphertext, searched key can be quickly found in this database. Disadvantage of this type of attack lies in huge amount of memory required for storing all precomputed data. Both of these attack types can be used with known plaintext.

TMTO balances the time and memory requirements together to get benefits of both attacks and make the attack possible. Due to the overhead computation needed to control the attack, real trade-off is not linear. In spite of this, for

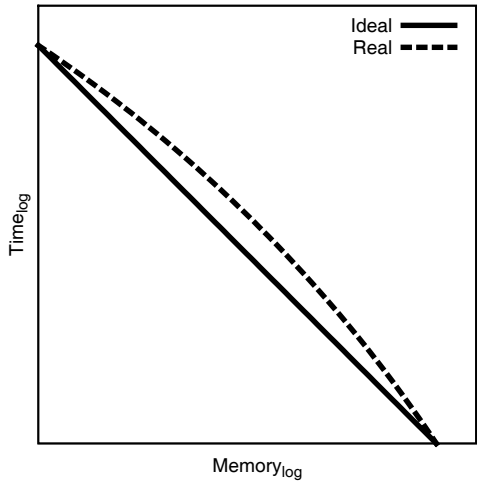

Figure 3. TMTO memory and time dependency (source: [12])

some ciphers under some conditions, trade-off parameters satisfying both the time and the memory demands can be found. The relation of the time and memory in TMTO is shown in Figure 3, adopted from [12].

The TMTO methods (same as the table lookup method, from which it is derived) have two main phases. The first phase, called *offline phase*, is done only once, prior to all attacks. The second phase, called *online phase*, represents the actual attack.

In **offline phase** the data are precomputed. The data are organized in "chains", which are stored in tables, sorted, etc. The amount of calculations performed in the offline phase is comparable to the amount of calculations made during precomputation of lookup table (which is equal to the amount of calculations made during the brute-force attack), however, the amount of stored data is significantly smaller.

When offline phase is completed, we can perform the **online phase** to find the key. In online phase the data prepared by offline phase are used and remaining computations are performed. How much data are used and how many calculations are done depends on selection of TMTO method and its parameters.

A. TMTO History

Hellman demonstrated his method on "chosen plaintext" attack on DES. Chains introduced in [13] consist of encryption steps, where output of one step (the ciphertext) is, after certain modification, used as the key in the next step. In all steps the same (chosen) plaintext is encrypted. The structure of chain can be observed in Figure 4.

We can express one chain link by Equation 2, where C is a ciphertext, P_0 is a chosen plaintext, K is the key used for encryption function $E(K, P)$ and $M(C)$ is a modification function. Finally, $F(C)$ is one whole chain link, the *step-function*, which is the function transforming one key into another.

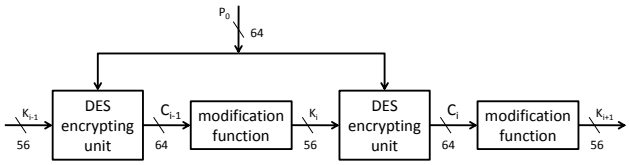

Figure 4. Structure of TMTO chains introduced in [13]

$$\begin{aligned} C_i &= E(K_i, P_0) \\ K_i &= M(C_{i-1}) \\ F(K_i) &= M(E(K_i, P_0)) = K_{i+1} \end{aligned} \tag{2}$$

First key of the chain, K_0, is called a *starting point* SP. The *end point* EP is the last key K_m in the chain, where m is the length of a chain, which is constant for all chains in all tables. Out of each chain only pairs (SP, EP) are stored in the table, which reduces the memory requirements significantly. Pairs are sorted by end points EP. In one table we generate n chains. Example of creation of the table can be observed in Equations 3. Coverage of the table should be $m \cdot n$ keys, but chains in the table can collide together in loops or merges making the real coverage smaller. To have reasonable real coverage, table of size $N^{\frac{2}{3}}$ at most is suggested by [13], where N is the size of the whole key space. Therefore at least $s = N^{\frac{1}{3}}$ tables with different $M()$ functions (to not replicate erroneous states from other tables) have to be generated to have (nearly) the full coverage of the key space.

$$\begin{aligned} SP_0 = K_{0,0} &\xrightarrow{F()} K_{0,1} \xrightarrow{F()} \cdots \xrightarrow{F()} K_{0,m} = EP_0 \\ SP_1 = K_{1,0} &\xrightarrow{F()} K_{1,1} \xrightarrow{F()} \cdots \xrightarrow{F()} K_{1,m} = EP_1 \\ &\vdots \\ SP_n = K_{n,0} &\xrightarrow{F()} K_{n,1} \xrightarrow{F()} \cdots \xrightarrow{F()} K_{n,m} = EP_n \end{aligned} \tag{3}$$

When generating s tables, each of them containing n chains of length m, the typical choice of parameters is $s = n = m = N^{\frac{1}{3}}$. The time needed for offline phase is then N step-functions and memory needed to store all pairs (SP, EP) is $s \cdot n \cdot MEM_{pair} \approx N^{\frac{2}{3}}$ where MEM_{pair} is memory needed to store information about one chain, i.e. the pair (SP, EP). This is significantly less than for full look-up table which consumes $\approx N$ memory. The ideal coverage of the key space would be N (and the success rate would be 100%), however, the real success rate is lower. The success rates for one table S_{table} and for the set of s tables are summarized in Equation 4.

$$S_{table} = \frac{1}{N} \cdot \sum_{i=1}^{n} \sum_{j=0}^{m-1} \left(\frac{N - (i \cdot m)}{N} \right)^{j+1}$$
$$S_{Hellman} = 1 - (1 - S_{table})^{s} \tag{4}$$

In the **online phase** we are given the ciphertext C_x. Assuming this is a product of encryption of a plaintext P_0, we are looking for a key K_x, s.t. $C_x = E(K_x, P_0)$. To do so, we have to identify the chain containing the key K_x first, and then we have to reconstruct the chain to find the key itself. This is done for every table until match is found. At first the ciphertext is modified by the $M()$ function to get the key immediately after K_x in the chain we are searching for: $K_{x+1} = M(C_x)$. The result is compared with all the end point in the actual table. If match occurred, we have found the chain containing the key K_x. To obtain this key we only need to reconstruct the chain from the start point and to check the result for the false alarm. If match did not occur, the key K_x is not in the last column of the table. We need to apply $F()$ function to obtain $K_{x+2} = F(K_{x+1})$ and to compare the result with all endpoints again. If no match appears until we reach the beginning of all chains, another table is used in the same way. For one table the time needed to find the key is m in all cases because the whole chain has to be calculated. In the worst case all tables have to be searched, giving the worst time of the online phase to be $s \cdot m$ steps. The same is the number of table accesses (i.e. table searches), because every single calculated result has to be checked against the known endpoints.

The main **disadvantages** of this method lies in loops and merges which decrease the coverage and can cause false alarms. Merges can occur for example when the size of the key is smaller than the size of the ciphertext, so the $M()$ function have to be reduction (the DES case). On the contrary, loops can occur in any time when the function $F()$ has the same results for different arguments.

Another disadvantage of Hellman's chains is a high rate of table access, which is the most time demanding operation. Method introduced by Ronald L. Rivest [14] addresses this problem. This method is based on so-called **Distinguished Points**, where the distinguished point is a point with certain property that is easy to verify (e.g., six most significant bits are equal to '1'). In this case, the chain is generated until the distinguished point is reached, therefore, the chains do not have fixed length. The attacker has to set a maximum chain length m_{max} to prevent computing loops in chains and also the minimum chain length m_{min} to increase coverage of the chain. As all endpoints are distinguished points, in online phase only such points are searched in the table, which reduces the amount of table accesses.

B. Rainbow Tables

One of the important parameters of both previously stated methods is the modification function. The modification function should be chosen carefully to guarantee the randomness of the $F()$ function because of the key coverage of chains. In 2003 Philippe Oechslin presented in [7] improved TMTO method. In this method, the different modification function $M()$ is used in every step of the chain. This different "stripes" in the table evoke rainbow thereof the name.

Offline phase of this method results in only one table of size $m \times n'$ where $n' = n \cdot s$. It has the same coverage, same memory demands and same time complexity as original Hellman's s tables of size $m \times n$. The different modification functions avoids merges[1] and loops so there are no needs of multiple tables. The fixed length of chains simplifies the computation in both the offline and online phase.

Due to the different modification function in each column, the online phase is little bit complicated. The chain cannot be computed linearly like in previous methods, but for each column it has to be computed all over again. The first few steps of online phase are stated in Equations 5 as long as there is no match. $X_1, X_2, X_3 \ldots$ stands for end point candidates. In the worst case $\frac{m(m-1)}{2}$ steps with m table accesses have to be done. If the match occurs, appropriate chain is reconstructed from the start point.

$$\begin{aligned} X_1 &= M_m(C_x) \\ X_2 &= F_m(M_{m-1}(C_x)) \\ X_3 &= F_m(F_{m-1}(M_{m-2}(C_x))) \\ &\vdots \end{aligned} \tag{5}$$

The success probability with one table of size $m \times n$ is according to [7]:

$$S_{RT} = 1 - \prod_{i=1}^{m} \left(1 - \frac{n_i}{N}\right) \tag{6}$$

where $n_1 = n$ and $n_{i+1} = N \left(1 - e^{-\frac{n_i}{N}}\right)$.

VI. Implementation of our Attack

The design was written in VHDL and synthesized and implemented in Xilinx ISE 11.5. Because of existence of two phases (the offline and the online) we have created two separated designs. The first design (offline) is intended to generate chains and the second one will conduct online phase of the attack.

In [11] we have presented three types of PRESENT core, which can be used. Overview of test results for these types can be found in Table II. For attack presented in this work we have chosen pipelined type of the PRESENT core because of the best chip throughput result.

[1]Merges in Rainbow Tables are possible only if they occur in the same column, but there is very low probability of this.

Table II
COMPARISON OF PRESENT CORE TYPES, [11]

core	simple	pipelined	serial
size	270 kGE	450 kGE	20 kGE
chip area	27%	45%	2%
min. period	83.3 ns	4.4 ns	5.9 ns
max. frequency	12 MHz	227 MHz	170 MHz
clock cycles/result	1	1 ★	32
throughput	0.77 Gbit/s	14.5 Gbit/s	0.34 Gbit/s
cores per chip	3	2	50
chip throughput	2.31 Gbit/s	29.1 Gbit/s	17 Gbit/s

★ with 31 cycles of setup delay (without results)

A. Design overview — offline phase

The main goal of the offline design is to generate chains as described earlier. It will get some subset of all chains that have to be generated and work on this piece. In our design we are using incremental start points so chains can be generated by the counter. Within the chain generation different modification functions have to be used. We have decided to use following modification function:

$$\begin{aligned} newKey = {} & cipherText[15:0] +\!\!+ cipherText \\ & \otimes \\ & round +\!\!+ \{(80 - 2 \cdot CLW) \times \ '0'\} +\!\!+ round \end{aligned} \tag{7}$$

where the $+\!\!+$ symbol means concatenation of two bit vectors and $\otimes$ means bitwise XOR. In brief: part of the ciphertext is duplicated to match the key length and the round value is XORed to the new key in two places. This supports randomness of the modification function, thus improving the key space coverage of the generated chains. From the generated chains only the last step (the end point) is saved into the FIFO to be transferred into controlling application later.

Pipeline core of the PRESENT cipher (as described in [11]) is used and its pipeline structure is utilized to compute 32 chains in parallel. We have fitted two pipelined cores ("workers") into one FPGA. Details of the inner part of the offline design can be seen in Figure 5. The chain generation is divided into three steps. In the first step, pipelines of both cores are filled with the start points generated by *CounterSP* (each core in every cycle uses different start point). This step takes 32 cycles. In the second step the output of the PRESENT cores are (through the modifier) closed in a loop and computation of chains is initiated. Every 32 cycles (when the whole pipeline is processed) the *Counter* is incremented by one to indicate the number of actual round. For the round value equal to length of the chain minus 1 (the last round computed) the third step is executed. In this step the loop is disconnected and the pipeline is "flushed out" to the FIFO, which has capacity of up to 66 end points. When this step is finished, the new set of 64 start points is loaded into cores and computation continues until the value of last SP is reached.

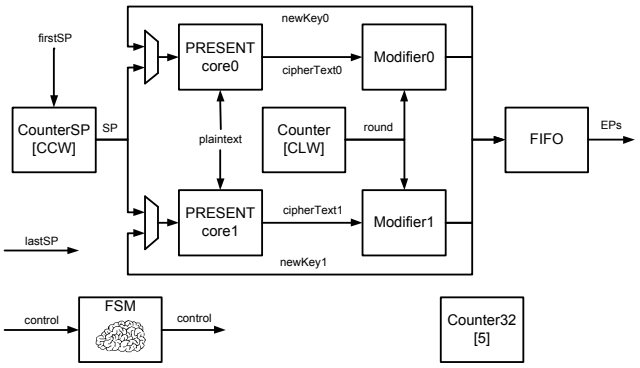

Figure 5. Inner part of the Rainbow Tables offline design

B. Design overview — online phase

The online design is intended for online phase of the Rainbow Tables TMTO attack. According to attack's online phase description the task of this design is to compute one chain stepwise from different rounds and try to identify it with some known chain by comparison of its end points. Because huge amount of data is generated in the offline phase, the identification is not done in the design. The design only computes end points and these are transferred into controlling computer by the same way as generated end points in the offline design.

The overall computation is not linear, i.e. it does not start from the succeeding start rounds. This is due to the data transfer limits of the COPACOBANA communication. Instead of this the "hopping" counter (UpDownCounter) is implemented to distribute the data transfer load.

In this design the pipelined PRESENT core is used too. However, we have implemented only one core in the FPGA to preserve 100 MHz clock. Inner part of the online design is similar to the offline design, see Figure 6. It is adjusted to correspond to Equations 5. Chains are computed in subsets of 32, because one pipelined core is used. Every computed chain from this subset starts from different round. In one subset these starting rounds are succeeding and are derived from the value given by UpDownCounter. This means that chains from rounds $\langle startRound; startRound + 31 \rangle$ are calculated in one subset. For the first starting round one chain is loaded into the first slot of the pipeline. In every next 31 calculated rounds, new chain is started in the next slot of the pipeline (every 33 clock cycles). Filling the whole pipeline takes $33 \cdot 31 = 1023$ clock cycles, which is the start-up delay. After all 32 chains in the pipeline is calculated, the results are saved in the FIFO to be read out by the computer.

The value of the UpDownCounter (startRound) changes after completion of one subset in the "hopping" way. Input parameters for this counter are the first (FR) and the last

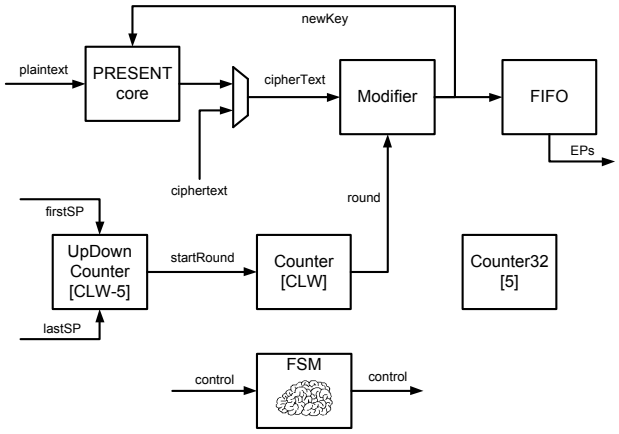

Figure 6. Inner part of the Rainbow Tables online design

(LR) round, which specify the whole range of the chains assigned to the actual FPGA to be calculated. Output value alternates between these two values to cover all intermediate points (multiples of 32). The output of the UpDownCounter can be expressed like this:

$$\{FR, LR, FR+32, LR-32, FR+64, LR-64, \cdots\} \quad (8)$$

When the alternating values of the UpDownCounter meet in the middle of the interval, the computation is finished and the design is ready for the next set of chains to be computed. This hopping processing is implemented to eliminate the data bottleneck of the design — the FIFO.

VII. Results

The above described designs have been implemented and tested at the running frequency of 100 MHz. Results measured on the final designs are divided into two parts — for the offline and online part of the design.

A. Offline part results

The overall time needed to generate all chains is (for fully occupied COPACOBANA):

$$T_{overall} = \frac{1}{120}\left(T_{chain}(CLW)\cdot 2^{CCW} + T_{setup}\cdot 2^{80-spw}\right) \quad (9)$$

Here CLW means chain-length-width and CCW means chain-count-width. These two constants in the design specify the length of each chain (2^{CLW}) and the number of chains (2^{CCW}). Computation of one chain consumes $T_{chain}(CLW) = 2^{CLW}\cdot 5$ ns. Value of 2^{spw} determines the amount of chains calculated by one worker in parallel. This results in 2^{80-spw} setups needed to cover all chains. Every setup takes $T_{setup} = 7{\cdot}93\ \mu s = 651\ \mu s$. Time for data (end points) to be read out from the workers is omitted in the stated calculations. Each worker has FIFO structure which allows data to be read out in the time of the computation of next bulk of chains. This is conditioned by the size of $CLW > 28$ for the fully occupied COPACOBANA.

Table III
Selected value of CLW and corresponding memory and worst time demands for the online phase

CLW	time [years]	memory [TB]
27	0.0244 (= 8.9 days)	131072 (= 128 PB)
30	1.53	16384 TB (= 16 PB)
34	390	1024 (= 1 PB)
35	1559	512
36	6239	256
40	1597287	16

Memory needed for the attack is determined by data generated in offline phase. Amount of these data was not examined in detail in this work. Only rough estimate was made (without overhead of actual storage system) stated as $M = 2^{CCW}\cdot M_{pair} = 2^{CCW}\cdot 2\cdot 64\ bits$.

According to equations stated above, overall time for the whole offline phase of this attack is nearly equal to the brute-force attack, because sum of CCW and CLW should be equal to width of the key of the cipher (for the best coverage). This is about $1.6\cdot 10^6\ years$ [11].

B. Online part results

Overall time needed to crack the given ciphertext for the known plaintext in the online phase of the Rainbow Tables attack (for the fully occupied COPACOBANA) can be expressed as:

$$T_{overall} = \frac{1}{120}\cdot\Big(837\ \mu s\cdot\left\lceil\frac{2^{CLW}}{bulkSize}\right\rceil + 5\ ns\cdot 2^{CLW}\cdot\left(2^{CLW}-32\right) + 2^{CLW-6}\ s\Big) \quad (10)$$

Parameter CLW is described in the previous section. The $bulkSize$ parameter determines number of chains to be calculated by one worker after single setup and thus the setup overhead. For the described PRESENT cipher and CLW set to $\left\lceil\frac{keyWidth}{3}\right\rceil = 27$ we have:

$$T_{overall} \approx \frac{15.6\ minutes}{bulkSize} + 8.89\ days \quad (11)$$

VIII. Conclusion

Goals of this work have been accomplished. We have created the design for TMTO Rainbow Tables attack and we have verified it on the real HW. As stated in Section VII, time and memory complexity of this attack depends on selected parameters and have to be considered specially for offline and online phase of the attack.

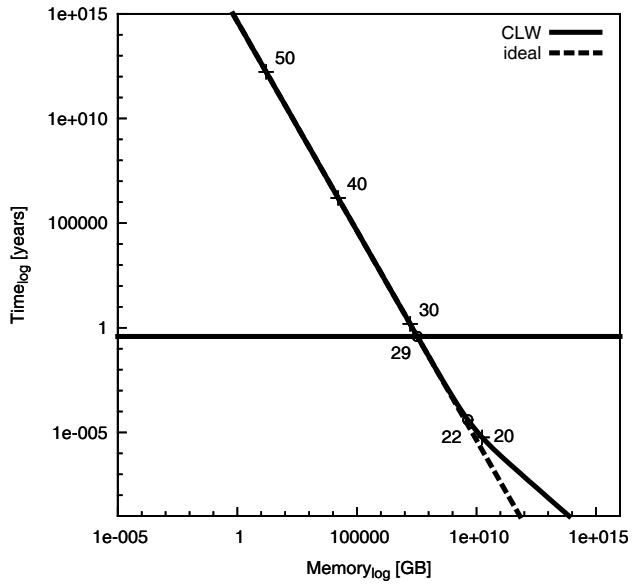

Figure 7. Dependency of memory and time needed in online phase according to CLW

The **offline phase** (during which all data needed later in the online phase are precomputed) lasts the same time as the brute-force attack, which is approximately $1.596 \cdot 10^6$ years with one COPACOBANA. This drawback can be counterbalanced by the greater amount of COPACOBANA machines, because the scalability of our implemented design is very good. This is only the matter of price. The amount of generated data is subject of parameter 2^{CLW} — the length of computed chains. The **online phase** uses data from the offline phase and within the significantly shorter time is able to recover the unknown key. Time needed for the online phase to recover the key is subject of the 2^{CLW} parameter too.

In our work we have computed the time and memory complexity of the implemented attack according to one COPACOBANA. The dependency of these values on the 2^{CLW} parameter can be observed in Figure 7. The ideal TMTO dependency is shown along with the real dependency for our work. The value shown by dotted line is theoretical minimum of the CLW parameter to being efficient. The part of the dependency below this border is shown only for comparison with Figure 3 and to illustrate tendency of the real TMTO dependencies against the ideal ones.

Few "usable" values of CLW are listed in Table III along with time and memory demands. We can see that we can speed up the online phase to last several days, but we need huge amount of data. On the other hand we can use all million COPACOBANAs bought for the offline computation and perform the online phase in 1.5 year with only 16 TB of data needed.

In comparison with [15] our results seem to be two orders of magnitude faster. Authors of cited work stated that their design has throughput-to-area ratio being $0.1211\ Kbps/GE$ for PRESENT cipher, whereas our result is $17.83\ Kbps/GE$ (where GE stands for "gate equivalent"). However, it is hard to compare concretely, because there were different approaches in both of these works and different platforms used. Authors of the cited work were aiming chip area and power consumption at the first place and their results have been proposed to be implemented in 90 nm CMOS standard-cell ASIC running on 100kHz clock. The primary goal of our design was a high speed on a COPACOBANA platform.

Even without revealing any unknown keys, our work has been successful. We have proved that the PRESENT cipher is sufficiently safe for normal usage according to TMTO attacks and nowadays knowledge. Resources spent to break the cipher are too high as stated above.

ACKNOWLEDGMENT

This work has been partially supported by grant No. SGS12/094/OHK3/1T/18 and grant No. SGS10/119/OHK3/1T/18.

REFERENCES

[1] G. de Koning Gans, J.-H. Hoepman, and F. Garcia, "A Practical Attack on the MIFARE Classic," in *Smart Card Research and Advanced Applications*, ser. Lecture Notes in Computer Science, G. Grimaud and F.-X. Standaert, Eds. Springer Berlin / Heidelberg, 2008, vol. 5189, pp. 267–282. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-85893-5_20

[2] T. Eisenbarth, T. Kasper, A. Moradi, C. Paar, M. Salmasizadeh, and M. Shalmani, "On the Power of Power Analysis in the Real World: A Complete Break of the KeeLoq Code Hopping Scheme," in *Advances in Cryptology – CRYPTO 2008*, ser. Lecture Notes in Computer Science, D. Wagner, Ed. Springer Berlin / Heidelberg, 2008, vol. 5157, pp. 203–220. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-85174-5_12

[3] S. Indesteege, N. Keller, O. Dunkelman, E. Biham, and B. Preneel, "A practical attack on KeeLoq," in *Proceedings of the theory and applications of cryptographic techniques 27th annual international conference on Advances in cryptology*, ser. EUROCRYPT'08. Berlin, Heidelberg: Springer-Verlag, 2008, pp. 1–18. [Online]. Available: http://portal.acm.org/citation.cfm?id=1788414.1788415

[4] N. Courtois, S. O'Neil, and J.-J. Quisquater, "Practical Algebraic Attacks on the Hitag2 Stream Cipher," in *Information Security*, ser. Lecture Notes in Computer Science, P. Samarati, M. Yung, F. Martinelli, and C. Ardagna, Eds. Springer Berlin / Heidelberg, 2009, vol. 5735, pp. 167–176. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-04474-8_14

[5] P. Štembera and M. Novotný, "Breaking Hitag2 with Reconfigurable Hardware," in *Proceedings of the 14th Euromicro Conference on Digital System Design*. Los Alamitos: IEEE Computer Society Press, 2011, pp. 558–563.

[6] 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

[7] P. Oechslin, "Making a Faster Cryptanalytic Time-Memory Trade-Off," in *Advances in Cryptology - CRYPTO 2003*, ser. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2003, vol. 2729, pp. 617–630. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-45146-4_36

[8] T. Guneysu, T. Kasper, M. Novotny, C. Paar, and A. Rupp, "Cryptanalysis with COPACOBANA," *IEEE Transactions on Computers*, vol. 57, pp. 1498–1513, 2008. [Online]. Available: http://doi.ieeecomputersociety.org/10.1109/TC.2008.80

[9] S. Kumar, C. Paar, J. Pelzl, G. Pfeiffer, and M. Schimmler, "Breaking Ciphers with COPACOBANA – A Cost-Optimized Parallel Code Breaker," in *Cryptographic Hardware and Embedded Systems - CHES 2006*, ser. Lecture Notes in Computer Science, L. Goubin and M. Matsui, Eds. Springer Berlin / Heidelberg, 2006, vol. 4249, pp. 101–118. [Online]. Available: http://dx.doi.org/10.1007/11894063_9

[10] O. Özen, K. Varıcı, C. Tezcan, and e. Kocair, "Lightweight Block Ciphers Revisited: Cryptanalysis of Reduced Round PRESENT and HIGHT," in *Information Security and Privacy*, ser. Lecture Notes in Computer Science, C. Boyd and J. González Nieto, Eds. Springer Berlin / Heidelberg, 2009, vol. 5594, pp. 90–107. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-02620-1_7

[11] J. Pospisil and M. Novotny, "Lightweight Cipher Resistivity against Brute-Force Attack: Analysis of PRESENT," in *2012 IEEE Symposium on Design and Diagnostics of Electronic Circuits and Systems - to appear*, april 2012.

[12] S. Spitz, "Time Memory Tradeoff Implementation on Copacobana," Master's thesis, Ruhr-Universität Bochum, 2007.

[13] M. E. Hellman, "A Cryptanalytic Time-Memory Trade-Off," *Information Theory, IEEE Transactions on*, vol. 26, no. 4, pp. 401–406, Jul. 1980.

[14] D. E. R. Denning, *Cryptography and Data Security*. Addison-Wesley Publishing Company, Inc., 1982, p. 100.

[15] P. Kitsos, N. Sklavos, M. Parousi, and A. N. Skodras, "A comparative study of hardware architectures for lightweight block ciphers," *Computers & Electrical Engineering*, vol. 38, no. 1, pp. 148 – 160, 2012, special issue on New Trends in Signal Processing and Biomedical Engineering. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0045790611001984