Breaking Hitag2 with Reconfigurable Hardware

Petr Štembera
Czech Technical University in Prague
Faculty of Electrical Engineering, Karlovo nám. 13
Prague, Czech Republic
Email: stembpe1@fel.cvut.cz

Martin Novotný
Czech Technical University in Prague
Faculty of Information Technology, Kolejní 550/2
Prague, Czech Republic
Email: novotnym@fit.cvut.cz

Abstract— **The Hitag2 stream cipher is used in many real-world applications, such as car immobilizers and door opening systems, as well as for the access control of buildings. The short length of the 48-bit secret key employed makes the cipher vulnerable to a brute-force attack, i.e., exhaustive key search. In this paper we develop the first hardware architecture for the cryptanalysis of Hitag2 by means of exhaustive key search. Our implementation on the Cost-Optimized Parallel Code-Breaker COPACOBANA is able to reveal the secret key of a Hitag2 transponder in less than 2 hours (103.5 minutes) in the worst case. The speed of our approach outperforms all previously proposed attacks and requires only 2 sniffed communications between a car and a tag. Our findings thus define a new lower limit for the cloning of car keys in practice. Moreover, the attack is arbitrarily parallelizable and could thus be run on multiple COPACOBANAs to decrease the time to find the secret key.**

Keywords: Hitag2, cryptanalysis, FPGA, reconfigurable hardware, COPACOBANA

I. Introduction

Hitag2 is a stream cipher primarily used in Radio Frequency Identification (RFID) applications, such as car immobilizers. It has been developed and introduced in late 90's by Philips Semiconductors (currently NXP). According to [5], [6], [12], [13], Hitag2 is for example used in access systems for army and government buildings in Germany as well as in RFID car locks, where, by pressing the button, an electronic *tag* sends command to open or close the door of a car. Concerning car security systems, Hitag2 is allegedly used in models produced by car manufacturers such as BMW, Audi, Alfa Romeo, Porsche, Bentley, VW, Peugeot, Renault, Citroën, Iveco trucks and others [12], [13]. It is not clear whether the Hitag2 is still used in newly produced cars, however, for its relatively recent introduction, it is sure that many cars with Hitag2 are still in daily use.

Hitag2 is by its internal structure very similar to its predecessor, i.e., the Crypto1 cipher [3], [4] used in Mifare-Classic cards. Hitag2 uses a 48 bit key and its internal state also has the corresponding length of 48 bits. Due to its internal structure, Hitag2 is vulnerable to algebraic attacks. Due to the relatively short key length of the cipher, brute-force attacks on Hitag2 are feasible and practical.

In this work we introduce our implementation of a parallel brute-force attack implemented on a Field Programmable Gate Array (FPGA) platform, which outperforms previously proposed algebraic attacks. In Sect. II we describe the Hitag2 cipher, its internal structure and the protocol between the transponder and the car. In Sect. III we summarize the related work and the previously proposed algebraic attacks that require almost two days of computation on a standard PC and cannot be parallelized. In Sect. IV we describe the architecture of our attack implemented in reconfigurable hardware. The attack reveals the key in less than two hours in maximum (i.e. less than one hour on average) using a cluster of 120 FPGAs. Unlike algebraic attacks, our attack can be easily scaled, which enables trading the speed of the attack for the amount of resources. The knowledge of the key is required for cloning the tag [13] and unauhorized opening of a car and driving away. In Sect. V we evaluate the performance of our attack and we compare it with the algebraic ones and in the last Sect. VI we conclude with final remarks.

II. Hitag2

According to [7] and [8], Hitag2 RFID chips use base transmit frequency 125 kHz with Biphase or Manchester modulation. The average bit rate for a *reader* (embedded in a car) is 5.2kbit/s and up to 8kbit/s for a *transponder* (embedded in a tag). Data are transmitted bidirectionally in half duplex mode. Hitag2 RFID chips contain 256 bits of data that are divided into 8 pages of 32 bits.

The chips can operate in 3 read-only modes (denoted as Public mode A, B and C) in which data are broadcast in plaintext. These modes are suitable for applications such as animal identification, but they offer no security.

Hitag2 transponders can operate also in so-called *Password mode*, which is mostly used in access systems for buildings. However, this mode also does not provide any security. In this mode the transponder and the reader interchange their passwords, which are always the same and never encrypted. Therefore, implementation of a replay attack is as simple as recording to a tape recorder.

The only mode providing some (weak) security is a so-called *Crypto mode*, which is mostly used in car locks. In this work we solely focus on cryptanalysis of this mode of Hitag2 system.

In Crypto mode the transponder and the reader share common 48 bit secret key. To prevent replay attacks, the unique initialization vector (IV) is generated for every transaction between the reader and the transponder. The secret key, together with the initialization vector and the serial number of

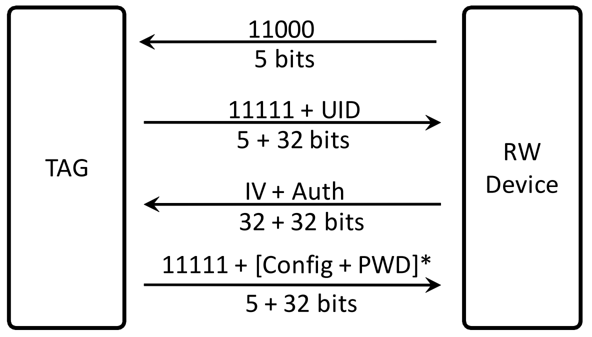
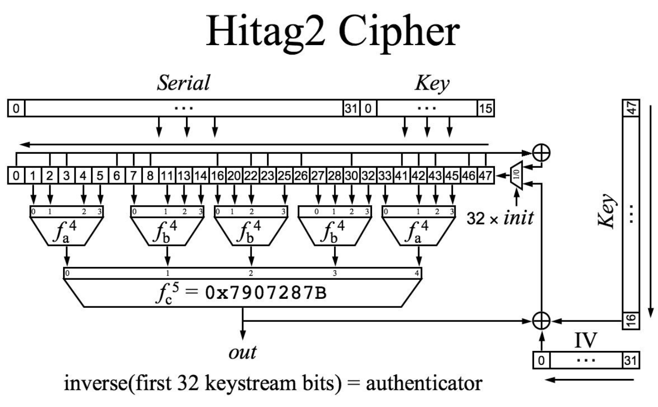


Fig. 1. Hitag2 protocol in *Crypto mode*



f_a^4 = 0x2C79 = abc+ac+ad+bc+a+b+d+1

f_b^4 = 0x6671 = abd+acd+bcd+ab+ac+bc+a+b+d+1

Fig. 2. Internal structure of Hitag2, initialization and encryption

the tag are used for initialization of a cipher. After initialization the Hitag2 cipher produces a keystream. First 32 bits of the keystream are used as an *authenticator*, and the remaining bits are used for encryption like in any standard stream cipher. The details of Hitag2 protocol in the Crypto mode are described below.

A. Hitag2 Protocol in Crypto Mode

The protocol in Crypto mode is depicted in Figure 1. It works as follows:

1) The reader (embedded inside a car) sends the command 11000 to the transponder (embedded in the portable tag).
2) The tag responds by 11111 followed by its 32 bit serial number (SN).
3) Then the reader sends a 32 bit pseudo-random IV and the 32 bit *authenticator*. The authenticator consists of the first 32 bits of the keystream. The authenticator bits are transmitted in reverse order.
4) Transponder computes authenticator as well. If both authenticators match, then the transponder sends 11111 followed by the content of Page 3 of its memory (these are 8 bits of configuration flags and 24 bits of ”transponder password”). Those 32 bits are encrypted by bitwise XOR with the next 32 bits of the keystream.

The initialization vector is randomly generated for every transaction between the reader and the transponder to prevent replay attacks. Hitag2 transponders use challenge-response authentication protocol. It is the reader (car) that generates the initialization vector and proves its identity to the transponder (tag) first by sending the authenticator. This prevents a chosen-IV attack. Moreover, unless the attacker knows the content of Page 3 of the transponder memory, the attacker obtains only 32 bits of the keystream. As the secret key has 48 bits, for successful recovery of this key the attacker needs data from at least two sniffed transactions between the tag and the reader.

Because of low carrier frequency (125 kHz) and low rate, it is very simple to sniff transmitted data by simple antenna circuit [6], [14].

B. Hitag2 Internal Structure

The internal structure of the Hitag2 cipher is shown in Figure 2 adopted from [1]. Hitag2 consists of a filter generator with 48 bit LFSR and a nonlinear function with 20 inputs, producing 1 output bit per 1 clock cycle. The other components are used only at the initialization phase. The reference source code in C language can also be downloaded from [1].

Hitag2 operates with 48 bit secret key, shared between the reader (car) and the transponder (tag), 32 bit serial number of the tag and 32 bit initialization vector IV. The serial number and the initialization vector are, without any encryption, transmitted between the reader and the transponder during their communication.

1) Initialization Phase: At the beginning, the LFSR is loaded with 32 bits of the serial number and 16 lower bits of the secret key (top of Figure 2). Then, in 32 steps the LSFR is filled from right with 32 bits, each of them being a XOR of three bits—one bit of the key, one bit of the randomly generated initialization vector IV, and one bit produced as the output of the boolean function applied to the previous state. The initialization phase takes 32 clock cycles.

2) Keystream Generation: For generation of a keystream, the multiplexer on right is switched to its upper input (feedback from LFSR). At each clock cycle, the LFSR is updated first, and then the output bit is computed as a result of the boolean function of 20 inputs. This boolean function is composed of instantiations of 4-input boolean functions f_a^4 and f_b^4, and a 5-input boolean function f_c^5. The functions are described by their truth tables, where e.g. $f_a^4 = 0x2C79 =$ 0010110001111001 is the content of the truth table—the least significant bit is the output for $f_a^4(0000)$, while the most significant bit is the output for $f_a^4(1111)$.

III. Related Work

Several attacks on Hitag2 have been published in the open literature. Since the cipher had been kept secret (security by obscurity) by the manufacturer, an important achievement was revealing the detailed principle of the Hitag2 cipher [2]. The description of the Cipher together with a reference program code were published in [1]. On the basis of this cipher description, several algebraic attacks evolved.

At the present time there are two known algebraic attacks, as described in [5] and [6]. Both published attacks exploit the low complexity and lack of sufficient non-linearity of Hitag2. In principle, both attacks transform the state of Hitag2 into system of equations. Then the system of equations is transformed into a SAT problem and solved with a SAT solver on a PC.

In an attack described in [5], the authors are able to extract the secret 48 bit key within 6 hours, on the basis of 16 chosen initialization vectors, by running MiniSat 2.0 on a PC. However, as the Hitag2 protocol described above prevents chosen-IV attacks, this attack has to be regarded as theoretical. Another more practical attack needs data from at least 4 sniffed transactions. With these (random) data, the calculations require 45 hours.

Another attack, presented in [6], lacks any detailed description. Therefore it is not clear whether the attack time (6 hours with CryptoSAT) is again valid only for chosen-IV or whether it is valid for any random data. We assume that the stated attack time was only for the theoretical case of chosen-IV.

IV. Our Attack

We have implemented a brute-force attack on Hitag2. The attack works as follows: From one transaction between the car and the tag we get the serial number of the tag, the initialization vector, and the authenticator. Then, we generate and test all 2^{48} keys. Each key we load to the Hitag2 core together with the serial number and the initialization vector, and we generate the authenticator. Generated authenticator we compare with the authenticator obtained from the transaction. If we get the match, then the key loaded to the Hitag2 core becomes a *key candidate*.

As the key has 48 bits, while the authenticator has only 32 bits, it is clear that about 2^{16} key candidates would generate the same authenticator. To select the right key, all those key candidates are then checked against data from another transaction between the car and the reader. These data contain the same serial number of the tag, but the initialization vector and the authenticator are different.

The brute-force attack could be implemented in software, however, the attack time would be extremely long. For example, the Pentium-IV processor is able to check about 2 million keys per second, therefore, the attack time would be about 4 years.

Much better results we obtain when implementing the brute-force attack in hardware. Reconfigurable devices, namely FPGAs, offer ideal platform for implementation of such an attack.

A. Implementation Platform—COPACOBANA

The COPACOBANA (Cost-Optimized Parallel Code Breaker) machine [9] [10] is a high-performance, low-cost cluster consisting of 120 Xilinx Spartan3-XC3S1000 FPGAs. Currently, COPACOBANA and its successor RIVYERA [11] 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.

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.

B. Hardware Architecture of the Attack

To parallelize the attack, the search space is divided into *key subspaces*. Each FPGA is by the host computer assigned with one subspace to search in. If the search in the subspace is finished without any success, then the FPGA is assigned with another subspace. The attack runs until the key is found or until all key subspaces are explored.

In our case the search space is divided into 512 subspaces, which are defined by 9 most significant bits of the key. The FPGA internally generates all 2^{39} keys of assigned subspace. The block-level structure of the Hitag2 breaker implemented in each FPGA can be found in Figure 3. The breaker consists of the *control module* and 256 *Hitag2 executional cores*, denoted as *H2 Core*. Therefore, each FPGA verifies 256 keys concurrently. If any H2 Core produces a key candidate, then this key candidate is passed to the *H2 Core – final* (at the bottom of Figure 3) for verification against data from second transaction between the transponder and the reader.

1) Design Approach: In order to achieve higher performance we exploited some properties of underlying implementation platform (FPGA). However, these properties require special design approach described below.

Look-up tables in Xilinx Spartan-3 FPGAs are dedicated for implementation of combinational logic. However, in some slices the LUTs can also be configured to work as a shift register with a maximum length of 16 bits (denoted as SRL16). This property enables to implement much bigger shift-register-based circuits. For example, using all flip-flops available in used FPGA we may build the circuit containing equivalent of 150 Hitag2 execution cores. Note that in this case all flip-flops would be used for cores and there would be no controller and other circuits. On the other hand, using SRL16s

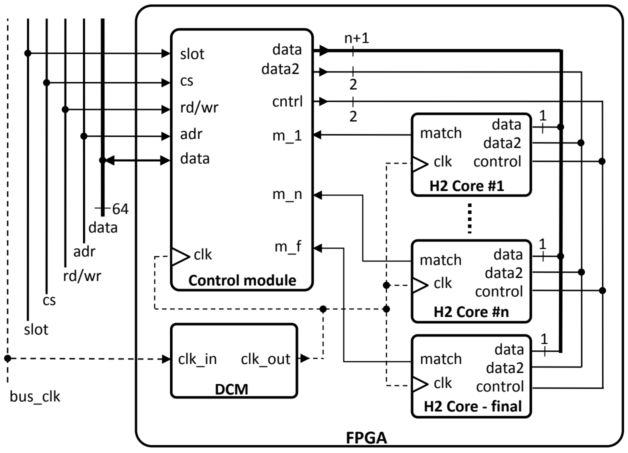


Fig. 3. Block-level structure of the Hitag2 breaker in one FPGA

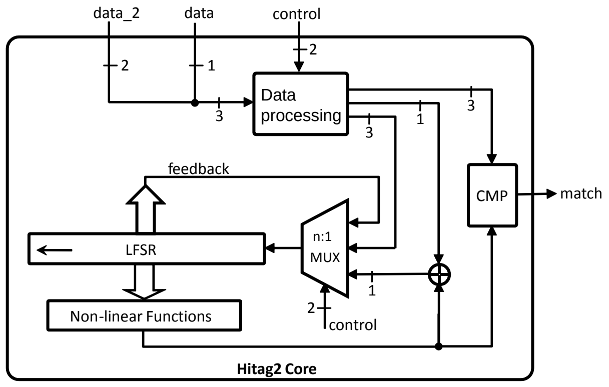


Fig. 4. Hitag2 execution core

we can implement up to 300 Hitag2 cores inside one FPGA, still leaving enough LUTs and flip-flops for a controller and other necessary circuits. However, the usage of SRL16s brings some limitations to the circuit design—we have to avoid any parallel input or output to the shift register. Therefore, we can use only serial input and output. Due to this fact we have not implemented e.g. a pipeline (which was one of design options), since such a pipeline would require parallel access to all bits of registers in each pipeline stage. Instead, we decided to implement an array of small, encapsulated, independent processing units, each having only few serial inputs.

Although up to 300 Hitag2 cores would fit into one FPGA, we have placed there just 256+1 of them. This allowed us significant simplification of the control module. Additionally, we could achieve higher frequency due to more relaxed placement and routing.

2) Control Module: Control module implements interface to COPACOBANA bus and performs communication between the host computer and FPGA. We developed simple communication protocol to ensure data integrity and error detection functions. The module receives commands and data from host application. Based on them it controls processing of assigned subspace.

Another part of control module monitors and controls the process in execution cores. Beside FSM it contains 31 bit counter which is used for generating input data for execution cores.

3) Hitag2 Execution Core: The Hitag2 execution core simulates Hitag2 cipher operation. It is slightly modified to simplify brute-force attack implementation. The structure of the Hitag2 execution core is shown in Figure 4. Execution core vital parts are very similar to Hitag2 chip implementation. They consist of LFSR, non-linear function and few control parts.

Execution core has only 5 input bits and 1 output bit. The input signals consist of 3 data bits and 2 control bits. The control signals are used for selection of one of operation modes (Load, Initialization, Authenticator generation and verification). Control signals are common for all execution cores inside one FPGA. Data signals are further divided into 2 groups—*data* and *data2*. The 2 bit signal *data2* is common for all execution cores and is used only during Load phase to speed up LFSR load. The 1 bit signal *data* is unique for every execution core.

The execution core has only 1 output signal—*match*. This signal is used during Authenticator generation and verification phase to indicate whether successively generated authenticator still conforms to the sniffed one.

4) How it Works: The attack runs in rounds which are divided into 3 phases (Load, Initialization, Authenticator generation and verification). At each round every execution core is assigned with one key for verification. The key is composed of 9 bits of the key subspace, 31 bits generated by the counter in the control module, and 8 bits which are unique for each core. If no key candidate is found, then the counter is incremented and the FPGA verifies the set of another 256 keys in the next round. Below we provide detailed description of all three phases.

In the *Load phase* all execution cores are loaded with 32 bits of serial number and upper 16 bits of the key. These data are common to all cores. This phase would require 48 clock cycles, however, we reduced the phase to just 16 clock cycles (below).

Then, in the *Initialization phase*, the cores are via input bit *data* loaded with product of xor operation of initialization vector and lower 32 bits of the key. The first 24 bits of this xor product are again common to all cores, but the last 8 bits are unique for each core. This phase requires 32 clock cycles.

Finally, in the last phase, the *Authenticator is generated and verified*. At the beginning of this phase the output bit *match* is set. Then the authenticator is bit-by-bit generated in each core. At the same moment the control module sends corresponding bits of sniffed authenticator to the cores via their input bit *data*. Each core on-the-fly compares bits of sniffed and generated authenticator. If the two bits are not equal, output bit *match* is cleared. This phase may require up to 32 clock cycles.

If the signal *match* is still set in some core by the end of the last phase, then the key candidate is found. Such key candidate is then send to the *H2 Core – final*, which repeats the same above three phases, but with another data.

5) Improvements: By default each round requires 112 clock cycles. By detailed examination of Hitag2 cipher operation and internal processes we have implemented two special features increasing the performance of the breaker. They are (i) parallel 3-wire LFSR load and (ii) round truncate function.

Parallel 3-wire LFSR load: As mentioned earlier, to allow synthesis tool to configure LUTs as shift registers (which enables effective utilization of FPGA chip), the LFSR has to be loaded via serial input. However, loading then requires 48 clock cycles. In order to achieve better design performance, we have decided to add 2 more load wires to Hitag2 execution cores (in Figure 4 denoted as *data2*). LFSR is divided into 3 parts, each being 16 bits long. These parts are loaded separately in only 16 clock cycles. Implementation of 3 wire LFSR load significantly reduces amount of clock cycles required to load LFSR. We save 32 clock cycles per round, which is 28% of the total number of clock cycles.

Round truncate function: During the Authenticator generation and verification phase, every execution core on-the-fly verifies conformity of the generated and the sniffed authenticator. In case when generated and sniffed authenticator discontinue to conform to each other, this non-conformity is signalized to the control module via the signal *match*. When all execution cores signalize authenticator non-conformity to the control module, the round is interrupted. The counter is incremented, new data are generated and new round is started. Implementation of this feature saves about 20 clock cycles per one round on average, which is about 18% of the total number of clock cycles in average.

Implementation of both these features saves about 52 clock cycles out of 112, which represents about 46% on average.

V. Implementation Results

Our proposed parallel brute-force breaker for COPACOBANA platform implements 256+1 Hitag2 execution cores in each FPGA chip. The design utilizes 90% of the hardware resources available on one FPGA chip and can run at a maximum frequency of 90 MHz. In each round of the key-search, 256 keys are verified in one FPGA. One round requires 60 clock cycles on average plus one clock cycle for initialization of the round. Every FPGA is thus able to verify about 378 million keys per second.

As a result, one COPACOBANA with 120 FPGAs is able to verify all 2^{48} keys in just 103.5 minutes. For comparison, the previously proposed attack methods are listed in Table I. Data for software implementation of brute-force attack are adopted from [5]. Our implemented design outperforms all known attacks by several orders of magnitude. Moreover, our proposed design has very low requirements on the amount of sniffed data — only 2 sniffed transactions are sufficient to reveal the secret key. Other attack implementations require data from at least four sniffed transactions.

TABLE I
Comparison of attack methods

	Type of attack	Implementation platform	Attack time
HAR 2009	Algebraic	PC	N/A
ISC 2009	Algebraic	PC	45 hours
SW implementation of a brute-force attack	Brute-force	PC	4 years
Our implementation	Brute-force	COPACOBANA	103.5 mins

Another advantage is the almost linear design scalability. It is straightforward to use more COPACOBANA machines in order to reduce the attack time. For example, when using 4 COPACOBANA machines, the time required to verify all keys in the key space would be less then half an hour.

We have practically verified the entire design and all its modules with large set of testbenches and test data sets to verify its proper function. The design fulfills all requirements and passed all tests.

VI. Conclusions and Final Remarks

In this work we have introduced a highly efficient implementation of a parallel exhaustive key-search of the Hitag2 cipher. Our attack is practically realized on the cryptanalytic hardware platform COPACOBANA. Each FPGA in COPACOBANA verifies about 378 million keys per second, therefore, one fully equipped COPACOBANA with 120 FPGAs is able to determine the correct key in less than 2 hours (103.5 minutes) in the worst case. The proposed design is almost linearly scalable, which allows further reduction of the attack time by employing more COPACOBANA machines.

The brute-force attack outperforms all previous implementations by several orders of magnitude. Just two monitored communications between a Hitag2 transponder and a reader, instead of 4 sniffed transactions required in other published attacks, are sufficient to reveal the secret key.

The attack also demonstrates the power of reconfigurable devices. Although the brute-force attack is in general the most demanding type of attack, its implementation in hardware is much faster then software implementation of less complex algebraic attack.

Further improvement may be gained by implementing the algebraic attack in hardware. However, this type of attack requires SAT solver, which represents serious design limitation — to the best of our knowledge, there is sill no existing implementation of SAT solver in hardware.

Acknowledgment

This research has been partially supported by MSMT under research program MSM6840770014, by the grant of the Czech Grant Agency GA102/09/1668, and by the grant of the Czech Technical University SGS10/119/OHK3/1T/18.

References

[1] Nicolas T. Courtois and Sean O'Neil, HITAG 2 Stream Cipher – C Implementation and Graphical Description, 2006-2007. http://cryptolib.com/ciphers/hitag2/, as of February 28, 2012.

[2] Nicolas T. Courtois and Sean O'Neil, FSE Rump Session – HITAG2 Cipher, 2008. http://fse2008rump.cr.yp.to/00564f75b2f39604dc204d838da01e7a.pdf, as of February 28, 2012.

[3] I.C. Wiener, Crypto1 specification, reference implementation and test vectors, 2007-2008. http://cryptolib.com/ciphers/crypto1/, as of February 28, 2012.

[4] Nicolas T. Courtois, Karsten Nohl and Sean O'Neil, Algebraic Attacks on the Crypto-1 Stream Cipher in MiFare Classic and Oyster Cards, Cryptology ePrint Archive, Report 2008/166, 2008. http://eprint.iacr.org/2008/166, as of February 28, 2012.

[5] Nicolas T. Courtois, Sean O'Neil, and Jean-Jacques Quisquater, Practical Algebraic Attacks on the Hitag2 Stream Cipher, In: *ISC '09: Proceedings of the 12th International Conference on Information Security*, pp. 167–176, LNCS 5735, Pisa, Italy, Springer-Verlag 2009.

[6] Henryk Plötz and Karsten Nohl, Breaking Hitag2, HAR2009, 2009. https://har2009.org/program/events/135.en.html, as of February 28, 2012.

[7] Philips Semiconductors, Hitag2 protocol datasheet, 1996. http://www.keeloq.boom.ru/HT2protocol.pdf, as of December 20, 2010.

[8] Philips Semiconductors, HT2 DC20 S20, HITAG^{TM}2 Transponders (datasheet), 1998. http:/www.synometrix.com/Hitag_2_Data_Sheet.pdf, as of December 20, 2010.

[9] Sandeep Kumar, Christof Paar, Jan Pelzl, Gerd Pfeiffer and Manfred Schimmler, *Breaking Ciphers with COPACOBANA - A Cost-Optimized Parallel Code Breaker.*, In: *Cryptographic Hardware and Embedded Systems — CHES 2006*, pp.101–118, LNCS 4249, Springer-Verlag 2006.

[10] Tim Güneusu, Timo Kasper, Martin Novotný, Christof Paar and Andy Rupp, *Cryptanalysis with COPACOBANA.*, In: *IEEE Transactions on Computers*, 2008, vol. 57, no. 11, pp.1498–1513.

[11] SciEngines, RIVYERA. http://www.sciengines.com/products/computers-and-clusters/rivyera-s3-5000.html, as of February 28, 2012.

[12] Car transponders table. http://www.keeloq.boom.ru/table.pdf, as of December 20, 2010.

[13] NKAAY, HITAG-2 Key Tool V50. http://www.nkaay.com/manual/hitag2.pdf, as of February 28, 2012.

[14] Henryk Plötz, Analyzing an unknown access control system, 2007. http://www2.informatik.hu-berlin.de/~ploetz/analyzing-an-unknown-access-control-system.pdf, aso of February 28, 2012.