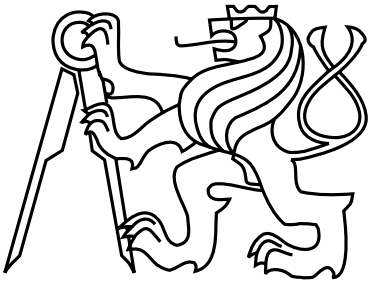
Czech Technical University in Prague
Faculty of Electrical Engineering
Department of Computer Science and Engineering



Master's Thesis

Cryptanalysis of Hitag-2 Cipher

Bc. Petr Štembera

Supervisor: Dr.-Ing. Martin Novotný

Study Programme: Electrical Engineering and Information Technology

Field of Study: Computer Science and Engineering

May 13, 2011

Acknowledgments

I would like to thank to Dr.-Ing. Martin Novotný for his valuable suggestions and to my family for their support.

Declaration

I hereby declare that I have completed this thesis independently and that I have listed all the literature and publications used. I have no objection to usage of this work in compliance with the act §60 Zákon č. 121/2000Sb. (copyright law), and with the rights connected with the copyright act including the changes in the act.

V Praze dne 13.05.2011 .....

Abstract

The Hitag2 stream cipher is used in many realworld 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 hour (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.

Abstrakt

Šifra Hitag2 je používána v mnoha různých aplikacích, jako například v imobilizérech a dálkových ovládání u automobilů, stejně tak ve vstupních systémech v budovách. Z důvodu že v šifře je použit pouze 48 bitový klíč, je napadnutelná pomocí útoku hrubou silou, tj. systematickým hledáním klíče. V rámci této práce jsme vyvinuli první hardwarovou architekturu pro kryptoanalýzu šifry Hitag2 formou systematického hledání klíče. Naše implementace na zařízení COPACOBANA je schopná v nejhorším případě odhalit tajný klíč použitý v transpondéru s šifrou Hitag2 za méně než 2 hodiny (103,5 minut). Rychlost útoku překonává všechny dříve publikované útoky a zároveň vyžaduje pouze data ze 2 zachycených komunikací mezi autem a transponderem. Naše zjištění tak definují novou nejnižší hranici pro klonování klíčů od auta v praxi. Útok je navíc velmi dobře paralelizovatelný, a tedy může být spuštěn na více zařízeních COPACOBANA pro zkrácení času potřebného k nalezení klíče.

Contents

List of Figures

List of Tables

1 Introduction

This diploma thesis deals with security of Hitag2 stream cipher. A Hitag2 cipher is commonly used in lightweight cryptography systems, RFID (contact less identification) and remote keyless systems. The RFID and remote keyless systems are widely used in many areas such as animals and goods identification, e-tolling, building access, etc. Remote keyless systems are used for remote permission or access denial to premises or cars where the usage of mechanical keys is replaced.

The goal of this thesis is exploration of resistivity against attacks, namely the brute-force attack. The thesis focuses mainly on hardware implementation of brute-force attack and uses FPGA as a target platform.

The Hitag2 is a synchronous stream cipher. The stream cipher system is based on a linear feedback shift register (LFSR) and nonlinear combining functions. The detailed scheme of Hitag2 cipher is in Figure 2.3.

The Hitag2 cipher was developed in late 90's by Philips Semiconductors (now NXP) as a successor of Crypto1 cipher [3]. The Hitag2 cipher is designed for usage in contactless identification and secure memory access. The cipher provides mutual authentication between transponder and read/write device (RWD). The Hitag2 cipher is used in building access applications and it is also widely used in car locks and immobilizers. According to [5, 6, 11, 12] cipher is used in access systems for army and government buildings in Germany and widely spread in car security systems. The Hitag2 cipher is extensively used in car models of many car manufacturers such as BMW, AUDI, Alfa Romeo, Porsche, Bentley, VW, Peugeot, Renault, Citroen, Iveco trucks and other. It is not clear whether the Hitag2 cipher is still used in new cars, however it is used in numerous models in a widespread use.

The rest of the diploma thesis is organized as follows. Chapter 2 discuss the related work, Hitag2 algorithm and its implmentation and COPACOBANA as the implementation platform are introduced. In Chapter 3 we present possible design approaches and analyse the attack methods and their feasibility. Chapter 4 examines implementation of our attack in the target platform and implementation of software for a PC. In Chapter 5 we describe design verification and validation process. Chapter 6 deals with the implementation results obtained. In Chapter 7 the future work is discussed and Chapter 8 concludes the work.

2 Background

In the first part of this chapter, we describe Hitag2 transponders, its operation modes and resistivity against attacks in these modes. In the following part, we describe the Hitag2 cipher and its structure. In the next part of this chapter, we introduce previous achievements and related work on the Hitag2 cipher. Finally, we introduce implementation platform and its properties.

2.1 Hitag2 Transponders

Hitag2 transponders are made as ASIC modules. They can be operated in 3 standard read only modes (TTF Public modes A,B,C) and in secured Password mode and Crypto mode. Transponders use base transmit frequency 125 kHz with Biphase or Manchester modulation. The average bit rate for read/write device is 5.2 kbits and up to 8 kbits for the transponder. Data are transmitted bidirectionally in half duplex mode. Operation modes and modulation type are configured by configuration byte in transponder's memory.

Hitag2 transponders provide 256 bits of non-volatile (EEPROM) memory organized in 8 pages – 32 bits each. Pages 0 through 3 contain transponder unique serial number, key, tag password and configuration data. Pages 4 through 7 are reserved for end-user data. For detailed overview of the Hitag2 transponder memory organization see Table 2.1. The transponder memory works like a FIFO (First-In-First-Out) memory. Hence the order of the transmitted bits differ in read and write operation.

2.1.1 Operation Modes and Configuration

According to Hitag2 transponder datasheet [8] and Hitag2 communication protocol datasheet [7], Hitag2 transponders can be used in these modes:

- TTF Public mode A, B, C - These modes are read only, 64 or 128 bits of user data is cyclically transmitted with no security applied
- Password mode - Mode for writing or reading the transponder with plain data transmission, password check is performed on transmission start
- Crypto mode - Mode for writing or reading the transponder with encrypted data transmission

Table 2.1: Hitag2 memory organization

Page	Password mode	Crypto mode
0	Serial Number	Serial Number
1	Password RWD	32 bit "KEY LOW"
2	reserved	16 bit "KEY HIGH", 16 bit reserved
3	8 bit Configuration, 24 bit Password TAG	8 bit Configuration, 24 Bit Password TAG
4-7	read/write pages	read/write pages

Operation modes can be configured using the configuration byte in transponders memory. The Hitag2 transponder tries to perform AUTH operation with read/write device for selected mode. If read/write device does not reply in specified time frame, the transponder automatically starts transmission in one of selected TTF public modes.

TTF Public modes A,B, C are suitable for applications such as animal identification. Data stored in transponders memory are periodically broadcasted in plain text, therefore these modes offer no security. Public modes differ only in type of modulation and amount of transmitted data.

Password mode is mostly used in buildings' access systems. In Password mode mutual authentication between transponder and read/write device is applied. Two different passwords are used in this mode. The password of the transponder (TAG Password) is 24 bit long, password of the read/write device (RWD Password) length is 32 bits. The TAG Password and RWD Password are both kept in both devices and must be the same.

Authentication process is performed by exchanging stored passwords for transponder and read/write device. The default transponders setting uses only the RWD Password to authenticate. Verification of TAG Password is optional and could be turned on by altering corresponding bit in configuration byte. However, verification of the TAG password is not required for accessing transponder memory.

Authentication process in Password mode consists of these steps:

- The tag sends: "11111" + SN
- The read/write device sends its stored password (RWD password)
- If the RWD password corresponds with password in tag memory, then tag sends "11111" + (configuration + TAG password)

For detailed graphical overview of the authentication process in Password mode, see Figure 2.1.

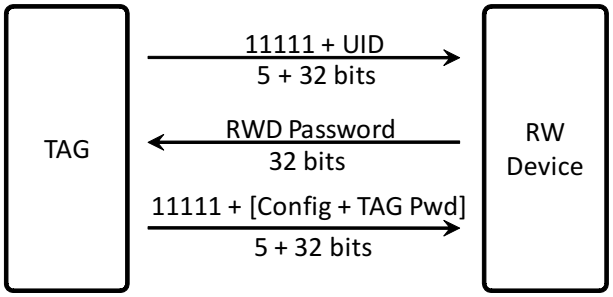


Figure 2.1: Password mode authentication process

The TAG Password and RWD Password do not change in time and are transmitted in plain text with no encryption applied. Hence communication between devices could be sniffed and both passwords could be very easily recovered. According to [6] only simple recording device, as some MP3 player, is needed to perform replay attack.

The Password mode offers only a very poor security level, even if verification of TAG password is used.

Crypto mode is used in applications where higher security requirements are applied. The most common fields of application is in car industry (car remote controls and immobilizers). It is the only mode with some sort of advanced security applied. The Hitag2 cipher (described in section 2.2) is used for encrypting data transmissions in this mode. According to Philips/NXP protocol specifications [7], Hitag2 transponders offer challenge-response authentication protocol. A detailed overview of mutual authentication between the transponder and the read/write device in Crypto mode is depicted in Figure 2.2. It works as follows:

- The tag sends: "11111" + SN
- The read/write device generates and sends a random IV (32 bits) and an authenticator (32 bits)
- If the authenticator is correct, tag sends "11111" + encrypted configuration and TAG Password
- All following transactions are encrypted

The initialization vector (IV) is randomly generated for every transaction between the reader and the transponder to prevent replay attacks. Mutual authentication is performed between transponder and read/write device. It is the read/write device, which proves its identity first by sending the authenticator. This prevents chosen-IV attacks.

Transmission of the configuration byte and TAG Password is encrypted. All the following communication takes place enciphered too. Access to the transponder is possible only after mutual authentication and password checking routine.

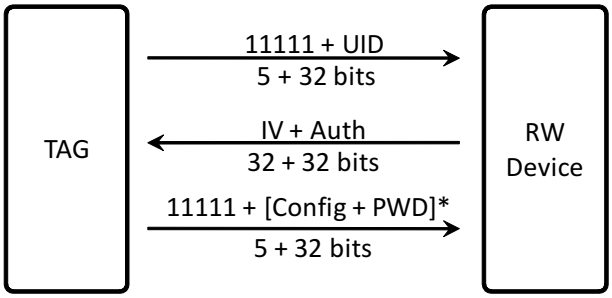


Figure 2.2: Crypto mode authentication process

According to Hitag2 protocol implementation, the attacker is able to recover transponders unique Serial number, generated IV and first 32 bits of Hitag2 keystream (Authenticator) in plain text from one transaction. Until the attacker does not know the content of Page 3 of the transponder memory, he/she will obtain 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 transponder and the read/write device. Because of low carrier frequency (125 kHz) and low rate, it is very simple to sniff transmitted data by simple antenna circuit [6], [13].

2.2 Hitag2 Cipher

The Hitag2 is a synchronous stream cipher based on a linear feedback shift register (LFSR) and nonlinear combining functions. Hitag2 is by its internal structure very similar to its predecessor, i.e., the Crypto1 cipher [3, 4] used in Mifare-Classic cards. Internal structure of the Hitag2 cipher is shown in Figure 2.3 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 be also downloaded from [1]. Hitag2 operates with 48 bit secret key, shared between the read/write device and the transponder (tag), 32 bit serial number of the tag and 32 bit initialization vector IV.

Hitag2 operation

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 2.3). Then, in 32 steps the LSFR is filled from right with 32 bits, each of them being an 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.

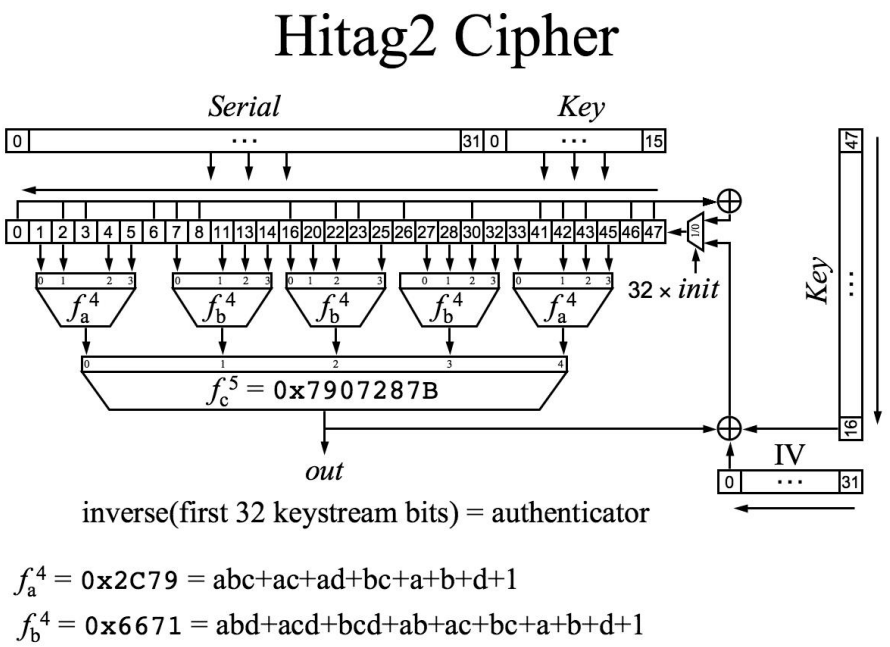


Figure 2.3: Hitag2 stream cipher. Adopted from [1]

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, f_b^4 and 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 $f_a^4(1111)$.

The Hitag2 cipher nonlinear combining function with 20 inputs would provide good cipher complexity. However, result of nonlinear function is filled into LFSR only during initialization phase. Lack of nonlinear input during Keystream generation phase decreases overall cipher complexity. This property has been exploited in algebraic attack.

2.3 Related Work

Several attacks on Hitag2 have been published in open literature. Since the cipher had been kept secret (security by obscurity) by the manufacturer, an important achievement was reveal of the detailed principle of the Hitag2 cipher [2]. The description of the Cipher together with a reference program code was 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. Afterward, the system of equations is transformed into a SAT problem and solved with a SAT solver on a PC.

2.3.1 ISC 2009

In ISC 2009 Nicolas T. Courtois, Sean O'Neil and Jean-Jacques Quisquater introduced "Practical Algebraic Attacks on the HITAG2 Stream Cipher" [5].

In the first part of their work they concentrate on description of the algebraic attack on the Hitag2 cipher. Transformation to SAT problem is demonstrated and principle of the attack is described. 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 implementation of more practical attack is described in the second part of their work. Practical attack needs data from at least 4 sniffed transactions between transponder and read/write device. Attack is performed by repeating these steps:

- Guess/fix 14 bits of key
- Write equations for four known IVs
- Solve with SAT solver

With these (random) data, the calculations require 45 hours on a PC.

2.3.2 HAR 2009

In HAR2009 Karsten Nohl and Henryk Plötz introduced their achievement on the Hitag2 cipher. They introduced algebraic attack method in proceedings "Breaking Hitag2" and "Hitag2 insecurity" [6]. In proceeding "Breaking Hitag2" they describe a Hitag family ciphers and transponders with focus on the Hitag2 cipher. In that work they show configuration modes and complete description of mutual authentication protocol for these modes. They also specialize in hardware side of crypto system and describe method of sniffing and decoding data transmitted between the transponder and read/write device.

In the second proceeding "Hitag2 Insecurity" they focus on describing weak points in this encryption system and describe approaches for performing attack on cipher with assumed

need for system resources. They concentrate on applying algebraic attack and describe cipher as system of equations. This description allows them transformation of this equation system into SAT problem. For applying algebraic attack they use advanced Crypto SAT [16] solver which is able to solve terms including XOR operation. They state that it is possible to compute secret key in less than 6 hours in standard PC.

There is a lack of detailed description in available material. Hence 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. Unfortunately, we were also unable to find out required amount of sniffed data used for performing stated attack. We assume that the stated attack time was only for the theoretical case of chosen-IV.

2.3.3 Summary

To the best of our knowledge only these two proceedings about Hitag2 cipher were published in open literature. Comparison of achieved results is summarized in Table 2.2.

Table 2.2: Summary of known attacks on Hitag2 cipher

Conference	Used method	Theoretical attack	Practical attack	Pre-requisities
ISC 2009	Algebraic attack	6 hours	2 days	4 transactions
HAR 2009	Algebraic attack	6 hours	NA	NA

The Hitag2 cipher can be successfully attacked by algebraic attack. In the 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. Hence it is not clear whether the attack time is valid only for theoretical case (chosen-IV attack) or whether it is practical (valid for any random data). We assume that the stated attack time was valid only for the theoretical case of chosen-IV.

2.4 Implementation Platform

COPACOBANA (Cost-Optimized Parallel Code Breaker) [9] machine is a high performance cluster of up to 120 FPGAs. COPACOBANA is reconfigurable and can be used in many applications. COPACOBANA was developed as a parallel machine for code breaking tasks.

Figure 2.4: COPACOBANA machine - front view. Adopted from [9]

Table 2.3: Summary of Spartan3 - XC3S1000 hardware resources

Hardware resource type	Amount
Input/Output Blocks (IOBs)	175
Configurable Logic Blocks (CLBs)	1,920
Slices	7,680
Flip-Flops (FFs)	15,360
Look-Up Tables (LUTs)	15,360
Digital Clock Managers (DCMs)	4
Dedicated Multipliers	24
Block RAM [Kbit]	432

Depending on used algorithm, it can outperform conventional computers by several orders of magnitude. Figure 2.4 shows a front view of COPACOBANA machine.

COPACOBANA is equipped with Xilinx Spartan3-XC3S1000. These FPGAs offer balanced ratio between its performance and price. For an overview of Spartan3-XCS1000 hardware resources, see Table 2.4. In this diploma thesis we use COPACOBANA equipped only with 60 FPGA cores. Due to almost fully linear scalability of designs implemented in COPACOBANA, we recount all measured times as it was achieved on fully equipped COPACOBANA with 120 FPGAs.

The FPGA chips are located on custom made DIMM modules. The COPACOBANA machine can host up to 20 DIMM modules with 6 FPGA chips on each module. The DIMM modules are connected on the backplane by a 64 bit data bus and a 12 bit address bus with an operating frequency of $f_{Bus} = 20\,MHz$. Figure 2.5 shows the COPACOBANA backplane fully housed with DIMM modules.

Address bus is organized as follows:

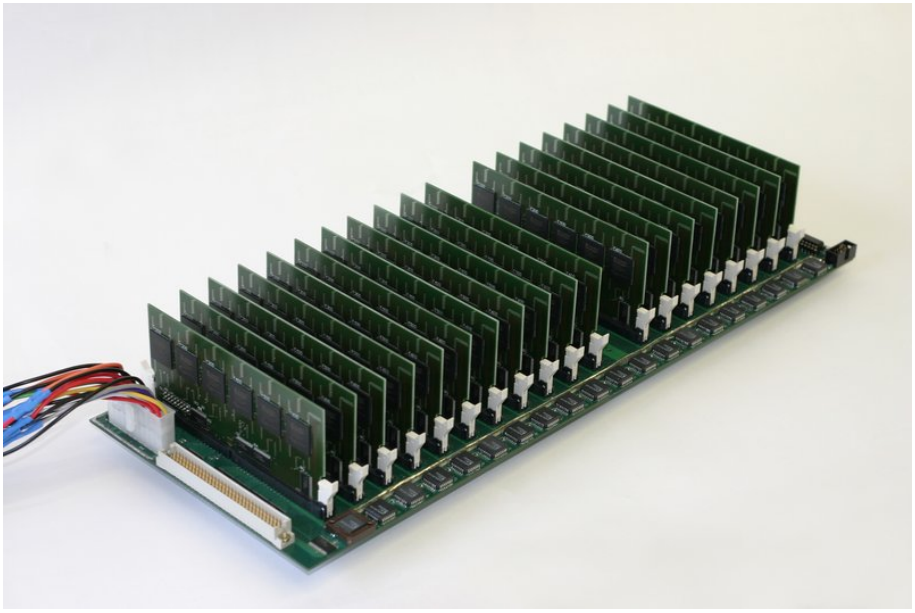


Figure 2.5: COPACOBANA backplane with DIMM modules. Adopted from [9]

- 5 bits encode module slot
- 6 bits encode FPGA chip on module (one-hot encoding)
- 1 bit is used to choose between two different 64 bit data registers on each FPGA

The whole communication with the host computer is provided by a controller card based on an FPGA with a Xilinx MicroBlaze softcore. It is connected via Ethernet and supports the connection via TCP/IP protocol.

To implement designs running at the frequency higher than $f_{Bus} = 20$ MHz, the internal clock signal is synthesized on each FPGA with the embedded DCM. The operating frequencies up to 300 MHz can be achieved on the FPGAs. Power consumption of fully housed COPACOBANA machine running on 100 MHz frequency is only about 600 Watts.

The COPACOBANA was designed under assumption that

- computationally costly operations are parallelizable

- parallel instances usually do not need to communicate with each other

- the demand for data transfers between host and nodes is low because of the fact that computations dominate over communication

- typical crypto algorithms and their corresponding hardware implementations do not require much memory

This leads to some limitations which have to be taken into account during design implementation. In general, these limitations do not affect typical code breaking applications but can significantly lower performance in applications where memory requirements are higher or where data exchange between FPGA chips is needed.

3 Analysis

In the first part of this chapter, we summarize Hitag2 resistivity against attacks of Hitag2 transponders operating in crypto mode. In the next part, we discuss potential implementations of attack methods and describe possible design approaches. Finally, we summarize analysis results and select the best candidate for implementation.

3.1 Hitag2 Resistivity Against Attacks

According to Philips/NXP specifications [7], Hitag2 transponders use base transmit frequency 125 kHz. Since the transmit frequency used in Hitag2 transponders is low, sniffing device could be build very easily. According to [6], only simple diode detector and soundcard for A/D conversion are needed.

Hitag2 implements only 48 bit secret key. Hence, the attack methods with high complexity, such as brute-force attack, are feasible for this cipher.

The 48 bit secret key, together with the initialization vector and the serial number of the transponder are used for initialization of the cipher. The initialization vector is randomly generated for every transaction between the reader and the transponder to prevent replay attacks. However, until the cipher is initialized, transmissions are performed with unencrypted data. Therefore, the attacker could easily sniff transponder SN, generated IV and corresponding authenticator. These data are sufficient to perform successful attack on crypto system.

As the secret key has 48 bits and sniffed authenticator is only 32 bits, the attacker needs data from at least two sniffed transactions between the transponder and the read/write device to perform successful recovery of this key.

The Hitag2 cipher nonlinear function contains 20 inputs. However, result of nonlinear function is filled into LFSR only during the initialization phase. Lack of nonlinear input during Keystream generation phase decreases overall cipher complexity. This property has been exploited in algebraic attack. The encryption system could be described as a system of equations, transformed into SAT problem and resolved by conventional SAT solver.

3.2 Attack Methods

In this section, we deals with the attack methods applicable to the Hitag2 cipher and select the optimal one for implementation on target platform. At the present time, there exist several attack methods which can be applied to encryption system. To the best of our knowledge

there are 4 possible attack methods applicable to the Hitag2 cipher—algebraic attack, brute-force attack, pre-computation and LFSR content backtracking.

Other attack methods as side channel attacks (e.g. time analysis, power analysis, etc.), are not suitable for Hitag2 encryption system. These attack methods are based on information aquired from physical implementation of the cryptosystem and require direct monitoring of the physical cipher device. In case when Hitag2 transponder device is kept by an attacker all secret data could be retrieved by a Hitag2 device reader/writer [12], hence no further analysis is required and performing these methods would be unnecessarily.

3.2.1 Algebraic Attack

The algebraic attack method was used in all previous attacks on cipher [5] and [6]. The algebraic attack exploits cipher's weakness in its low complexity which allows transferring cipher system into SAT problem and solving it with SAT solvers. A practical attack performed with this method requires only less than 2 days [5]. The algebraic attack implemented on special hardware could eventually outperform all other methods and/or implementation platforms.

Disadvantage of this method is complicated hardware implementation of universal and effective SAT solver in FPGA. Compared with modern software solvers, the hardware SAT accelerators are usually slow and capacity limited, and they are unable to accommodate some important features in modern software solvers such as learning. According to basic description of published FPGA based SAT solver accelerators i.e. [14] and [15], these implementations require reconfiguration for each set of clauses and/or require transferring large amount of data between FPGA and control application on a personal computer. Unfortunately, COPACOBANA does not allow exchanging data between nodes directly and also used communication interface is unable to transfer big amount of data between machine and host computer in short time. Due to some technical complications, Ethernet interface in COPACOBANA could achieve maximum throughput of only 300 kbit per second.

To the best of our knowledge no VHDL or Verilog source code of FPGA based SAT solver was published, neither the source code for [14] and [15]. Therefore if we implemented this method, we also would have to implement SAT solver for FPGA platform. SAT solver implementation on FPGA platform is difficult and beyond the scope of this diploma thesis.

Algebraic attack implemented without effective parallel SAT solver wouldn't be able to utilize the potential of target platform and thus is unsuitable for implementation in COPACOBANA.

3.2.2 Brute-force Attack

The brute-force attack method (or exhaustive key search strategy) can be theoretically used for attack on any encryption system. This method systematically verifies all possible keys until correct key is found. The key length used in encryption system determines feasibility of performing the brute-force attack because resources required for brute-force attack scale exponentially with increasing key size.

The brute-force attack is often used when no weakness in encryption system is found or cannot be exploited. Practically it is the attack method with highest complexity and other methods may outperform the brute-force attack. Time necessary to perform brute-force attack is often used for measuring encryption system safety.

Considerable advantage of this method is its simple parallelization—search space can be divided into independent subspaces, which could be processed separately. This property can be with advantage used in FPGA platform such as COPACOBANA.

Hitag2 cipher uses 48 bit key. According to previous experience with COPACOBANA, namely the DES implementation [9], and also with respect to complexity of Hitag2 cipher, we estimate that implementation of brute-force attack on Hitag2 would achieve at least 2^{26} key verifications per second in one FPGA chip. Traversing entire key space would then take about 1200 hours (50 days) in single FPGA chip. COPACOBANA can be equipped with up to 120 FPGA cores. In that configuration it would be able to perform complete brute-force attack in about 10 hours.

3.2.3 Precomputation

Precomputation is a method implemented by computing combinations of input and output values. There exists many possible approaches e.g. TMTO or Rainbow table. These computed values create tables, where output values are assigned to corresponding input values and keys. Main advantage of this method is simple attack repeatability and possibility to decrypt data quickly. Computations are done only once and can be used in all future attacks on cryptographic system. Main disadvantages of this method are its big demand for huge memory resources and potentially very long time necessary for computation of entire table.

According to Karsten Nohl and Henryk Plötz [6] precomputation would require 2^{49} key computations.

We assume that TMTO tables would require about 48 GB of memory space. However, to create the table we would require 48 bit of keystream. As long as we do not know the content of the Hitag2 transponder memory, we will not be able to obtain more than 32 bit of keysteam from sniffed transactions. Hence implementation of this type would be only

for theoretical case. We suppose the precomputation method implementation for performing practical attack would be very complicated and beyond scope of this diploma thesis.

3.2.4 LFSR Content Backtracking

Hitag2 is a stream cipher based on a linear feedback shift register. LFSR is used in Hitag2 in clearly linear mode during cipher operation. Main concept of content backtracking method is based on computing LFSR content from known cipher output. LFSR operation could be mathematically described as multiplication modulo X, where X is a polynomial. The Hitag2 cipher uses 48 bit LFSR. According to LFSR mathematical description, it's clear that LFSR returns to its initial state after 2^{48}iterations. LFRS content (of any previous or future state) could be simply computed in case its content in current state is known.

The content backtracking attack works as follows:

Phase I — During this phase, LFSR is stepped forwards and a keystream is generated through non-linear functions. Generated keystream is compared to sniffed authenticator. In case the match of keystream and authenticator is found, LFSR content is stored and then processed in phase II. The stored content equals to Hitag2 post-initialization state.

Phase II — In this phase LFSR is stepped backwards, the Hitag2 cipher initialization is also reversed. During backward stepping, bits of sniffed serial number are shifted back into LFSR. According to value of initialization vector and output of non-linear functions, corresponding bit of key is computed. After 32 steps, we would have the complete key— 32 bits computed + 16 bits from LFSR actual content.

An advantage of this method is that Phase I operations are parallelizable and searchspace could be divided into independent parts and processed separately. Disadvantage could be complicated hardware implementation and high attack complexity. Complexity of this attack method equals to the brute-force attack. To traverse entire key space, we need 2^{48} LFSR iterations. Moreover, since we can use only 32 bits of keystream (sniffed authenticator), we would need to repeat entire process with another initialization vector and authenticator to verify the key.

3.2.5 Attack Methods Summary

We have analyzed all of above stated attack methods. In accordance with our analysis and scope of this diploma thesis, we have decided to implement brute-force attack. We assume that the brute-force attack implemented in COPACOBANA would effectively utilize target platform and would achieve attack time less than 10 hours. This would be significant reduction of attack time in comparison to other known attack methods.

3.3 Design Approach

We have explored several design options to implementd effective and high performance brute-force attack on Hitag2. Target platform properties and requirements of selected brute-froce attack determined the design options. To implment brute-force attack, we mainly discussed two design options—a pipeline model and model with parallel cipher cores. 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-registerbased 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 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.

3.3.1 Key Candidates Processing

As we mentioned above, Hitag2 key is 48 bits and data from sniffed transaction contains only 32 bits authenticator. According to that, the keyspace verified with one pair of initialization vector and authenticator would generate in average $2^{48} - 2^{32} = 2^{16}$ key candidates. These key candidates have to be verified with another couple of initialization vector and authenticator to select correct secret key. Verification of candidate keys could be implemented externally in an application on a PC or locally in FPGA chip.

External - The first approach verifies key candidates in an application on a PC. FPGA verifies keys in assigned subspace and if the key candidate is found, it is stored in FPGA memory. The application on a PC periodically reads the key candidates from all

FPGAs in COPACOBANA and then verifies them with another data set. To implement this concept, the memory module and memory controller have to be included in design.

Internal - Verification of key candidates is performed directly in FPGA chip. In this concept one Hitag2 execution core is added. This core is reserved for verification of key candidates. If the key candidate is found, it will be transferred into the reserved core and verified in next round. The reserved core verifies the candidate key with another set of initialization vector and authenticator.

We assume that both discussed concepts could offer similar performance. To implement concept with external key candidates verification, we would also need to implement software application to process key candidates. We assume the implementation of memory controller would be more complicated than adding one more execution core. Therefore we have decided to implement concept with internal candidate keys verification.

4 Implementation

In the first part of this chapter, we describe architecture of selected brute-force attack and explain used algorithm. In the second part, we focus on a hardware implementation and define implemented modules and its functions. Third part of this chapter deals with software implementation of control application for host computer.

4.1 Attack Architecture

We have implemented the brute-force attack on the Hitag2 cipher. The attack works as follows: From one transaction between the transponder and the read/write device we get the serial number, initialization vector, and authenticator. Then, we generate and test all 2^{48} keys. We load each key to the Hitag2 core together with the serial number and the initialization vector, and we generate the authenticator. We compare generated authenticator 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 checked against data from another sniffed transaction between the transponder and the read/write device. These data contain the same serial number of the tag, but the initialization vector and the authenticator are different.

To parallelize the attack, the search space is divided into key subspaces. Each FPGA is assigned by the host computer with one subspace to search in. If the search in the subspace is finished without any success, 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 4.1. The breaker consists of the control module and 256 Hitag2 execution 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 4.1) for verification against data from second transaction between the transponder and the read/write device.

4.2 Hardware Implementation (VHDL)

Hardware design is written in VHDL language and implemented with Xilinx ISE 9.2i. Hardware implementation is based on results of analysis and respects all limitations of implemen-

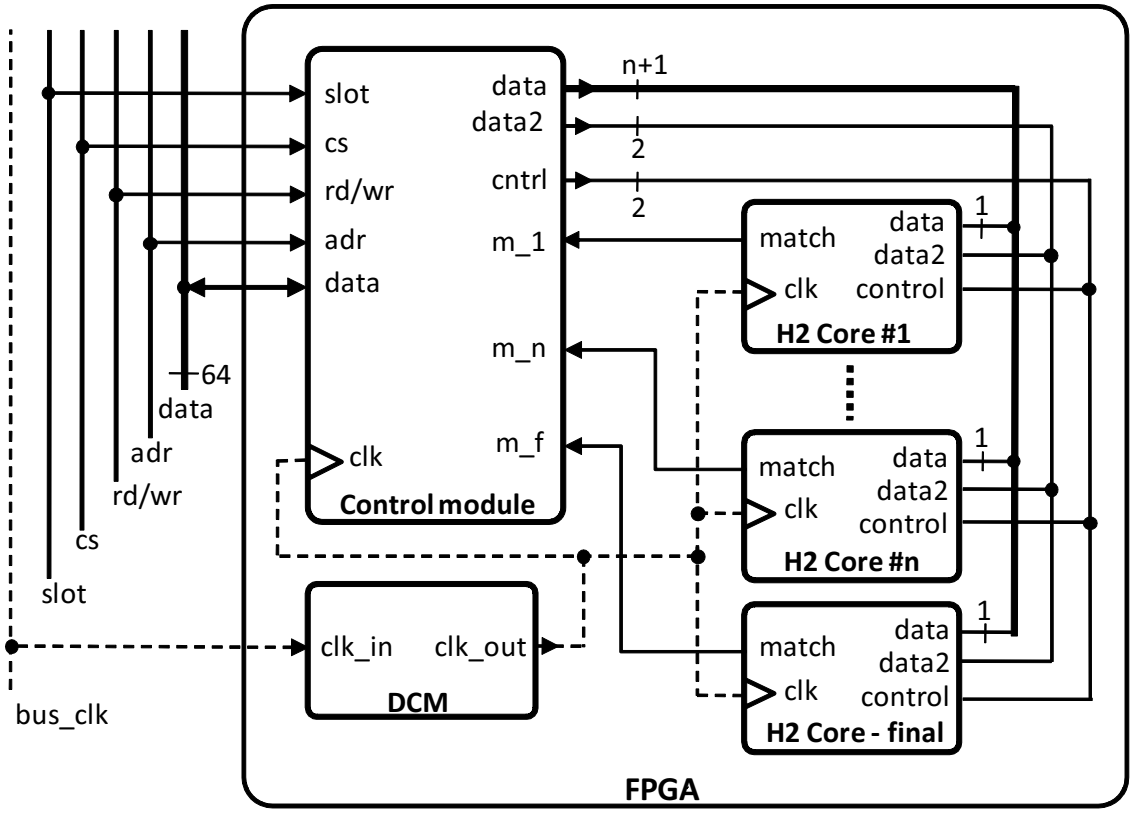


Figure 4.1: Top-level overview of the Hitag2 breaker in one FPGA

tation platform. Our hardware design is based on concept with one control unit and many parallel execution cores. One of execution cores is reserved for key candidates verification. The top-level overview of the Hitag2 breaker implemented in one FPGA is displayed in Figure 4.1. Source codes of our hardware design are available on the attached CD in folder SRC.

The design consists of 2 main components—Control module and Hitag2 execution cores denoted as H2 Core and H2 Core-final. DCM unit is responsible for clock synthesis in FPGA chip. The control module is responsible for controlling attack steps and for communication through COPACOBANA interface with host application on a PC. Hitag2 execution cores are used to verify generated keys.

4.2.1 Control module

Control module provides communication with host PC, process received data a controls key subspace processing in Hitag2 execution cores. Control module consists of COPACOBANA bus inteface, main controller unit and Hitag2 cores controller. Block-level overview of the Control module can be found in Figure 4.2.

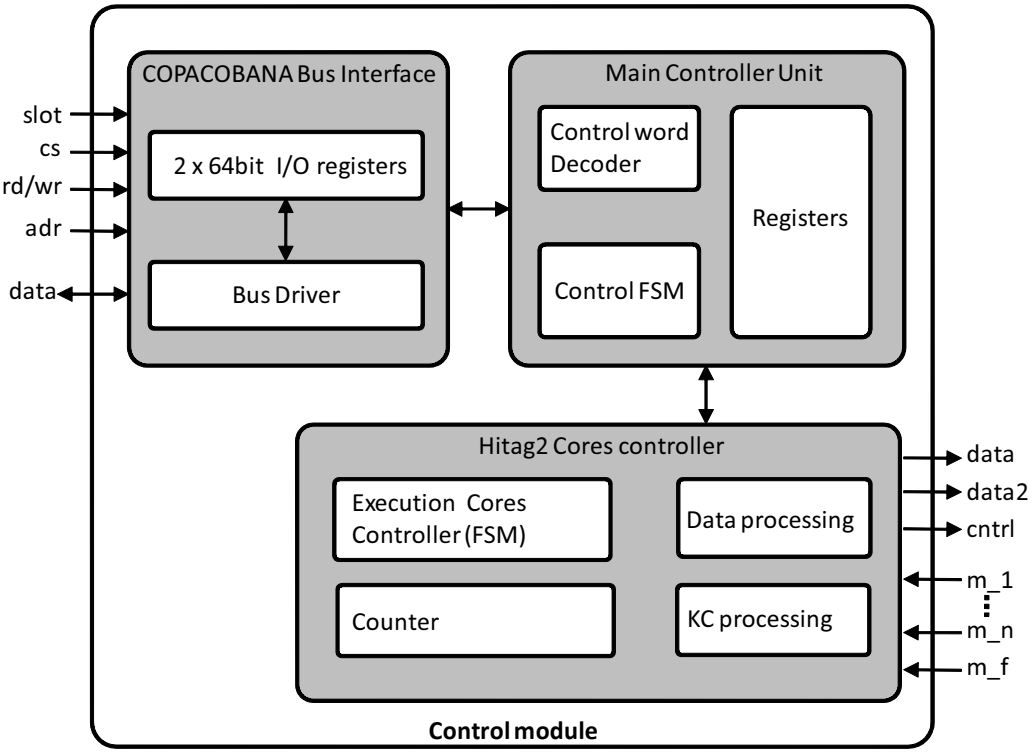


Figure 4.2: Overview of the Control module

COPACOBANA Bus interface provides communication between COPACOBANA bus and main controller unit. Since the COPACOBANA bus operates on frequency 20 MHz, while the implemented design in FPGA operates on 90 MHz, we have to implement some additional features to preserve reliable and secure communication. This module implementation is primarily based on bus interface module imeplementation, previously used in DES breaker implementation [9].

The interface module consists of two blocks - COPACOBANA bus driver and I/O registers. The bus driver interprets COPACOBANA bus address signals and generates signals for superset module (main controller unit). To store data, two 64 bits I/O registers are implemented.

Connection to COPACOBANA bus is implemented by 3-state registers. The interface module output and stores data to the addressed register when this FPGA is selected. Otherwise, set bus pins to high impedance.

The COPACOBANA bus interface module has to be implemented in pure logic (not sequentially) to ensure true single cycle read accesses of the bus controller.

Main controller unit primary objectives are supervising the subspace processing in the Hitag2 execution cores and communication with application on a PC. To communicate with application on a PC, simple communication protocol has been created. COPACOBANA

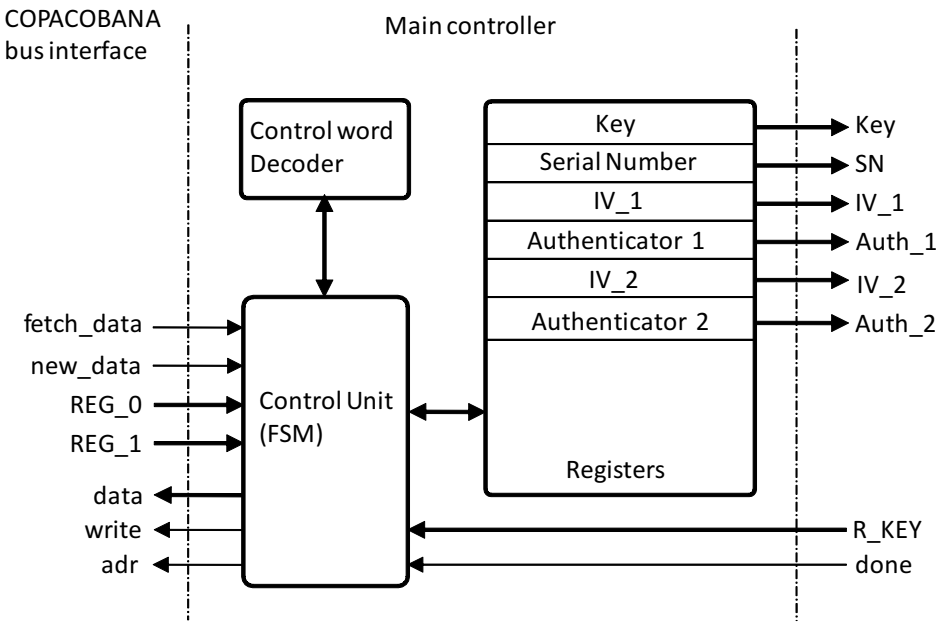


Figure 4.3: Main controller unit overview

bus has 64 bits data bus and 12 bits address bus. Address bus 11 bits used to address FPGA, 1 bit is used to select between I/O registers. Data received from an application on a PC are divided to header (top 16 bits) and data section (remaining 48 bits). Header contains commands for control module and description for included data. Overview of the Main controller unit could be found in Figure 4.3.

The controller unit operation is divided into three phases—Stand-by, Run and Wait. In the stand-by phase, the main controller unit collects all necessary data such as 9 most significant bits of key (subspace identifier), serial number and two pairs of initialization vectors and authenticators. These received data are stored in corresponding registers. Since the all required data are stored in controller, Run phase begins. Hitag2 Cores control module is initialized and subspace verification in Hitag2 execution cores is started. The controller monitors Hitag2 cores controller and generates status data. Then the status data are written into COPACOBANA bus interface. After finishing the subspace verification, the third phase (Wait) is initiated. The result of subspace verification is written into COPACOBANA bus interface and further instructions from application on a PC. User data are written into I/O register only when previous data has been read by host. Flowchart of main controller unit operation is depicted in Figure 4.4.

Hitag2 cores controller module supervises execution of the brute-force attack in Hitag2 execution cores. Its primary objectives are generating data and control signals for Hitag2 execution cores, evaluating results and generating status information for the main controller unit. The Hitag2 cores controller overview can be found in the bottom part of

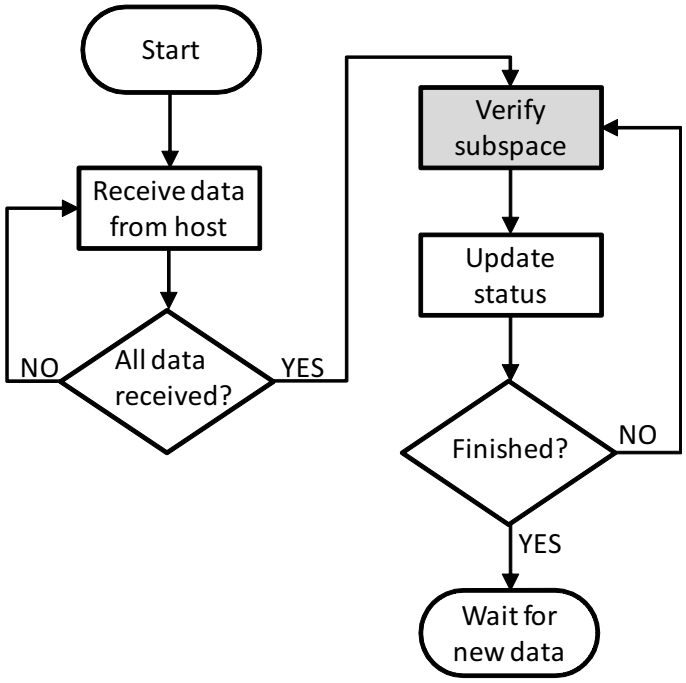


Figure 4.4: Opearation flowchart of the Main controller unit

Figure 4.5.

The Hitag2 cores controller module is controlled by Excecution cores Controller implemented by FSM. Its flowchart is depicted in Figure 4.6.

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, 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 loaded with product of xor operation of initialization vector and lower 32 bits of the key via input bit data. 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 two bits are not equal, output bit match is cleared. This phase may require

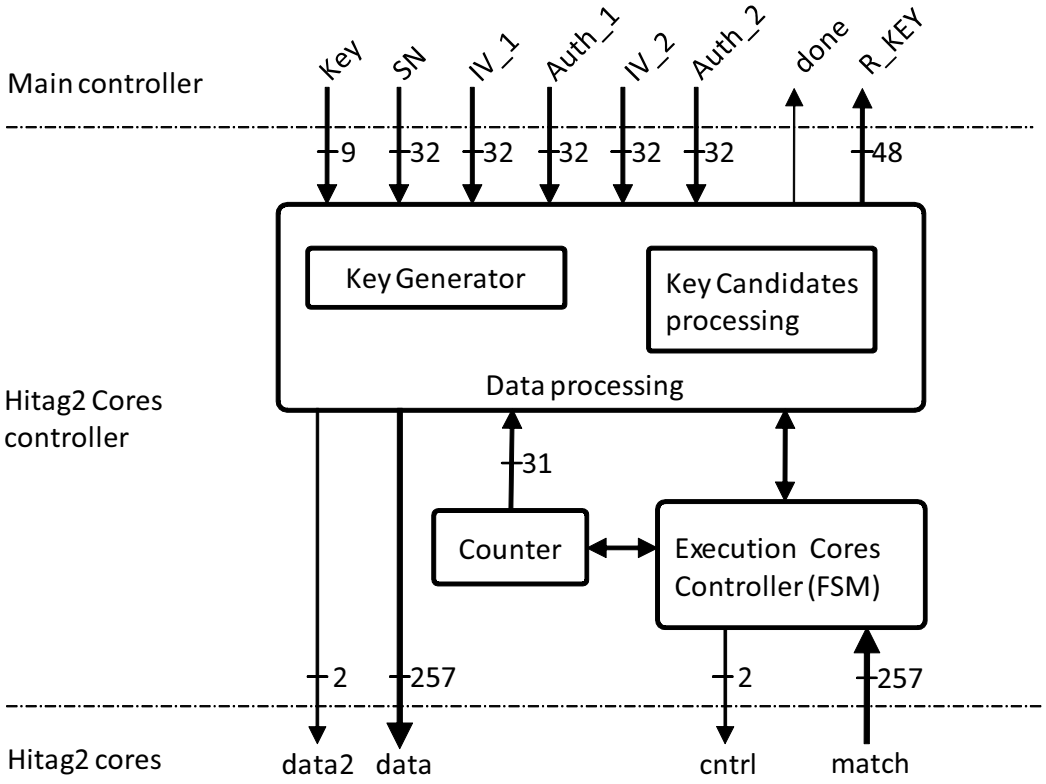


Figure 4.5: Hitag2 cores controller - overview

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. Then, such key candidate is send to the H2 Core – final, which repeats the same above three phases, but with another data.

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. These are (i) parallel 3-wire LFSR load and (ii) round truncate function.

Parallel 3-wire LFSR load: As mentioned above, 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, then the loading 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 4.5 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,

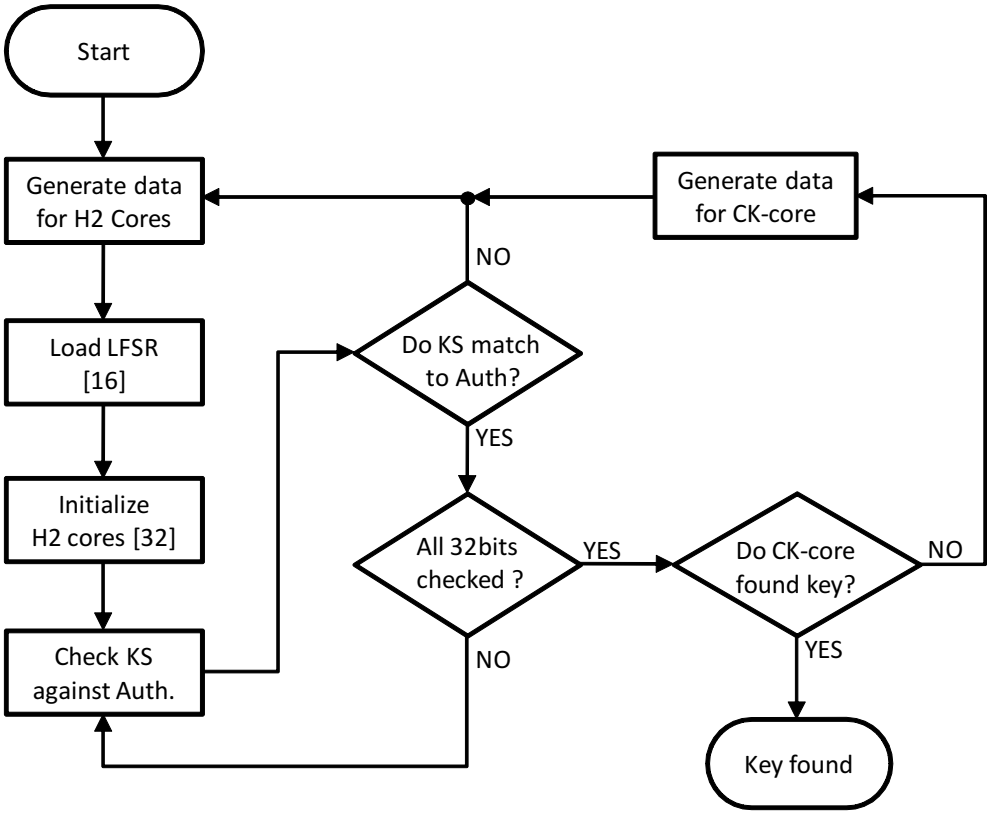


Figure 4.6: Flowchart of the FSM of a Excecution controller

every execution core on-thefly verifies conformity of the generated and the sniffed authenticator. In case when generated and sniffed authenticator discontinues conforming to each other, this non-conformity is signalized to the control module via 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.

4.2.2 Hitag2 Execution Core

The Hitag2 execution core simulates Hitag2 cipher operation. It is slightly modified to simplify brute-force attack implementation. The Hitag2 core consists of 48 bit LFSR, 3 nonlinear functions and few others components such as comparator, multiplexer and few XORs. The structure of the Hitag2 execution core is in Figure 4.7.

Execution core has only 5 input bits and 1 output bit. The input signals consist of 3 data

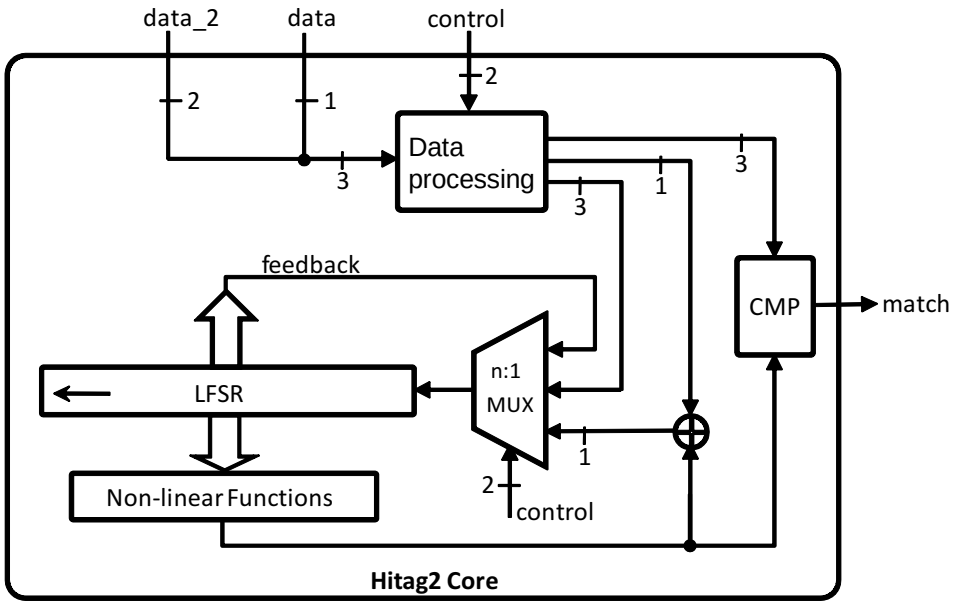


Figure 4.7: Overview of the Hitag2 execution core

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.

The execution unit implements these 3 phases:

I. LFSR load – LFSR register is loaded with initialization data (serial number and high 16 bits of key). The LFSR load phase requires 16 clock cycles.

II. Initialization – The Hitag2 cipher's initialization is rerformed. LFSR is filled by result of XOR function of incoming data(low 32 bits of key) and result of nonlinear function. Initialization phase requires 32 clock cycles.

III. Authenticator generation and verification – The keystream is generated and compared to sniffed authenticator. LFSR is filled by feedback data, *match* signal is generated as a result of comparing the generated keystream and incoming data(sniffed authenticator). Phase length is variable from 1 up to 32 clock cycles.

The output signal - MATCH is initially set to high during phase II - Initialization. If the

sniffed authenticator value does not equal to result of nonlinear function (authenticator for current key), MATCH signal is set to low state. When MATCH signal is in low state, computation can be interrupted and restarted with new data (key).

Improvements: As mentioned before, we have exploited the FPGA platform in our design by using LUT configured as SRL16. Therefore, we were able to reduce demand of hardware resources per one execution core by more than 50 %. This enhancement reduced amount of flip-flop registers per one LFSR from 48 to only 24. However, this optimalization brings some design limitations into circuit design. To make it possible for synthesis tool to configure LUT as SRL16, we have to avoid parallel input or output to the shift register, using only serial input and output. Unfortunately, to load LFSR, 48 clock cycles are required. LFSR is divided into 3 parts (16 bit each), to reduce clock cycles count required to load. These parts are jointly loaded with data (data2 inputs are used). Then the LFSR is fully loaded in only 16 clock cycles, while the demand for the execution core data inputs increases only to 3. Moreover, the data2 inputs are common for all Hitag2 execution cores. This feature saves 32 clock cycles, which represents 28 % of total clock cycles.

4.3 Parameter selection

In order to achieve best possible results with our hardware configuration, we have to set various design parameters. The main design parameters, which we have to decide, were - size of assigned subspace for one FPGA and communication parameters. The communication parameters include for example wait time between reads from COPACOBANA, read/write operation latency and redundancy of these operations. We have set them experimentally in order to achieve reliable communication between COPACOBANA and host PC and also with respecting all COPACOBANA platform properties.

Subspace size parameter strongly depends on COPACOBANA configuration especially on a number of installed FPGA cores. In our case, COPACOBANA has 60 FPGAs installed. We have measured attack time for selected subspace sizes and calculated total attack time. Summarized results could be seen in Table 4.1.

Based on the achieved results, we have decided to divide search space into 512 subspaces.

4.4 Software Application (Java)

Software part of implementation is based on Java platform. The Java platform was selected for its portability. Implemented application allows programming and communication with COPACOBANA machine via TCP/IP interface. Application is designed in order to provide control center for performing the brute-force attack on the Hitag2 cipher.

Table 4.1: Subspace size and attack time for COPACOBANA with 60 FPGA

Subspace size [bits] / Number of subspaces	Number of repeatings	Subspace verification time [min]	Total attack time* [min]
40 / 256	5 (4.27)	46.4	234
39 / 512	9 (8.54)	23.1	212
38 / 1024	18 (17.1)	11.5	217
37 / 2048	35 (34.1)	5.7	218

* Total attack time includes time necessary to change subspaces

Our application is based on application core, which we have used from example memory test application created by COPACOBANA manufacturers [9]. Our implementation exploits its basic functions for programming FPGAs, data transfer functions and GUI structure. To meet all our requirements on the application, we have implemented several new functions. Source codes of our application are available on the attached CD in folder SRC.

4.4.1 Application (GUI)

The application GUI offers all necessary input and control elements. Snapshot of application main screen is depicted in Figure 4.8. Top part of application main window is dedicated for configuration fields and input fields for an attack implementation.

The configuration part is placed on the right side of the top panel. Text fields for setting COPACOBANA IP address and port are placed there. Additional fields are used to define COPACOBANA current hardware configuration, such as slot range and range of used FPGAs per slot. On the right side of the top panel, Hitag2 configuration entry fields are placed. Data from sniffed Hitag2 transactions have to be entered there. Namely tag serial number and two pairs of random generated initialization vectors and its respective authenticators. Values have to be entered in hexadecimal form.

Checkbox to enable FPGA programming, text field with path to Hitag2 RBT file together with the browse button and the resume function controls are placed in the bottom part.

In the central part of main window, array of buttons representing FPGA cores in COPACOBANA is situated. Each button in array displays actual FPGA state. Window with detailed communication log and debug controls for corresponding FPGA is displayed on button press. These controls have to be used for debug purposes only. Any inappropriate interventions into ongoing communication could cause severe communication error and eventually result in attack failure.

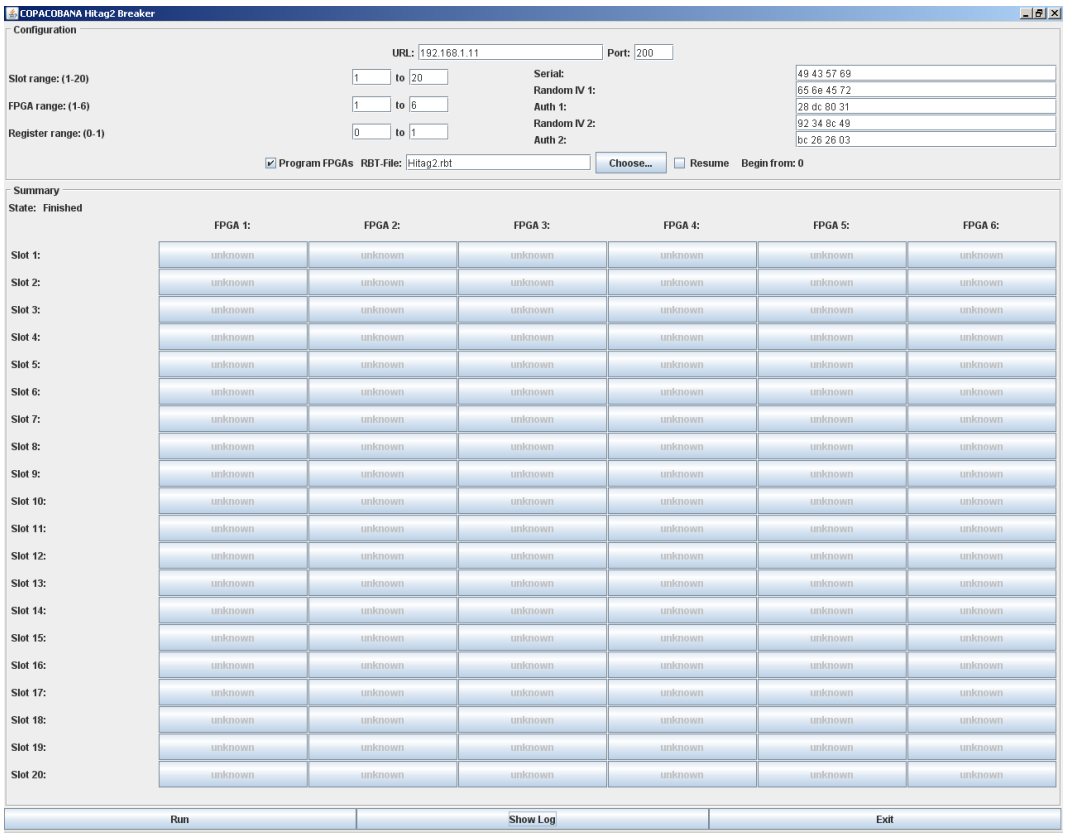


Figure 4.8: Application GUI - Main window

In the bottom part of main window, three control buttons are placed. These buttons—Run, Show log and Exit, are used to control main actions of Hitag2 attack. The Run button start/stops attack, the Show log button turn on/off window with application log and finally Exit button terminates application.

4.4.2 Special Features

There are several new features implemented in the application. Such as failure detection, recovery and resume function. All these function allow easier control and reliable communication with the FPGAs in the COPACOBANA.

Failure detection & recovery feature allows detecting FPGA failure and try to recover it back to functional state. This feature consists of Monitor and Recovery functions. The Monitor function continuously oversees communication between host and FPGA nodes in COPACOBANA. If any violation is detected, FPGA is marked with Error flag and Recovery process is initiated. The Monitor function also determines error type and severity. The application differs between these error severities:

Low	Most often caused by communication error or Read-after-Write operation fails.
Medium	Could be caused by incorrect data in FPGA memory or by other FPGA fail state.
High	This severity is used for cases when FPGA is not responding or if FPGA cannot be recovered by reset command.

In case of less severe error (low severity), the operation which caused error is repeated. If the error persists, error severity is increased. The medium severe errors are resolved by performing FPGA reset. The reset command is sent and FPGA is reset to initial state. Then, the subspace data are resent to target FPGA to restart computation. If the FPGA does not respond, or if it is unable to reset itself to initial state, highest severity error is invoked. The most severe errors are resolved by reprogramming FPGA chip. In case, the FPGA is still not working, it is excluded from further use in the attack.

Resume function allows to resume previously interrupted attack from last known position in search space. Entire search space is divided to $2^9 = 512$ subspaces which are sequentially processed. If computation process is interrupted, current settings and position in searchspace (ID of last verified subspace) are stored into configuration file. Then, these settings are loaded automatically on application start. In case, the configuration file is corrupted or missing, search space is verified from the beginning.

5 Testing

In this chapter, we describe how we tested our design. In the first section, design verification process is introduced. In the next section, we introduce how the design has been validated in COPACOBANA. Finally, we summarize achieved test results.

5.1 Design Verification

We have intensively tested implemented design and all its parts during design implementation. To simulate circuit behavior and test its function, we used Modelsim software. This software allows wide array of options and diagnostic tools which helped us to improve our tests.

We prepared several testbenches for every module and for design blocks. Also testbenches to verify entire design function were created. The verification process is iterative. Collected test results were used in order to improve design implementation. The verification process could be divided into phases as follows:

- Modules verification
- Blocks verification
- Verification of entire design

All modules and blocks are very complicated sequential circuits, hence the entire state space is huge. Therefore we were unable to perform exhaustive testing. Used test vectors were created in order to cover possible margin cases and also several random cases within the state space. We used slightly modified reference Hitag2 implementation in C, to generate data for the test vectors. All used testbenches and test results are available on the attached CD in folder Testing.

5.1.1 Modules Verification

The design consists of the following modules—Hitag2 core, Hitag2 cores controller, main controller and COPACOBANA bus interface. The custom testbench was created for each module. Because the modules strongly depend on each other, fundamental parts of implemented parent modules were used in testbench implementation.

Hitag2 core testbench mainly verifies proper function of implemented Hitag2 cipher. Testbench compares Hitag2 core output signal with pre-computed reference values from Hitag2 software implementation.

Hitag2 cores controller testbench focuses on testing vital controller parts and functions e.g. FSM, key generation function or control signals generation. However, test vectors for this module are very complicated. Therefore, we decided to create only a few test vectors to verify basic functions of controller. This module was then more extensively tested together with Hitag2 cores as an *executional block*.

Main controller testbench verifies FSM function and control word decoder function. We tested main controller with prepared test vectors and also with generated random data. Implemented testbench verifies control signals for Hitag2 cores controller and status data generation for application on a PC.

COPACOBANA bus interface module function was tested by reading and writing random data into module I/O registers. COPACOBANA bus control signals are generated and random data are read or written into inerface I/O registers.

Each module has been tested with a set of prepared test vectors. Once all modules were sucessfully tested, we started a block verification for entire design.

5.1.2 Block Verification

For testing purposes, the design was divided into two functional blocks—a communication block and an executional block. The communication block consists of main controller and COPACOBANA bus interface module. The executional block is composed of Hitag2 cores controller and group of Hitag2 cores.

Communincation block testbench is used to verify main controller communication with application on a PC via COPACOBANA bus. We created several communication scenarios to verify all implemented controller functions. Communication between COPACOBANA bus interface and controller module was also verified.

Executional block testbench simulates Hitag2 attack. Hitag2 cores controller is initialized with values from reference test vectors and circuit behaviour is monitored. This testbench focuses on Hitag2 cores controller functions verification.

Both, Communication and Execution block passed all prepared tests. Only one deviation was dispatched during tests. If Hitag2 cores controller is initialized with data for correct key 0xFFFFFFFFFFFF, attack is finished immediately after the first round. The key candidate verification is not performed with second set of initialization vector and authenticator. We did several tests and carefully explore this issue. The essence of this problem is caused by hardware implementation and optimalization requirements. To achieve good design performance, it is clear, that all signal values in hardware design have to be properly initialized. Hence, the reserved core for processing key candidates is initialized with key

0xFFFFFFFFFFF. Therefore, when the key candidate value conforms to this value, core reserved to verify key candidates immediately reports this key as correct.

Fortunately, this issue could be easily handled. It is necessary to swap input data (pair of initialization vector and authenticator) and then restart attack. In case when attack ends with the same result, key 0xFFFFFFFFFFF is correct and was successfully revealed, otherwise attack will continue as usual.

5.1.3 Design Verification

We have prepared testbench and testvectors to simulate implemented design function. Testbench simulates COPACOBANA environment and examines design functionality for one FPGA. Testbench is designed to work with 2 different subspaces. To verify all design functions, only the second subspace should contain correct key. The first subspace is then completely verified. Hitag2 breaker is restarted, new subspace is assigned and second subspace is verified.

Due to extreme data and time demand to simulate design function in software, we were unable to simulate design operation with the same configuration as for FPGA. We had to significantly reduce subspace size. The subspace size 14 bits (instead of 39 bits) has been picked. Then, verification of one subspace could be simulated in 25 minutes.

Several test vectors were created to verify all marginal cases. Our design successfully passed all prepared tests.

5.2 Design Validation

Verified design was validated by running in COPACOBANA. However, COPACOBANA does not allow in-circuit-debugging. Therefore, we were unable to extensively diagnose FPGA internal states. Therefore, we decided to validate design in two speed modes, namely in low speed mode and full-speed mode.

Low speed mode was mainly used to verify proper design function and correctivity of communication protocol implementation on physical hardware. DCM multiplier was set to 1, therefore the design operates at same frequency as COPACOBANA bus (f_{BUS}= 20 MHz). Because the design was synthetised as for the full speed mode, all timing constrains were met.

Full-speed mode tests were performed on maximal design operation frequency (90 MHz). All design functions and design capability to operate in selected frequency was verified.

We used the same test vectors as in design verification phase. However, the validation process is very time demanding, hence we had to reduce number of validation runs. We reduced number of test testvectors for random data in state space.

Implemented design successfully passed all validation tests. Collection of used testvectors and design validation results are available on the attached CD in folder Testing.

5.3 Summary

We have created several testbenches to verify implemented design and its functions. To compute reference values for test vectors, we used software implementation of the Hitag2 cipher [1]. We created test vectors to cover all possible marginal cases, and several random cases from state space. All implemented modules and functional blocks passed verification tests. One design limitation has been discovered and documented.

Finally, we have validated design function by series of tests in COPACOBANA.

6 Implementation Results

We have implemented Hitag2 cipher brute-force attack tool in FPGA platform, respectively COPACOBANA. Our implemented design contains 256 Hitag2 execution cores and 1 core reserved for key candidates processing. The hardware resources utilization of used Spartan3 XCS1000 FPGAs is 90 %. Maximal operating frequency is 90.2 MHz. We have decided to operate our design on frequency 90 MHz. Therefore one FPGA is able to verify about 378 million keys per second. The attack time with fully equipped COPACOBANA is only 103.5 minutes. Summarized results of our implementation are listed in Table 6.1.

6.1 Other Attacks Comparison

The software implementation of brute-force attack in PC platform requires about 800 clock cycles for one encryption operation. Hence the standard computer would be able to compute only about 2.4 million keys per second ($\tilde{}2^{21}$). Therefore a search of entire key space would require almost 4 years.

With the algebraic attack published in [5], authors are able to reveal entire 48 bits of key in 45 hours. There are required data from 4 sniffed transactions. Algebraic attack published in [6] does not contain enough data to prove clearly that it could be used with random initialization vectors, also the prerequisites to perform the attack are unknown.

Our brute-force attack implementation requires only 103.5 minutes. When compared to software implementation of brute-force attack on a PC, one FPGA chip outperforms PC by more than 140 orders of magnitude. Then a fully equipped COPACOBANA with 120 FPGAs outperforms implementation on a PC almost by 17 000 orders of magnitude.

Comparison of achieved attack times is in Table 6.2.

6.2 Design Limitations

According to our design concept, the implemented design has some theoretical limitations. During our work on design implementation, we have focused on achieving high clock frequency. In order to keep clock frequency high, we have to simplify the control module which leads to some limitations. Since we desist from implementing memory and memory controller into our design, we could eventually loss some of the key candidates. Second limitation lies in non-standard processing of the key 0xFFFFFFFFFFFF.

Possible candidate key loss—The limitation is caused by lack of cache memory for candidate keys in our design. The key candidates found by execution cores are verified with

Table 6.1: Implementation results

Parameter	Value
Target device	Spartan-3 XCS 1000
Chip utilization (Used Slices)	90 %
Number of Hitag2 cores	256 + 1
Maximal clock frequency	90.2 MHz
Used clock frequency	90 MHz
Keys verified per second	378 M
Attack time	103.5 min

Table 6.2: Attack time comparison

	Type of attack	Implementation platform	Attack time
HAR 2009	Algebraic	PC	N/A
ICS 2009	Algebraic	PC	45 hours
BF software implementation	Brute-force	PC	4 years
Our implementation	Brute-force	COPACOBANA	103.5 minutes

different pair of initialization vector and authenticator in reserved execution core. The key candidate is processed during following cipher round. Theoretical limitation lies in possibility that more than one key candidate is found during one iteration. Because there is only one reserved execution unit for verifying key candidates and the control module contain no cache memory, only one candidate key could be processed at once. In case two or more key candidates are found, then only one of them is processed and others are lost.

We suppose that losing one or more key candidates is very unlikely to happen. We assume that key candidates are uniformly distributed within the search space. However, there still exists a very small possibility that more than two or more key candidates could be very close to each other in searchpace.

We assume that there exists 2^{16}key candidates in searchspace in average. Each key has probability to be candidate equal to $p = \frac{2^{16}}{2^{48}} = \frac{1}{2^{32}}$

As we have implemented 256 execution cores, then the probability that 2 keys are candidate keys equals to

$$P = \binom{256}{2} * (\frac{1}{2^{32}})^2 * (1 - \frac{1}{2^{32}})^{(256-2)} \doteq \frac{1}{2^{51}}$$

Based on this very small probability of this event, we assume that its very unlikely that any key candidate would be lost.

Key 0xFFFFFFFFFFFF—Due to hardware implementation and optimalization requirements, it is necessary to initialize signal values properly in hardware design. Therefore re-

served execution core, which verifies candidate keys, is initialized with key 0xFFFFFFFFFFF. Even if there is no key candidate discovered, the value (0xFFFFFFFFFFF) is used as default key value. If this key is correct or if it is one of key candidates, the reserved execution core marks this key as correct and terminates computation. That would cause some problems. All FPGAs would report this key as correct, even if assigned subspace does not includes this key. Since the software application verifies if the resolved key belongs to assigned subspace, it would mark all FPGAs as faulty.

This issue could be easily handled by swapping input data (pair of initialization vector and authenticator) and restart attack. In case when the operation ends with the same result, the key 0xFFFFFFFFFFF is correct and was successfully revealed. Otherwise computation will continue normally.

7 Future work

In the first part of this chapter we discuss other possible implementation platforms as GPU or Playstation. In second part, we introduce next potential cipher weakness which would be further explored in future work.

7.1 Other Implementation Platforms

The brute-force attack method is very computationally demanding. Its performance strongly depends on performance of implementation platform. Performance improvements in implementation platform would seriously decrease execution time of brute-force attack. Implementation on RIVYERA, GPU or Playstation 3 platform could offer better performance.

RIVYERA [10] is a technological successor of the COPACOBANA and provides a backplane with up to 16 cards and a total of 128 high performance FPGAs. RIVYERA is based on Xilinx Spartan-3 5000 FPGAs. These FPGA chips contain 4.3 times more logic cells and functional blocks then Xilinx Spartan-3 1000 used in COPACOBANA.

The design implemented in this diploma thesis could be simply modified for different implementation platform. The Xilinx Spartan-3 5000 FPGA allows implementation of up to 1024 Hitag2 execution cores. In total RIVYERA with 128 FPGAs would be able to implement 2^{17}Hitag2 execution units.

We achieved 75.6 MHz clock frequency with 1024 Hitag2 cores with only parametric modification of design during experimental design implementation for Xilinx Spartan-3 5000 FPGAs. According to these results, design implemented on RIVYERA would be able to compute about $1.63 * 10^{11}$ keys per second. Complete brute-force attack implemented in RIVYERA would take less than 30 minutes.

The RIVYERA is also equipped with new high speed IO architecture, SSD and many new features. These features would also allow many design improvements and could lead to new design approaches which eventually could further improve performance of actual design.

GPU - This modern implementation platform offers a broad spectrum of design options. The most known are NVIDIA with CUDA technology [17], and AMD-ATI with AMD FireStream technology [18]. These technologies allow simple implementation of GPU based parallel computations through simple API. Both technologies are based on GPU's parallel throughput architecture that emphasizes executing many concurrent threads slowly, rather than executing a single thread very fast. Advantage of this platform is its availability and low price.

The Hitag2 cipher breaker implemented on GPU platform could eventually offer similar performance as FPGA platform but with advance of significantly lower price and better availability than FPGA platform.

Playstation - The Playstation 3 platform is based on Cell processor technology. Cell processor is based on one PowerPC based processing unit and 8 synergistic processing elements, six of them are available. Due to the nature of its applications, Cell processor is optimized towards single precision floating point computation. PlayStation 3's Cell CPU achieves 204 GFLOPS single precision float and 15 GFLOPS double precision. The Playstation 3 has 256 MB of Rambus XDR DRAM, clocked at CPU die speed.

The original PlayStation 3 also included the ability to install other operating systems such as Linux. Cluster of PlayStation 3 console is an attractive alternative to high-end systems and could provide high-performance computing.

An implementation of Hitag2 cipher breaker on Playstation 3 platform could achieve good performance and provide available and cheap alternative to other platforms.

7.2 Key Space Uniformity

In this diploma thesis we assume that the key candidates are uniformly distributed within key space. The key space could eventually contain places with deviations and non-uniform key candidates distribution. This property should be further explored and eventually more cipher weaknesses could be revealed.

8 Conclusions

The goal of this thesis was exploration of resistivity against attacks, namely the brute-force attack, for Hitag2 cipher. This thesis focuses mainly on hardware implementation of brute-force attack for FPGA target platform.

We have explored Hitag2 weak points and carefully considered possible attack methods and design approach options. According to analysis results we have selected brute-force attack as optimal attack method and picked parallel cipher cores model with one reserved core for key candidates processing as the optimal model for implementation in FPGA platform.

Then, we have successfully implemented brute-force attack in FPGA platform. Implemented design could contain up to 257 execution cores per one FPGA chip, whereas achieved clock frequency is 90.2 MHz. Thus, each FPGA chip is able to verify about 378 million keys per second. Therefore, a fully equipped COPACOBANA with 120 FPGA chips is able to verify entire key space in less than 2 hours (103.5 minutes). This achieved attack time outperforms all previous implementations by several orders of magnitude. Just two monitored communications between a Hitag2 transponder and a read/write device, instead of 4 sniffed transactions required in other published attacks, are sufficient to reveal the secret key. Moreover, the attack is arbitrarily parallelizable and thus could be run on multiple COPACOBANAs to decrease the time to find the secret key.

The whole attack operation is controlled by Java application, hence the application is platform independent on host PC.

During the implementation, we have prepared set of testbenches for implemented design. We have tested all modules with prepared set of testing vectors to verify their proper functionality.The test vectors cover all possible marginal cases, and several random cases from state space. Once the all modules in design were successfully tested, we have created a set of test vectors for entire design. The design functionality was tested in searchpace borders and with random data. Finally, we have validated our design by live tests in COPACOBANA. All implemented modules and functional blocks passed all tests.

It would be useful to explore implementation in other platforms such as RIVYERA, GPU or Playstation or examine key space uniformity in order to eventually reveal another Hitag2 weakness. We estimate that implementation in RIVYERA would reduce attack time to only 30 minutes.

9 Bibliography

[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 May 13, 2011.

[2] Nicolas T. Courtois and Sean O'Neil, FSE Rump Session – HITAG2 Cipher, 2008.
http://fse2008rump.cr.yp.to/00564f75b2f39604dc204d838da01e7a.pdf, as of May 13, 2011.

[3] I.C. Wiener, Crypto1 specification, reference implementation and test vectors, 2007-2008.
http://cryptolib.com/ciphers/crypto1/, as of May 13, 2011.

[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 May 13, 2011.

[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 May 13, 2011.

[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] SciEngines, RIVYERA.
http://www.sciengines.com/products/computers-and-clusters/rivyera-s3-5000.html, as of May 13, 2011.

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

[12] NKAAY, HITAG-2 Key Tool V50.
`http://www.nkaay.com/manual/hitag2.pdf`, as of May 13, 2011.

[13] Henryk Plötz, Analyzing an unknown access control system, 2007.
http://www2.informatik.hu-berlin.de/ploetz/analyzing-an-unknown-access-control-system.pdf,as of May 13, 2011.

[14] Mona Safar, M. Watheq El-Kharashi, Ashraf Salem, May 2006. *FPGA-Based SAT Solver*, In: *Electrical and Computer Engineering, 2006. CCECE '06. Canadian Conference*, pp.1901–1904.

[15] John D. Davis, Zhangxi Tan, Fang Yu, and Lintao Zhang, June 2008. *A Practical Reconfigurable Hardware Accelerator for Boolean Satisfiability Solvers.* In: *45th Design Automation Conference (DAC).*

[16] CryptoSAT solver project, 2011.
`https://gforge.inria.fr/projects/cryptominisat`, as of May 13, 2011.

[17] NVIDIA, CUDA, 2011.
`http://www.nvidia.com/object/cuda_home_new.html`, as of May 13, 2011.

[18] AMD, AMD FireStream, 2011.
`www.amd.com/stream`, as of May 13, 2011.

A List of used abbreviations

ASIC Application-Specific Integrated Circuit

COPACOBANA Cost-Optimized Parallel Code Breaker

CPU Central Processing Unit

DCM Digital Clock Manager

DES Data Encryption Standard

DIMM Dual In-line Memory Module

DRAM Dynamic Random-access Memory

EEPROM Electrically Erasable Programmable Read-Only Memory

FIFO First-In-First-Out

FPGA Field Programmable Gate Array

FSM Finite-state Machine

GFLOPS Giga Floating point Opetations per Second

GPU Graphics Processing Unit

GUI Graphical User Interface

ID Identification

IV Ininitalization Vector

LFSR Linear Feedback Shift Register

LUT Lookup Table

PC Personal Computer

RFID Radio Frequency Identification

RIVYERA Reconfigure Versatally your raw architecture

SAT Boolean satisfiability problem

SN Serial Number

SRL Shift Register LUT

TMTO Time-Memory tradeoff

VHDL VHSIC Hardware Description Language

B Content of CD

- Bibliography ..... Bibliography and information sources
- SRC ..... Source codes for HW and SW
- Testing ..... Testbenches and test vectors
- Stembera_2011.pdf ..... Diploma thesis text