HOCHSCHULE
RAVENSBURG-WEINGARTEN
UNIVERSITY
OF APPLIED SCIENCES

Evaluating encryption methods for the JTAG-debug port

Soham Sanjay Dekhane, Andreas Siggelkow

***Abstract*—Debug ports are one of the most important tools for designing, debugging, configuring and programming, however, they can be very vulnerable to a hacker with malicious intent. This project describes one of the many possible solutions to secure this debug port with the help of hardware based cybersecurity. This solution is demonstrated by designing a Pseudo-Random-Number-Generator (PRNG) using VHDL implemented on an FPGA and the seed exchange secured using the RSA algorithm.**

***Index Terms*—FPGA, RSA, PRNG, VHDL, LFSR, Cryptography, JTAG, Debug**

I. Introduction

Cyber threats present enormous risks to individuals, businesses, and even entire nations in today's connected world. Hardware cryptography is a vital defense against these threats. Strong key management and strong authentication are provided by cryptographic hardware modules. Only authorized individuals can access sensitive information and vital systems because of these hardware-based cryptographic solutions. Individuals and organizations can reduce the danger of unauthorized access, data leakage, and other criminal activity by utilizing hardware cryptography. Encryption is used in almost everyday life right from opening garage doors to credit cards. For such encryptions or cryptographic operations, generation of unique, random and secure keys is very important. Debug ports can be extremely helpful in designing embedded systems but they present a huge vulnerability. A hacker with malicious intent can get physical access of the system using this debug port can cause a huge damage. The obvious solution is securing the debug port by limiting the access to it by implementing some encryption algorithm. Hardware based cybersecurity; i.e. integrating cybersecurity measures in the processing unit of the embedded system itself can provide with various advantages; the biggest being that it is not at all vulnerable to cyber attacks unless the attacker has physical access to the hardware. In order to demonstrate such an example, this project focuses on designing a cryptographically secure Pseudo-Random-Number-Generator encrypting it using the RSA algorithm. Pseudo-Random-Number-Generators (PRNGs) are used for various applications like Key Generation, Initialization Vectors, Nonces etc. This makes the output generated by this PRNGs very crucial. A perfect PRNG should have three main attributes: Integrity (prevent undesired modification to the output), Authenticity (allow access only to authentic users) and Confidentiality (prevent secrets from becoming known to the attackers). One has to make sure that the PRNG output is not illegally disturbed i.e. the random output of a PRNG should not become non-random/predictable. A PRNG could be vulnerable to plethora of cyber attacks such as Algebraic attack, State recovery attack, Clock attack etc. Implementing these attacks makes sure that the output of the PRNG could be predicated or tampered with by the attacker. The PRNG should be initialized using a seed value which will be fed to the PRNG via the debug port. This seed value should be first encrypted and then sent to the Device-under-Test (DUT) using the JTAG interface. This seed value is then decrypted in the DUT, the output of the PRNG is generated, encrypted and sent back to the debug interface via JTAG. This is done so as to secure the generated output of the PRNG during the exchange via the debug interface during which it is the most vulnerable. The data does not have to be secured for operations within the DUT but only during the exchange via the debug interface. The hardware consists of a MAX10 FPGA in which the PRNG as well as the encryption algorithm are implemented using VHDL.

Soham Sanjay Dekhane, sohamdekhane@gmail.com, Andreas Siggelkow, andreas.siggelkow@rwu.de. Hochschule Ravensburg-Weingarten, Doggenreidstraße, 88250 Weingarten.

II. Pseudo-Random-Number-Generator

A device used to generate a sequence of random numbers or symbols is called as a Random Number Generator. Random numbers have been historically used in many applications ranging from Cryptography, Simulations, Machine Learning Algorithms, Computing Applications etc. There are various methods and algorithms used to generate random numbers. The generators are called as Random Number generators as the symbols or numbers generated by them have no mathematical or statistical relation amongst the generated sequence of numbers. Once such random number generator is a Pseudo-Random-Number-Generator (PRNG). A PRNG is designed using a deterministic algorithm to produce the sequence of numbers. These generated sequences pass all the "Statistical pattern tests for Randomness" but can easily be predicted

once the seed value (initial condition) or the algorithm used to generate the PRNG is known. Hence the term "Pseudo" is used. A PRNG at best can generate a total of $(2^n - 1)$ number of random numbers and the same sequence then repeats itself. There are various algorithms used to build the PRNG namely: Xorshift, Inversive congruential generator, ISAAC (cipher) (indirection, shift, accumulate, add, and count), Blum Blum Shub, Multiply with carry, Lagged Fibonacci Generator, Linear Feedback Shift Register, Mersenne Twister, Linear congruential generator and the Well Equidistributed Long-period Linear. In this project, the Linear Feedback Shift register (LFSR) algorithm is used for designing the PRNG. For designing a PRNG, a seed value i.e. an initial condition has to be specified from which the next sequences of the random numbers are generated. This seed value determines the length of the PRNG (i.e. the total number of random symbols or sequences generated before the pattern is repeated). As discussed before, the maximum possible length of a PRNG is $(2^n - 1)$; n being the number of bits of the seed value. A seed value of Hamming Weigth of $n/2$ is usually desired, n being the total number of bits in the seed value. Taps are also set up at desired bits. The bit values where the taps are located are XORed and the result of this XOR is placed at the $(n-1)^{th}$ bit, the rest of the bits are shifted to the right, and the 0^{th} bit is discarded or vice versa. PRNGs are vulnerable to various types of cyber attacks such as:

- Brute Force Attack: The attacker tries every possible seed value and checks if the observed sequence of numbers is obtained or not.
- Algebraic Attack: The output sequence of the LFSR can be described by a set of equations that an attacker can discover using algebraic methods. As a result, the attacker may be able to figure out the seed value and forecast future results.
- State Recovery Attack: If the attacker is aware of the previous results of the PRNG, they can predict the current as well as the future values of the PRNG.
- Clock Attack: The attacker forces the clock to skip or repeat certain states of the PRNG which will allow him to predict the future values of the PRNG.
- Side-Channel Attack: The attacker can use the leaked values of the electromagnetic emissions or the power consumption from the PRNG to determine the future values of the PRNG.
- Non-Linear Feedback Attack: An attacker could be able to create a set of equations that describe the LFSR's output sequence if it uses non-linear feedback. As a result, the attacker may be able to figure out the seed value and predict future results.
- Known Plaintext Attack: An attacker may be able to detect the state of the LFSR and anticipate future outputs if they have access to some of the plaintext that was used to generate the pseudorandom sequence.
- Birthday Attack: An attacker might be able to perform a birthday attack to find collisions in the key space and recover the key if the LFSR is being used to create cryptographic keys.

III. The RSA Algorithm

To encrypt the data during exchange or during debug, the RSA algorithm is implemented. RSA is the most well known Public Key Encryption method. It was developed by Ron Rivest, Leonard Adleman and Adi Samir in 1977 and is a type of asymmetric encryption method. Using such an algorithm is very advantageous as a key pair is always generated i.e. a public key and a private key. The public key is used for the encryption of the data while the private key which is used for decryption is kept a secret. The private key cannot be determined by using the public key and hence it can be distributed freely or even be published on websites. This is a pretty huge advantage of using asymmetric encryption method over a symmetric encryption method where the same key is used to encrypt and decrypt the data. The following steps illustrate the generation of the key for the RSA algorithm:

1) Select two distinct prime numbers; Assume they are p and q.
2) Compute their product "n" such that $n = p \cdot q$.
3) Calculate the Euler's totient function $\varphi(n) = (p-1) \cdot (q-1)$.
4) Select an "e" such that $0 < e < [\varphi(n)]$ and e & $\varphi(n)$ are coprime i.e $gcd(e, \varphi(n) = 1)$.
5) Calculate a "d" such that $d \mod \varphi(n) = 1$
6) (n, e) is the public key and is used for encryption while (n, d) is the private key and is used for decryption.

Once the public and private keys are calculated, the messages can be encrypted and decrypted as follows: Let x be the data and y be the encrypted data. Then, the encryption is done as $y = x^e \mod n$ while the decryption is done as $x = y^d \mod n$. Usually, x, y, n and d are $1024\,\text{bit}$ or more. Security level of $80\,\text{bit}$ is offered by RSA when $1024\,\text{bit}$ keys are used while a security level of $128\,\text{bit}$ is offered by RSA when $3072\,\text{bit}$ keys are used. The RSA algorithm is vulnerable to cyber attacks as well. By knowing the product "n", attackers can try to factorize the product and try to find out the prime numbers p and q. This type of cyber attack is known as a factorization attack. Currently, it is believed that it will be possible to factor $1024\,\text{bit}$ values within the next 10 to 15 years, and that intelligence agencies will likely be able to do it even sooner [1]. To minimize the risk of such an attack, it is recommended to choose the RSA parameters of 2048-4096 bits. The RSA algorithm is also vulnerable to Side-Channel attacks but for such an attack, the

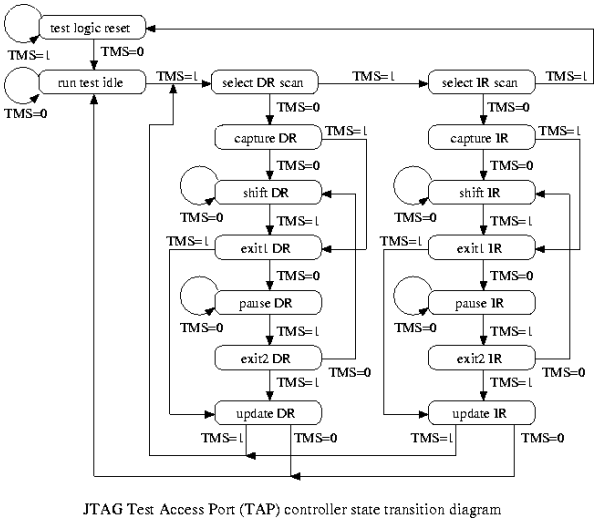


Figure 1. TAP-FSM

attacker must have access to the RSA implementation. Attackers try to find leaked information about the private key through the timing behaviour or power consumption.

IV. JTAG

Almost all digital systems have a debug interface [2] with different possibilities to attack the system [3]. This debug logic connects all sub-blocks in the system by means of a shadow bus system in order to test or debug it. This could act as a back door which is not secured. To equip this back door and all connection points of the debug bus with a lock, is the focus of the system introduced in the following. It is just the base system for different cypher/debug pairs, it is the base of a system evaluation.The lock can be a cypher system. Parallel to the debug problem is the update over the air possibility in such systems, especially modern cars and IoT. Also this back door can be secured by cyphering. An emulator of this kind has been presented in [4].

The back door itself is the well known JTAG (Joint Test Action Group) port [5]. The element, which accesses all logic on chip is the JTAG port together with the test access port (TAP) controller. The TAP-Controller is implemented as a finite-state-machine (Figure 1).

The signal timing is defined as follows: The **test mode select (TMS)** will be captured with every rising edge of **test clock (TCK)**. Also **test data in (TDI)** will be taken with the rising edge of TCK. Contrary to this, **test data out (TDO)** will be driven with the falling edge of TCK. So, the wiring to a second chip, which receives the output of the actual SoC, could be allowed a delay of one half of the period of TCK.

The FSM has 16 states. Two general states (test logic reset and run test idle) and seven states for the instruction register and seven for the data register. Changing from one state to the other, it is required to

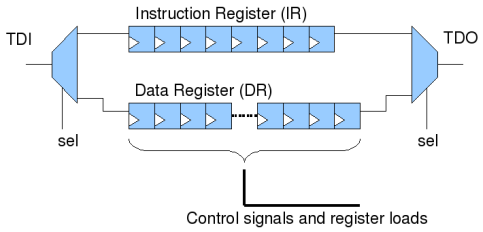


Figure 2. JTAG data register

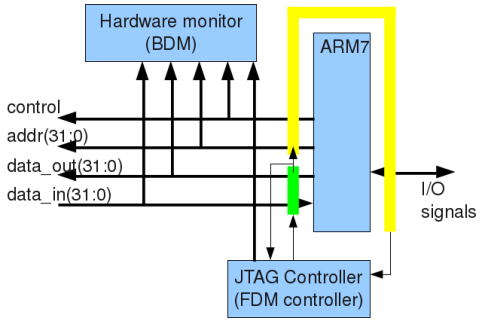


Figure 3. ARM7 wrapped by JTAG data registers

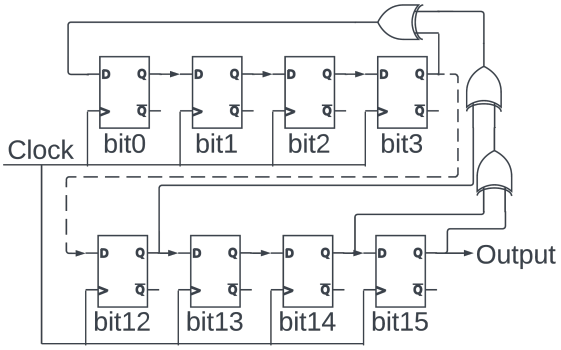


Figure 4. 16 bit PRNG

change TMS with the rising edge of TCK according to the diagram (Figure 1). According to the IEEE specification many tests will be predefined but for the debug purpose, the data register (Figure 2) is the important element. With the data register every design element in the system-on-chip (SoC) can be encapsulated (all inputs can be read and written, all outputs can be read and written) and is under total control of the TAP controller (Figure 3). Additionally, all flip-flops in the design are chained up to the so called scan chain, which is also a JTAG data register.

This is needed for debugging a SoC, but is obviously also a serious security threat.

To solve this security problem, different cypher methods on the data stream (TDI to TDO) will be researched.

V. Implementation

A. PRNG

The Pseudo-Random-Number-Generator is designed such that it produces 16 bit pseudo random number sequences by taking a seed value to initialize the sequence and XORing the bits of the last output by placing taps at certain bits. When the VHDL program is reset, the first input is taken as the seed which is taken through the testbench and that would be the first output produced by the PRNG. The 16 bit seed is carefully chosen keeping the Hamming weight of the seed as 8 in order to get a proper sequence of outputs. Taps are then placed at bits 4, 13, 15 and 16 i.e. from the previous output, the bit are positions 4, 13, 15 and 16 are XORed, the most significant bit from the previous out is removed, the remaining bits are shifted to the left and the result of XOR is placed at the position of the least significant bit. This 16 bit number will be the new output of the PRNG. As this is a 16 bit PRNG, we will have such $(2^{16} - 1)$ i.e. 65535 unique pseudo random outputs. As mentioned in this paper previously, these outputs can be used for applications like Key Generation, Initialization Vectors, Nonces etc. and hence need to be secured during the debug phase.

B. Encryption during the debug phase

The debug phase for a PRNG is the most critical phase and also the most vulnerable at the same time. The seed exchange as well as the exchange of the outputs is done between the test interface and the DUT over the debug interface. During this exchange, both the seed and the generated outputs are vulnerable to a hacker with malicious intent. To prevent this, in this project, the seed value and the generated outputs are secured using the RSA algorithm. For the RSA algorithm, a set of 64 bit public and private keys are generated. The seed value is first encrypted using the public key in the the testbench itself and transmitted over the debug interface to the DUT. This is then decrypted in the DUT using the private key and then given as an input to the PRNG. This is then processed by the PRNG and the output that is to be transmitted is again encrypted using the public key and transmitted over the debug interface.

VI. Conclusion and Future Work

In this project, the first steps have been taken to securing the debug interface. It has been demonstrated how the seed exchange over the debug interface can be encrypted in order to secure the leak of information due to a potential cyber attack. Further steps would include integrating a JTAG Tap controller with the PRNG and test the encryption using an ASIC tester. The complete ecosytem for the RSA algorithm also needs to be created such as a public key and private key infrastructure including a database for the keys that can be used for encryption and decryption purposes.

References

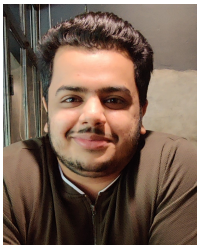
[1] Christof Paar and Jan Pelzl. *Understanding Cryptography*. Springer, 2010. ISBN: 978-3-642-04101-3.

[2] C.F. Kao and H.M. Chen. *Hardware-Software Approaches to In-Circuit Emulation for Embedded Processors*. IEEE Design and Test, 25 (5): 462 - 477, 2008.

[3] Swarup Bhunia, Sandip Ray, and Susmita Sur-Kolay (Editors). *Fundamentals of IP and SoC Security: Design, Verification, and Debug*. Springer, Cham, Switzerland, 2017. ISBN: 978-3-319-50055-3.

[4] Gregor Benz and Andreas Siggelkow. *Implementation of a GPS and GSM module into a Zynq Z7 SoC based emulator tracking system*. Workshop der Multiprojekt-Chip-Gruppe Baden-Württemberg, 2020.

[5] *IEEE Standard for Test Access Port and Boundary-Scan Architecture*. IEEE Std 1149.1-2013 (Revision of IEEE Std 1149.1-2001), pp.1-444, 2013. ISBN: doi: 10.1109/IEEESTD.2013.6515989.



Soham Sanjay Dekhane received his B.Tech. degree in Electronics and Telecommunication Engineering from Symbiosis International (Deemed) University, India in July 2021. Since September 2021, he is pursuing his Master's degree in Electrical Engineering and Embedded Systems at Hochschule Ravensburg-Weingarten.

Andreas Siggelkow received the academic degree Dipl. -Ing. in 1988 from the University of Karlsruhe. In 1996, he obtained his doctorate at the University of Stuttgart for Dr. -Ing. From 1996 to 2007 he worked for Infineon on specifications for base-band processor ASICs. Since 2007, he is a professor for ASIC-Design and Computer Architecture at the Hochschule Ravensburg-Weingarten.