Encryption-Based Secure JTAG

Emanuele Valea, Mathieu da Silva, Marie-Lise Flottes, Giorgio Di Natale, Bruno Rouzeyre

Encryption-Based Secure JTAG

Emanuele Valea[1], Mathieu Da Silva[1], Marie-Lise Flottes[1], Giorgio Di Natale[2], Bruno Rouzeyre[1]
[1]LIRMM (Université de Montpellier – CNRS), Montpellier, France
[2]Univ. Grenoble Alpes, CNRS, Grenoble INP*, TIMA, Grenoble, France

***Abstract*—Standard test infrastructures, such as IEEE Std. 1149.1 (JTAG), IEEE Std. 1500 and IEEE Std. 1687 (IJTAG), are widely used in nowadays Integrated Circuits (ICs). However, they pose an important security challenge to the designers because of the high controllability and observability they offer through the Test Access Port (TAP). For instance, malicious users can exploit test infrastructures in order to access the internal scan chains of crypto-cores and perform scan attacks. Moreover, these infrastructures connect all the devices of the system to the same network. For this reason, the data sent to a target device are potentially visible to all the others. Consequently, this poses a threat to the confidentiality of data content. The encryption of test data is a countermeasure that has been conceived in order to overcome these threats. In this paper, we propose a new secure version of the JTAG infrastructure, relying on stream-based encryption.**

Keywords—Test and Security; JTAG; Stream Cipher; Scan Encryption

I. INTRODUCTION

The test standard for board testing is the IEEE Std. 1149.1, also known as JTAG. The Test Access Port (TAP) allows the user to access every device on the board through a scan network, which connects them in a daisy-chain fashion. The same principle has then been extended to System-on-Chips (SoC), in order to facilitate the testing of internal cores. For this reason, the IEEE Std. 1500 provides that IP cores are equipped with standardized test wrappers. More recently, the IEEE Std. 1687, also known as IJTAG, has been released to facilitate the access to the hundreds of embedded instruments that are present in nowadays SoCs. The IJTAG is based on a Reconfigurable Scan Network (RSN). This is set by the user according to the instruments that must be reached. Most devices and IP cores contain scan chains for testing purposes. In complex SoCs, the scan network of the test infrastructure normally reaches the scan chain interface of each core. This avoids connecting each IP core to the external pins only for test purposes.

The access to the SoC test infrastructure by a malicious user represents a very serious security threat. The attacker can benefit from the enhanced controllability and observability offered by the infrastructure to compromise the whole system. For instance, having access to the scan chain content of a crypto-processor allows the user to retrieve its secret key. Several attacks on the hardware implementations of crypto-processors have been presented in the literature [1]-[4]. The JTAG interface can be exploited in a wide range of different attacks. This is the case when the test interface is used to access the system memory for debugging purposes, or to access the IJTAG reconfigurable network where some embedded instruments contain proprietary SoC configurations.

Another important threat is related to the presence of potentially malicious devices or IPs inside the system. They usually share the same daisy-chain connection with the others. Thus, all data sent to a target device can potentially be shifted through all the others. If confidential data is shifted through the network, malicious devices can observe and possibly alter such data [5].

The encryption of test data has been proposed as a countermeasure [5][6]. The encryption is performed relying on a secret key, which is shared between the user accessing the device and the device itself. The primary security feature provided by data encryption is the confidentiality of the communication. In fact, an attacker eavesdropping the exchanged data is not able to extract any useful information from the encrypted messages. Moreover, when test data encryption is employed, an unauthorized user is not able to successfully communicate with the target device.

Many proposals of test data encryption are based on stream ciphers, due to their lower cost compared to block ciphers. However, security vulnerabilities are easy to be spotted in this kind of implementation. For this reason, a secure implementation of the stream cipher for scan chain protection has been proposed in [7]. Nevertheless, the solution in [7] is limited to the protection of the scan chains.

In this paper, we propose a protection for the JTAG infrastructure, relying on the encryption of test data based on the stream cipher. Besides protecting the scan chains, this solution secures the IJTAG network, and the access to the memory through the test interface (e.g. software update). The fundamental of this idea is that the user must engage an encrypted communication every time specific *confidential* instructions are executed.

The remainder of this paper is organized as follows. Section II presents the threat model that the proposed countermeasure is intended to prevent. Section III lays the background on existing countermeasures based on stream ciphers. Section IV presents the proposed countermeasure. Section V reports the performance of this solution. Eventually, Section VI draws some conclusions.

II. THREAT MODEL

In this Section, we present the threats related to test infrastructures. At first, we provide a description of the different standards, which are accessible through the JTAG interface. Finally, we detail the attacks and the threats that we want to face with the proposed countermeasure.

A. Test infrastructures

Fig. 1 shows how the three test standards (i.e. JTAG, IEEE 1500 and IJTAG) can be found integrated together in the same system. Usually, a JTAG interface, including TDI, TDO, TCK and TMS signals is offered to the user at board level. Each device contains a TAP controller for internally interfacing with the test ports, i.e. an FSM driven by the TMS and TCK signals, which are generally distributed to all the TAP controllers. Test data are shifted through all the

*Institute of Engineering Univ. Grenoble Alpes

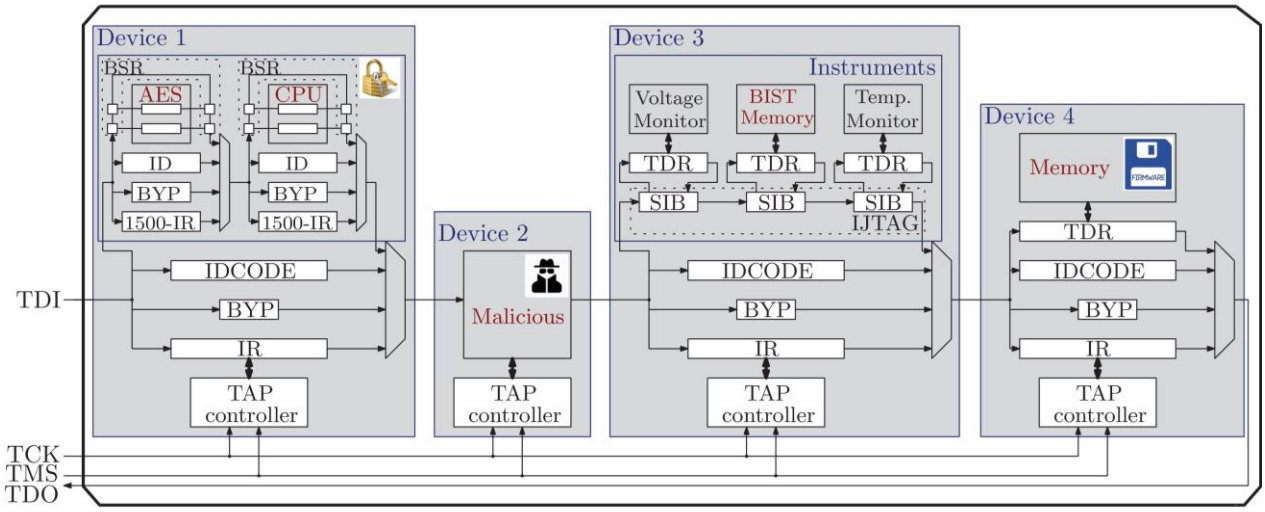

Fig. 1 Example of JTAG daisy-chained components: (*Device 1*) a SoC containing an AES crypto-core and a CPU, (*Device 2*) a malicious device, (*Device 3*) a device with embedded instruments accessible through an RSN, (*Device 4*) a memory containing firmware.

devices using the TDI/TDO signals. The Instruction Register (IR) in each device is first loaded with an instruction to be executed on each device. Some instructions are mandatory in the standard (*BYPASS*, *SAMPLE/PRELOAD*, *EXTEST*), but many other custom instructions can be added (e.g. *INTEST*, *RUNBIST*). Each instruction connects a specific Test Data Register (TDR) to the external test interface. For example, the *BYPASS* instruction connects one flip-flop, called BYP, between the TDI and TDO pins. The *EXTEST* instruction is used to test off-chip circuits and board level-interconnections by accessing the Boundary Scan Register (BSR). The *INTEST* instruction allows the tester to shift test patterns into the circuit and to collect test responses, potentially by accessing the internal scan chains of the device. Each device has a proper identification code, accessible with the *IDCODE* instruction and stored into the identification register.

When the device is a SoC, such as *Device 1* in Fig. 1, a specific instruction gives access to the IEEE 1500 network and thus to the IEEE 1500 compliant test wrappers of the IP cores. Another option is to have a specific instruction that gives access to an IJTAG reconfigurable network, such as in *Device 3* (Fig. 1). In this case, the external user configures the network by opening the selected SIBs in order to access the TDRs related to the specific instruments. Specific TDRs can also be integrated and associated to a customizable instruction, such as in *Device 4* (Fig. 1) where the TDR allows updating the firmware in the memory.

B. *Attacks exploiting the test infrastructure*

JTAG provides essential access to on-chip instrumentation and facilities for test and debug. Test infrastructures can also be exploited to possibly steal critical information or to disturb the system operations. One threat comes from embedded devices developed by untrusted third-parties. Moreover, an external attacker can also exploit the test infrastructure in order to discover the secret key of a crypto-processor.

1) *Untrusted third-party devices*

In the depicted scenario of a typical test infrastructure, when the user gains access to an internal feature of the target device, the involved data are shifted through the same test daisy-chain. As depicted in Fig. 1, TDI/TDO serial connection is used to access all embedded devices in the system. Consequently, data sent to (or read from) any device is propagated through the other chained devices. A malicious device connected to the network could easily steal confidential data from other devices by intercepting the test data issued from other cores or from the external tester [5]. For instance, the malicious IP can intercept the critical configuration data sent to a device by the user, such as firmware content, FPGA configuration, or the configuration bits in the IJTAG network.

2) *Malicious users*

A user accessing the test infrastructure can also control and observe the internal scan chains of the target device. If the target device is a crypto-core and its chain is accessible, it is possible to execute a differential scan attack aiming to disclose its secret key. The scan attack has been proposed in [1] to retrieve the AES secret key. The computation of the 128-bit AES is performed in ten iterative rounds. Each round is composed of several operations based on substitution and permutation, resulting on confusion and diffusion of the plaintext data on the ciphertext. The result of each round is stored into a dedicated round register, which is usually part of the scan chain for higher testability. The attack targets the result of the first round, when stored into the round register, after one cycle of computation. The procedure consists in switching the circuit to test mode after the first round, and scanning out the partially encrypted result, stored into the round register. The attacker carries out the differential scan attack by applying chosen plaintext pairs, and then calculating the Hamming distance between the two first-round results. When the result of the Hamming distance calculation hits some specific values, the attacker is able to identify one byte of the secret key. The attack strategy consists in trying several plaintext pairs until the difference between two intermediate results allows the attacker to determine one key byte. The attacker repeats these steps for all the bytes of the key, in order to retrieve the whole secret behind the AES crypto-core.

The test infrastructure can be used to access a large amount of sensitive information. The exploitation of the debugging features offered by the JTAG, can allow an attacker to steal firmware and software configuration data. Moreover, the on-chip debugging capabilities can be exploited to perform attacks where the memory tampering leads to privilege escalation of the user on the system.

III. STATE-OF-THE-ART

To protect against the threat model described in the previous Section, the encryption of the test communication

has been proposed in the literature. The encryption scheme consists in firstly encrypting off-board the data. The user then sends the encrypted data to device. The receiver device decrypts them using the same secret key used by the user. The device is able at this point to use the data for its internal operations. Finally, the responses that have to be sent back to the user are encrypted. The user has to decrypt the received encrypted responses outside the system to compare them with the expected ones.

This mechanism guarantees that the other devices, sharing the same test infrastructure of the target one, are not able to understand the data content. Moreover, a user that does not know the secret key used by the stream cipher inside the circuit, is not able to have a successful communication with the target device.

The TRIVIUM stream cipher [8] has been preferred so far for the encryption of test data. This preference is due to its low implementation cost and its easy adaptability to the serial interface offered by the TDI/TDO signals of the test infrastructure. The TRIVIUM is initialized with a secret key and an initialization vector *IV*.

In [5], [9] and [10] the JTAG communication is encrypted resorting to the TRIVIUM stream cipher. The *IV* is normally hardwired into the device. The secret key is either fixed inside the device or provided by the user. Since the *IV*/key couple does not necessarily change between different encryption sessions, two-time pad attacks can be performed [7], making these countermeasures ineffective.

The solution proposed in [7], relies on the random generation of the *IV* by the device. This is sent to the user that uses it to encrypt data for a single encryption session. This way a different *IV* is used at each encryption session. Using this methodology prevents the implementation of two-time pad attacks, making the encryption secure. However, the proposed solution is limited to secure the scan chains, and does not protect against all security threats in the JTAG interface.

IV. PROPOSED COUNTERMEASURE

We propose a modified JTAG infrastructure that deploys the encryption of the data when confidential instructions are executed. As in [7], the *IV* of the stream cipher is randomly generated in order to ensure a secure implementation of the stream cipher.

A. IV and key management

In order to never re-use the same keystream to encrypt test data, the random *IV* is generated by a True Random Number Generator (TRNG) at every circuit reset. For this reason, the stream cipher is not exposed to two-time pad attacks. However, the random value of the *IV* has to be known by the external tester/debugger. This way, he/she can correctly communicate with the target device. For this reason, a specific TDR is added, which contains the *IV* value that the user can read executing a custom instruction, called *GETIV*. With this method, the randomly generated *IV* is shared with the external world, making this value publicly known. Even if an attacker can read the *IV*, the security is not compromised, since the key of the stream cipher remains secret.

Concerning the stream cipher key management, we assume that the target device embeds at least one crypto-processor. Therefore, a secure storage and a Secure Key Management Unit (SKMU) is usually present inside the SoC. We propose to securely store the stream cipher key in the secure storage, already containing all the secret keys of the circuit. We also propose to re-use the SKMU of the crypto-processor to share the secret key of the stream cipher with authorized users. The SKMU also performs the key generation, activation and revocation during the life cycle of the stream cipher key. The re-using of the SKMU, already embedded within the circuit, has the advantage of mitigating the additional cost due to the key management.

B. Operating principle

For the sake of clarity in presenting the principle of the solution, we consider a SoC embedding a crypto-processor, hence susceptible to be the target of differential scan attacks. An attacker can shift in and out the scan content of the circuit using the *INTEST* instruction, provided by the JTAG standard. The proposed solution is effective in protecting against a malicious use of the *INTEST* instruction. We suppose that, when the *INTEST* instruction is executed, the TDR that is connected between the TDI and TDO signals is the internal scan chain. In the proposed solution, the content of data, shifted through the device after the execution of the *INTEST* instruction, is encrypted with the stream cipher. Since the generation of the *IV* is performed randomly, every single device generates a different random value, thus reducing the efficiency of the manufacturing test. Indeed, after the production of the ICs, many dies are usually tested in parallel, directly probing the silicon wafer, using the same test patterns. If the proposed scheme is also activated in this phase, parallel testing is not possible anymore, since each circuit requires the test patterns encrypted with a different keystream. In subsection 1) we explain how to use the proposed countermeasure to easily perform parallel wafer testing, while in subsection 2) we explain how the countermeasure is used in mission mode.

1) Wafer testing

The first test of newly fabricated integrated circuits is performed when the dies are still part of the silicon wafer. In order to save test time during manufacturing test, several dies are tested in parallel by applying the *same patterns* at the *same time*. As explained above, implementing the proposed encryption technique forces the applied test patterns to be unique for every single circuit, because they must be encrypted resorting to a random number that differs from one circuit to the other. Therefore, the proposed solution cannot be used for parallel testing of multiple circuits.

To thwart this disadvantage, we propose to disable the use of the TRNG during the manufacturing process and to use a predefined hardwired *IV* for all circuits. In this way, all keystreams are identical and all test patterns can be encrypted in the same way. To bypass the TRNG when the circuit is still on the wafer, we propose to use in-wafer sensors, able to identify whether the die is still part of the wafer. These sensors are either based on OTP (One Time Programmable) memories, or on the so-called *Saw Bow*. The latter is based on an electrical connection made by a strong pull-up and a weak pull-down elements, which are

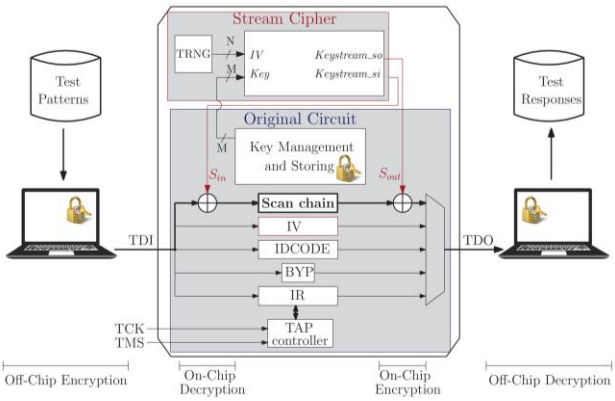

Fig. 2 High-level architecture of Secure JTAG

physically interconnected by a metal line across the sawing lines of the wafer. The strong pull-up resistance sets a logic value on the line when the sawing line is intact. When the dies are sawed, the weak pull-down resistance sets the opposite value on the line [11].

2) Mission mode

When in mission mode, the circuit can be the target of an attack. The principle of the solution is to encrypt the test data using the stream cipher, as illustrated in Fig. 2.

The stream cipher generates two keystreams: S_{in} for decrypting test data shifted into the scan network, and S_{out} for encrypting test data shifted out of the scan network. We have decided not to use a single stream cipher to generate the keystreams for both input decryption and output encryption in order to avoid any temporal correlation in the generated keystreams. The generation of independent keystreams can be done in an efficient way for some standard stream ciphers. For instance, the TRIVIUM [8] requires very few additional logic gates (i.e. 3 AND gates and 11 XOR gates) to generate two keystreams at the same time.

The utilization of the proposed countermeasure consists in an initialization phase and a successive encryption phase. During the initialization phase, the TRNG generates the *IV*, and sends it to the circuit implementing the stream cipher, in order to perform its setup. During this phase, the TAP controller locks the use of the *INTEST* instruction. If an external user requires this instruction, the TAP controller remains set on bypass mode. When the initialization phase is completed, the tester can ask for access to the protected instruction. The external user has to execute a specific instruction, called *GETIV*. When executed, this instruction connects a special register, containing the generated *IV* value, to the TDI/TDO signals. This way, the tester can shift out of the device the *IV* that has been produced by the TRNG during the initialization phase. If the *GETIV* instruction is executed before the initialization phase is completed, a sequence of all '0s' is returned as response.

During the encryption phase, the tester knows both the secret key (if the tester is authorized) and the *IV* obtained via the *GETIV* instruction. From this moment, it is possible to encrypt off-chip the test patterns using the *IV* recovered from the device. At this point, the test patterns, shifted through the TDI for the *INTEST* instruction, are decrypted on-chip before being introduced into the corresponding TDR. During the scanning-out operation, the test responses are encrypted on-chip. The tester collects encrypted responses from the TDO interface that can be decrypted off-chip, using the same *IV* and secret key used for the off-chip encryption.

The solution is not limited to classical testing purposes with the protection of the *INTEST* instruction and the scan chains, but the countermeasure can be extended to a whole set of protected instructions whose involved data can benefit from encryption. The designer has to define a set of protected instructions. For example, as shown in Fig. 3, in addition to the *INTEST* instruction, the protected instructions can include (1) the *IJTAG* instruction accessing the RSN including critical instruments, and (2) any instruction accessing a TDR containing confidential data, such as firmware updates of the device. Therefore, the stream cipher encrypts the data content addressed to these TDRs: (1) the RSN is encrypted, making the proper configuration of the RSN by unauthorized users more difficult, and preventing to read and control the TDRs associated to the instruments; (2) the firmware is decrypted before being saved into the memory, preventing a malicious user to sniff its content during an update process.

C. Control Unit

The stream cipher initialization procedure is controlled by an FSM, whose state transition graph is given in Fig. 4. The FSM is composed of 4 states (i.e. START_TRNG, SHIFT_IV, SC_SETUP, SC_ENCRYPT) and it outputs three control signals (i.e. *enable_reg*, *start_SC*, *init_completed*). All of them are initialized to '0'.

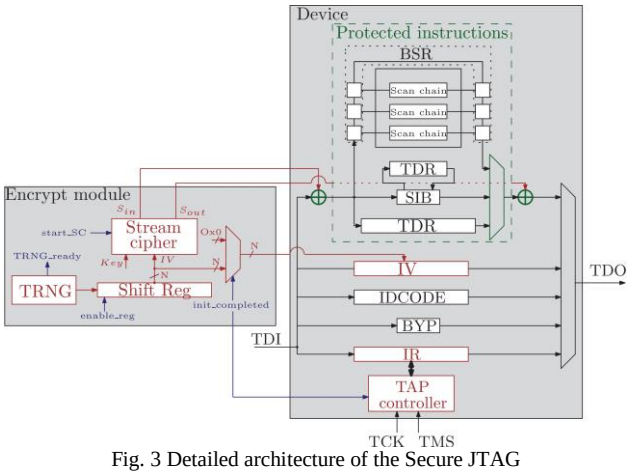

Fig. 3 Detailed architecture of the Secure JTAG

At reset, the TRNG starts the initialization, while in the START_TRNG state. TRNGs have usually an initial set-up time during which the generated numbers are not random enough. They require some time to reach sufficient entropy. Therefore, during this period, the generated value cannot be used. As soon as the TRNG reaches a good entropy ($TRNG\_ready = '1'$), the *IV* generation begins.

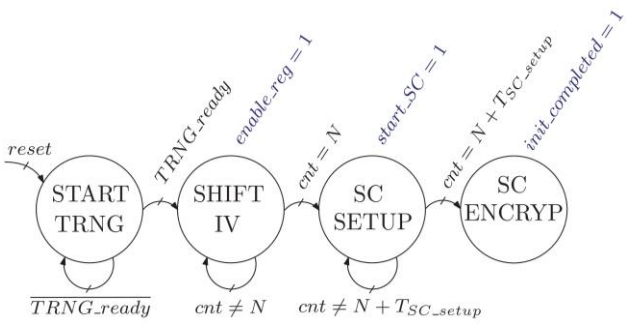

Fig. 4 Finite State Machine controlling the initialization procedure

During the SHIFT_IV state, the shift register (*Shift Reg* in Fig. 3) receives the random bitstream generated by the TRNG ($enable\_reg = '1'$). A counter cnt is launched at the same time. When cnt reaches the value *N*, the TRNG stops generating the random bitstream. *N* is equal to the number of bits of the *IV*. When the *N* bits of the random *IV* are generated, the TRNG is no longer used and it becomes available to other applications, if needed. Otherwise, it can be turned off.

Once the counter have reached the value *N*, the control unit goes to the SC_SETUP state and starts the stream cipher initialization ($start\_SC = '1'$). The FSM stays in this state for a time equal to T_{SC_setup}, needed for the stream cipher setup.

Once the counter reaches $N + T_{SC\_setup}$, the initialization process is completed ($init\_completed = '1'$). The stream cipher encrypts the data passing through the TDI and TDO terminals of the protected TDRs. The keystreams are generated only in the case in which the TAP controller is in the *Shift-DR* state and a protected TDR is selected by the instruction under execution. In the other cases, the encryption is not needed and the stream cipher is deactivated and it generates no keystream.

During the SC_ENCRYPT state, when the tester wants to write into a protected TDR, at first it executes the *GETIV* instruction to read the *IV*, in order to encrypt data using the shared secret. After that, the tester executes the protected instruction, in order to access the corresponding TDR. The tester places the IC into *Shift-DR* state where the stream cipher generates the keystream. The tester shifts in the encrypted data, which are decrypted before being sent to the TDR. The tester places then the IC into the *Exit-DR* state, in which the stream cipher stops the keystream generation. At the end of the operations, the TDR contains the plaintext data.

The initialization process cannot be interrupted. The control unit ensures the setup completion before any possible operations on the protected TDRs. If a circuit reset occurs, the control unit is reinitialized and the TRNG generates a new *IV* for the stream cipher.

V. PERFORMANCE ANALYSIS

In this section, we evaluate the security of the proposed solution and the implementation costs, measured employing the TRIVIUM [8] stream cipher.

A. Security analysis

Since the test communication is encrypted, the solution ensures that an untrusted third-party device in the test daisy-chain is not able to sniff confidential data or to send undesired data to another device.

Concerning the scan attacks, the stream cipher encrypts the scan chain content when the instruction *INTEST* is executed. The decryption of the scanned-in test data prevents to set desired values in the scan network without knowing the secret key. The encryption of the scanned-out test responses prevents to observe the internal states of the circuit under test without first performing the decryption. The stream cipher is initialized with a random *IV* preventing the exploitation of the two-time pad to perform a differential scan attack on encrypted test data, as is the case for the state-of-the-art stream-based countermeasures [5][9][10].

When the encryption is performed on an IJTAG network, an attacker without the knowledge of the RSN configuration faces troubles to open or close the SIBs, due to the keystream XORed with the test data at the RSN input. Even if the attacker is able to configure the RSN in a chosen configuration, he or she is not able to send and read data related to confidential instruments, since the data shifted through the RSN go through the input decryption and the output encryption.

When the test interface is used to configure a memory (e.g. for firmware updates), the encryption ensures that the content cannot be readable without knowing the secret key. Moreover, an attacker cannot update the firmware with a corrupted version due to the decryption performed on the data sent through the test interface.

B. Implementation costs

We have chosen to implement the TRIVIUM stream cipher in the proposed solution due to the low area overhead. The level of security that it guarantees is high enough for the target application. However, other stream ciphers can be used instead of TRIVIUM in the proposed infrastructure, especially if better stream ciphers will be developed in the future.

We do not consider the cost of the TRNG in the experimental results. In the case where a TRNG is already implemented in the original circuit, the proposed countermeasure can exploit this TRNG during the initialization process, implying no cost overhead for the random number generation. As previously shown in Section IV, after the generation of the *IV*, the TRNG is no longer useful for the proposed solution, and it can be used by another application. On the other hand, if no TRNG is available in the circuit, the implementation implies an additional area cost. This is evaluated as 15 000 Gate Equivalents (GEs), as is the case of the TRNG from the Synopsys DesignWare IP library [12].

1) Area and test time overheads

To evaluate the area overhead, we have considered a simple JTAG wrapper implementing a TAP controller, the IR, the BYP, and the *IDCODE* registers. As described in Section IV, the JTAG wrapper is modified to include the *GETIV* instruction and its associated register. Moreover, some modules are added in addition to the modified JTAG wrapper: the TRIVIUM stream cipher, the shift register containing the random *IV* and the control unit. Tab. 1 reports the area cost of the proposed solution compared to the original JTAG wrapper, representing an area overhead

Modules	Original JTAG (GEs)	Proposed solution (GEs)
JTAG wrapper	625	1 147
TRIVIUM	/	2 048
IV Shift Register	/	300
Control Unit	/	252
Total	***625***	***3 747***

Tab. 1 Area cost of the proposed countermeasure compared to the original JTAG wrapper

of 500%. The solution is dedicated to large devices, such as SoC designs. For instance, the solution implemented on a LEON3 processor, whose area is 1 902 095 µm², increases the total area of 7 794 µm², i.e. a 0.41% overhead.

Concerning the test time cost, the proposed solution introduces only an overhead due to the initialization process. This process takes 80 clock cycles to shift the random *IV* into the shift register, and 1152 clock cycles for the TRIVIUM setup. After this initialization process, the tester has to recover the *IV* executing the *GETIV* instruction before starting the encrypted test communication with the device. This corresponds to 80 clock cycles to shift out the content of the *IV* register. In addition to the time required to generate a random number, the solution implies a test time overhead of 1312 clock cycles at the beginning of a test procedure. This test time overhead has to be compared with the whole test sequence of the circuit. For instance, in the LEON3 processor, 11 612 051 clock cycles are needed to achieve a test coverage of 70%. Thus, the stream-based countermeasure introduces an overhead of only 0.01%, without considering the time to initialize the TRNG.

2) Test coverage

The stream cipher encryption of the JTAG interface does not affect the testing of the original circuit through the protected *INTEST* instruction. The test coverage of the whole circuit is not reduced. However, the architecture of the proposed solution must also be tested, without the help of scan chains that would expose the stream cipher to scan attacks. We propose to functionally test the cipher using the test data targeting the circuit under test.

Stream ciphers based on shift registers, such as TRIVIUM, are easily testable, since all the states of the stream cipher are shifted out the circuit as a keystream. The consequence is that the errors generated by possible faults are easily propagated to the circuit outputs during the encryption. To validate this assumption, we have evaluated the test coverage on the TRIVIUM stream cipher using the test sequences of several original circuits. At scan-input, test patterns are processed by the input keystream generated by the stream cipher, and the test responses are processed by the output keystream. We have performed experiments with the test sequences targeting several cores (Pipelined AES-256, Triple-DES, Pipelined AES-128, RSA 1024 and LEON3 processor). In all cases, the fault coverage for stuck-at faults in the proposed architecture is 100%.

VI. CONCLUSION

Test interface gives the access to unauthorized users, enabling them to perform attacks. Moreover, malicious devices connected to the test daisy-chain can sniff and tamper test data. To prevent these threats, several solutions based on stream cipher encryption have been proposed. In this paper, we have present a secure JTAG interface based on stream cipher encryption. The stream cipher used for the encryption is initialized with a different random *IV* at each reset, preventing the generation of the same keystream twice. The content is encrypted with a secret key developed for the current activity, and shared with the authorized users employing the key management system already present in the circuit. The encryption is performed on the TDR content of a set of protected instructions, established by the designer. The protected instructions can include the ones accessing the scan chains, the IJTAG network, or updating the memory. This solution presents a marginal overhead on area and test time, and can be integrated in a SoC, without causing issues in testing the other cores connected to the test daisy-chain.

ACKNOWLEDGMENT

This project has been partially funded by the French Government (BPI-OSEO) under grant FUI#20 TEEVA (Trusted Execution EVAluation) and by the French government under the framework of the PENTA HADES (“Hierarchy-Aware and secure embedded test infrastructure for Dependability and performance Enhancement of integrated Systems”) European project.

REFERENCES

[1] B. Yang, K. Wu, and R. Karri. Secure scan: a design-for-test architecture for crypto chips. In DAC, pp 135-140. ACM, 2005.

[2] J. Da Rolt, G. Di Natale, M.-L. Flottes, and B. Rouzeyre. Scan Attacks and Countermeasures in Presence of Scan Response Compactors. In ETS, pp 19-24, 2011

[3] J. Da Rolt, G. Di Natale, M.-L. Flottes, and B. Rouzeyre. Are advanced DfT structures sufficient for preventing scan-attacks? In VTS, pp 246-251. 2012.

[4] J. Da Rolt, G. Di Natale, M.-L. Flottes, B. Rouzeyre, A Novel Differential Scan Attack on Advanced DFT Structures, In ACM TODAES, Volume 18, Issue 4, October 2013, Article No. 58,

[5] K. Rosenfeld & R. Karri (2010). Attacks and defenses for JTAG. IEEE Design and Test of Computers, 27(1), 36–47.

[6] M. Da Silva, M. Flottes, G. Di Natale and B. Rouzeyre (2018). Preventing Scan Attacks on Secure Circuits Through Scan Chain Encryption. In IEEE TCAD.

[7] M. Da Silva, E. Valea, M. l. Flottes, S. Dupuis, G. Di Natale and B. Rouzeyre (2018). A new secure stream cipher for scan chain encryption. In IVSW 2018.

[8] C. De Canniere & B. Preneel (2005). TRIVIUM Specifications. ECRYPT Stream Cipher Project, Report, 30, 2005.

[9] K. Rosenfeld & R. Karri (2011). Security-aware SoC test access mechanisms. In VTS, 100–104.

[10] S. Kan, J. Dworak & J. G. Dunham (2017). Echeloned IJTAG data protection. In AsianHOST 2016.

[11] Di Natale, G., Flottes, M.-L., Rouzeyre, B., & Pugliesi-Conti, P.-H. (2017). Manufacturing Testing and Security Countermeasures. In N. Sklavos, R. Chaves, G. Di Natale, & F. Regazzoni (Eds.), Hardware Security and Trust: Design and Deployment of Integrated Circuits in a Threatened Environment (pp. 127–148). Cham: Springer International Publishing.

[12] Synopsys. (2015). DesignWare True Random Number Generator Core.