A New Scan Attack on RSA in Presence of Industrial Countermeasures

Jean da Rolt, Amitabh Das, Giorgio Di Natale, Marie-Lise Flottes, Bruno Rouzeyre, Ingrid Verbauwhede

A New Scan Attack on RSA in Presence of Industrial Countermeasures

Jean Da Rolt *, Amitabh Das **, Giorgio Di Natale *, Marie-Lise Flottes *, Bruno Rouzeyre *, and Ingrid Verbauwhede **
*LIRMM (Université Montpellier II /CNRS UMR 5506)
Montpellier, France, {darolt, dinatale, flottes, rouzeyre}@lirmm.fr
** Katholieke Universiteit Leuven, ESAT/COSIC, Leuven, Belgium, {amitabh.das, ingrid.verbauwhede}@esat.kuleuven.be

Abstract. This paper proposes a new scan-based side-channel attack on RSA public-key cryptographic implementations in the presence of advanced Design for Testability (DfT) techniques. The attack is performed on an actual hardware implementation, for which different test scenarios were conceived (response compaction, X-Masking). The practical aspects of scan-based attacks on the RSA cryptosystem are also presented. Additionally, a novel scan-attack security analysis tool is proposed which helps in evaluating the scan-chain leakage resilience of security circuits.

Keywords: Scan-attacks, public-key cryptography, DfT methods

1 Introduction

Security is a critical component of information technology and communication and is one of the levers of its development because it is the basis for establishing confidence to end users. Among the security threats, the vulnerability of electronic equipment that implement cryptography which enable the necessary services of confidentiality, identification and authentication, is perhaps the most important. Some fraudulent access or "attacks" on the equipment to extract sensitive information, such as encryption keys, undermine the whole chain of secure transmission of information. One of these attacks exploit the scan-chain Design for Test (DfT) infrastructure inserted for testing the equipment. Testing acts like a double-edged sword. On one hand, it is very important to test a cryptographic circuit thoroughly to ensure its correct operation, and on the other hand, this test infrastructure may be exploited by an attacker to extract secret information.

There have been many scan-attacks on cryptographic circuits proposed in the literature[1][2], which focus on extracting the stored secret key. Once the secret key is retrieved, more confidential data may be stolen. These attacks rely on the observability of intermediate states of the cipher. Even if the cryptographic algorithms are proven to be secure, accessing their intermediate registers compromise their strength. The process to mount a scan-attack is as follows: First the cipher plaintext input is set to a chosen value, then the circuit is reset, followed by its execution in normal mode

for some cycles, and finally the circuit is switched to test mode and the scan contents are shifted out. By repeating this multiple times with different chosen plaintexts, the scan contents may be analyzed to find the secret key. In the case of scan-attacks the basic requirement is that the cipher operation may be stopped at any moment, and the contents of the intermediate registers can be scanned out, thus compromising the hardware implementation of the cryptographic algorithm. A common technique adopted by many smart-card providers is to disable the test circuitry (such as JTAG) after manufacturing test. This solution may not be acceptable for systems which require test and debug facilities in-the-field. High quality test is only ensured by full controllability and observability of the secure circuit, which may compromise security. Another alternative is BIST, which is intrinsically more secure. However not all the circuits are suited for BIST (e.g. microprocessors) and BIST provides just a pass/fail signature which is not useful for diagnosis. Many countermeasures have been proposed in the literature [3][4], however, each of them have their limitations and there is no full-proof mechanism to deal with this leakage through the scan chains.

One of the attacks proposed in the literature, concerns the RSA algorithm [5]. However it supposes that the design has a single scan chain. Unfortunately, this assumption is not realistic, since more complex DfT methods are required for meeting the design requirements and reducing the test cost. Techniques such as multiple scan chains, pattern decompression [6], response compaction [7] and filters to increase the tolerance to unknowns [8] are commonly inserted in the test infrastructure. These structures are supposed to behave as countermeasures against scan attacks, due to the apparent reduction on the observability of internal states, as proposed in [9].

In this paper we propose a new attack on RSA that works even in the presence of advanced DfT methods. We describe all the issues on carrying out the attack, and how to overcome them. Additionally, we prove its feasibility by actually performing the attack on a RSA design. Moreover, the attack may be applied without knowledge of the DfT structures, which makes the attack more realistic.

The outline of the paper is as follows. In section 2, we present the previous work performed in the field of scan-attacks on symmetric and public-key ciphers and some proposed countermeasures. The RSA scan-attack itself is described in section 3. Then in section 4, we describe how we deal with the practical aspects of performing the attack. The experimental results containing a discussion about the applicability of the scan attack in the presence of industrial DfT methods and known scan-attack countermeasures is presented in section 5. A comparison with the previous RSA scan-attack is given in section 6. Finally, we conclude the paper with plans for future work in section 7.

2 Previous Work

The first scan attack proposed in the literature [1] was conceived to break a Data Encryption Standard (DES) cipher. Karri et al. described a two phase procedure which consists in first finding the position of the intermediary registers on the scan chain,

and then retrieving the DES first round key by applying only 3 chosen plaintexts. Later the same authors proposed [2] an attack on the Advanced Encryption Standard (AES). This one was based on the differential method, which analyses the differences of scan contents instead of the direct value itself. By using this method, the preliminary step of identifying the position of the intermediary registers is no longer required. Advances were also made on proving that public-key implementations are susceptible to scan attacks. RSA and Elliptic Curve Cryptography (ECC) keys are retrieved by methods described in [5] and [10] respectively. Besides, some scan-attacks were also proposed for stream ciphers, for example [8].

Binary exponentiation algorithm is used as the target algorithm for the RSA scan-attack in [5], while the Montgomery Powering Ladder is used for the ECC attack in [10]. Both the attack methods are based on observing the values of the intermediate register of interest on the scan chain for each bit of the secret key (decryption exponent for RSA, and scalar multiplier for ECC), and then correlating this value with a previous offline calculation, which the authors refer to as 'discriminator'. If the value matches with this discriminator value, a corresponding decision is taken on the key bit.

In order to secure the test structures, several countermeasures have been proposed. They may be classified in three different groups: (1) methods to control the access to the test facilities through the use of secure test wrappers [12]; (2) methods to detect unauthorized scan operations [10] as probing and other invasive attacks; (3) methods that provide confusion of the stream shifted out from the scan outputs [14]. Additionally, it was suggested in [9] that advanced industrial DfT methods such as response compression are enough to impede any attack. However, advanced attacks [15][16] have been conceived to deal with those methods.

3 Principles of RSA Attack

3.1 RSA

The Rivest-Shamir-Adleman (RSA) algorithm is a widely used public-key cryptographic algorithm, employed in a wide range of key-exchange protocols, such as the popular Diffie-Hellman scheme. A brief description of the RSA algorithm is presented below:

Algorithm 1: RSA Key generation

- Random primes p and q
- $N = p*q$ (1024 bit)
- e = random co-prime to $\varphi(N)=(p-1)*(q-1)$
- $d = e^{-1} \bmod \varphi(N)$

Algorithm 2: RSA Encryption & Decryption

- Ciphertext $c = m^e \bmod N$
- Decrypted plaintext $m = c^d \bmod N$

Both the above operations are large number modular exponentiations.

When RSA is implemented in hardware, there are various possible options and many algorithms are available. Montgomery Exponentiation method is most often used, owing to its efficient hardware implementation, as it does away with the expensive division operation required for modular multiplications involved in an exponentiation. Hence we choose the Montgomery method as the target for our scan-chain attack.

The Montgomery product of two n-bit numbers A with B is denoted by:

$A * B = A . B . R^{-1} \bmod N$,

where '.' denotes a modular multiplication, N is the modulus or prime number in the modular multiplications, and $R = 2^n$, with n being the number of bits of the RSA algorithm used. In this case study, we are using 1024-bit RSA.

The algorithm for a Montgomery Exponentiation used in RSA can be presented as follows [17]:

Algorithm 3: Montgomery exponentiation

INPUT: Prime $m = (m_{l-1} \ldots m_0)_b$, $R = b^l$, exponent $e = (e_t \ldots e_0)_2$ with $e_t = 1$, and an integer x, $1 \leq x < m$ (l is the number of bits in the prime number, 1024 in our case, b is the base, which is 2 for binary).

OUTPUT: $x^e \bmod m$.

1. xtilde ← Mont(x, $R^2 \bmod m$), A ← R mod m. (R mod m and R^2 mod m may be provided as inputs.)
2. For i from t down to 0 do the following:
   (a) A ← Mont(A, A).
   (b) If $e_i = 1$, then A ← Mont(A, xtilde).
3. A ← Mont(A, 1).
4. Return (A).

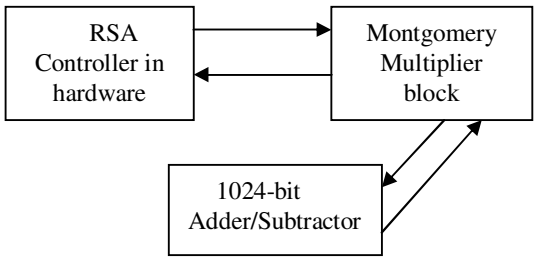
Mont (A, A) is known as the squaring (S) operation, while the Mont(A, xtilde) is known as the Multiplication operation (M) for Montgomery Exponentiation. The square and multiply operations are actually modular multiplications implemented using the Montgomery multiplication algorithm [17]. Each iteration of the loop within the algorithm consists either of a squaring and multiply operations if the key bit is 1, or only a squaring operation if the key bit is 0.

In our proposed scan-based attack, we are focusing on the intermediary register (A, in the algorithm above) which stores the value after each Montgomery multiplication. Irrespective of how the RSA modular exponentiation is implemented, the intermediate value will always be stored in a register. For instance, we may have a hardware/software co-design for the RSA crypto-processor, where the Montgomery multiplier is implemented as a co-processor in hardware (for efficiency) and the control logic or the algorithm for the Montgomery exponentiation implemented in software on a microcontroller. In this case, the results of the intermediate Montgomery operations may be stored in an external RAM, but this value needs to be transferred and stored in the registers inside the Montgomery multiplier datapath to allow the module to perform the computations correctly.

3.2 Target RSA Hardware Implementation

We have made a hierarchical 1024-bit RSA hardware implementation (employing Montgomery Exponentiation algorithm), which is the target of our proposed scan-attack. It consists of an adder/subtractor arithmetic module, a Montgomery multiplier block, and a RSA controller datapath for controlling the square and multiply operations involved in the exponentiation. This is shown in the block diagram below.

Gezel Hardware software co-design environment [18] was used to create the design, it was transformed into VHDL using the fdlvhd VHDL converter tool of Gezel, and finally Synopsys Design Compiler v2009.06 was used to convert the VHDL file into a gate-level netlist. Our implementation does not consider protection against Simple Power Analysis (SPA), Differential Power Analysis (DPA) and Fault Attacks, but test compression techniques supposedly acting as scan-attack countermeasures have been included.



3.3 Assumptions of scan attacks

The leakage analysis as well as the attack methods implemented by this tool lies on some assumptions:

- the cipher algorithm is known as well as the timing diagrams. The designer in charge of checking scan attack immunity should have this information;
- the scan chain structure is not known by the attacker. The scan length, as the number of internal chains and the order of the scan flip-flops are also supposed to be hidden. Although the input/output test pins (interface) are controllable;
- it is possible to control the test enable pin and then switch from mission mode to test mode, which allows the cipher operation to be "stopped" at any moment;
- it is possible to control the input plaintexts (e.g. a design primary input) and to observe the values related to the intermediate states by means of scan out;

It is important to notice that all these assumptions are shared among all the scan attacks proposed in the literature. Additionally, these assumptions are fulfilled by majority of the test scenarios due to the fact that high testability is achieved by controlling and observing a huge number of design internal nodes.

3.4 Attack basics: The Differential Mode

One of the main advantages of the attack proposed in our paper over the previous RSA attacks is the fact that it works in the presence of industrial DfT structures. For that purpose, the differential mode [2], [16] is used to deal with linear response compactors which are inserted by majority of the DfT tools. Without compaction, the values stored in the SFFs are directly observable at the test output while they are shifted out. On the other hand, in the presence of compaction, each bit at the test output depends on multiple SFFs. In the case of parity compactors, each output bit is the XOR operation between the scan flip-flops on the same "slice". It means that the actual value stored in one SFF is not directly observable. Instead, if it differs from the value expected, the parity of the whole slice also differs, and so faults may be detected. This difference may also be exploited by an attacker.

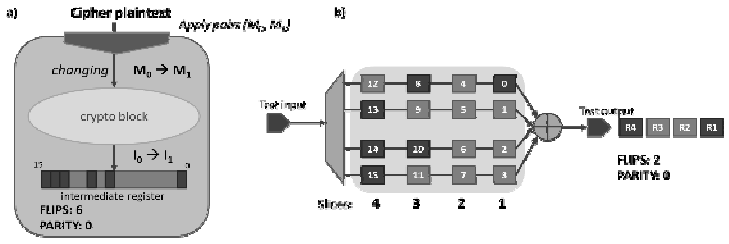


Fig. 1. a. Design with crypto block. b. example of DfT scheme

Fig. 1.a shows a crypto block, its cipher plaintext, and the intermediate register which is usually the target of the scan attack. The rest of the circuit will be omitted for didactic reasons. The differential mode consists of applying pairs of plaintexts, in this example denoted by (M0, M1). The circuit is first reset and the message M0 is loaded. Then after N clock cycles the circuit is halted and the intermediate register I0 is shifted out. The same procedure is repeated for the message M1 for which I1 is obtained. Let's suppose that I0 differs from I1 in 6 bit positions as shown in 1.a, where a bit flip is represented by a darker box. Let's also suppose that the intermediate register contains only 16 bits and the bits 0, 8, 10, 13, 14, and 15 are flipping. The parity of the differences is equal to 0, since there is an even number of bit flips.

In Fig. 1.b, the flip-flops of the intermediary register are inserted as an example of DfT scenario with response compaction. In this case there are four scan chains divided in four slices. RX represents the test output corresponding to the slice X. As it may be seen, if only the bit 0 flips in the first slice (an odd number) this difference is reflected into a flip of R1. In slice 2, no bits flip and thus R2 remains the same. Two flips occur in slice 3: 8 and 10. In this case, both flips mask each other, thus 2 flips (even) result in 0 flips at the output R3. In slice 4, 3 bit flips are sensed as a bit flip in R4.

The parity of flips in the intermediate register is equal to the parity of flips at the output of the response compactor. This comes from a basic property of this kind of response compactors: the parity of differences measured in the test output is equal to the parity of differences in the intermediate register.

This property is valid for any possible configuration of the scan chains (number of scans versus slices). Additionally it is also valid for compactors with multiple outputs. In this case, the difference measured should consider all compactor outputs. Thus using the differential mode, the attacker observes differences in the intermediate register and then retrieves the secret key. Complex scenarios with other FFs of the circuit are shown in Section 4.

3.5 Description of the Attack

As presented in sub-section 3.1, the Montgomery exponentiation consists of repeating the Montgomery multiplication operations several times. The first multiplication in the main loop, i.e., the squaring of A, is always performed independently of the value of the secret key bit. The second multiplication, A times xtilde, is performed only if the decryption key bit is 1. The main idea of the attack proposed here is to check if the second operation is executed or not, by observing the value of A afterwards. If it does, then the key bit is 1, otherwise it is 0. This procedure is repeated for the whole key (1024 or 2048 bits).

In order to detect if the second multiplication was executed, the attacker must scan out the value of A after each loop (timing issues detailed in Section 4). Additionally,

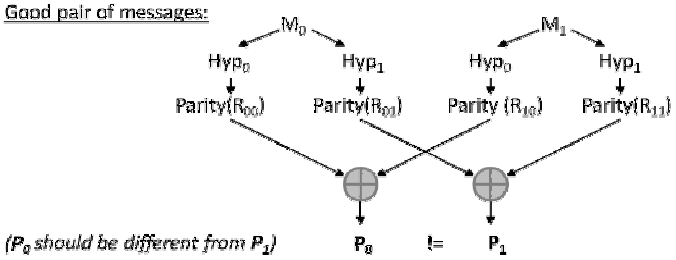


Fig. 2. Hypothesis Decision

as explained in the previous sub-section, a pair of plaintexts is used to overcome the obscurity provided by the response compactor. This pair must be properly chosen so that a difference on the parity of A would lead to the decryption bit. For that, it is important that we give a pair of specific message inputs to the algorithm. The process to derive these 'good' pairs of messages is as follows:

First, a pair of random 1024-bit messages is generated using a software pseudo-random number generator. We denote them here as (M_0, M_1). Then, the corresponding output responses (after one iteration of the exponentiation algorithm) are computed on each of these messages assuming the key bit to be both '0' and '1'. Let (R_{00}, R_{01}, R_{10}, R_{11}) be the responses for message M_0 and M_1 for key bit '0' and '1' respectively. Let Parity(R_{00}), Parity(R_{01}), Parity(R_{10}) and Parity(R_{11}) be the corresponding parities on these responses. Let P_0 be equal to Parity(R_{00}) XOR Parity(R_{10}) and P_1 be equal to Parity(R_{01}) XOR Parity(R_{11}). If P_0 != P_1, then the messages are taken to be useful,

otherwise they are rejected and the process is repeated till a pair of 'good' messages is obtained.

After a good pair of messages is found, it may be applied to the actual circuit. For both pairs of elements, the application is executed in mission mode for the number of clock cycles corresponding to the targeted step (decryption key bit). For these pairs of elements, the scan contents are shifted out and the parity of the difference at the test output bitstream is measured. If the parity of differences is equal to P_0, then the hypothesis 0 is correct and the secret key bit is 0. If it is equal to P_1, then the secret key bit is 1. This procedure is repeated for all the bits of the decryption key.

4 Practical aspects of the Attack

Performing scan attacks on actual designs requires additional procedures which have not been taken into consideration by some previous attacks proposed in the literature. The two main practical issues consist of (1) dealing with the other flip-flops of the design; (2) finding out the exact time to halt the mission mode execution and to shift out the internal contents. The first issue is solved by analyzing the leakage of the FFs of the intermediate register at the test output (described in sub-section 4.1). The second issue is described in sub-section 4.2.

4.1 Leakage analysis

The scenario of Fig. 1 is commonly taken into consideration by scan attacks, however in real designs other FFs of the design will be included in the scan chain. These additional FFs may complicate the attack if no workaround is taken into account. Fig. 2.a shows a design containing three types of FF. We define here three types of scan flip-flops (SFFs), depending on the value they store, as shown in Fig. 2.a. T1 SFFs correspond to the other IPs in the design, that store data not dependent on the secret. T2 SFFs belong to the registers directly related to the intermediate register, that store information related to the secret key and that are usually targeted by attackers (e.g. AES round-register). T3 SFFs store data related to the cipher but not the intermediate registers themselves (such as input/output buffers or other cipher registers). The leakage, if it exists, concerns the T2 type.

The goals of the leakage analysis is to find out if a particular bit of the intermediate register (T2) can be observed at the test output, and locate which output bit is related to it. Thus the analysis is focused on one bit per time, looking for an eventual bit flip in T2. In order to do that, the pair (M_0, M_1) is chosen so that the value on $T2_N$ for M_0 differs by a single bit from the value $T2_N$ for M_1. Denoting $T2_N$ as the value stored in T2 after N clock cycles while the design is running in mission mode from the plaintext M_0 (the first event in mission mode is a reset). In Fig. 2.a the darker blocks represent a bit that flips. Thus, in this case, the least significant bit of $T2_N$ flips. Since the attack tries to verify if it is possible to observe a flip in the LSB of $T2_N$, it is ideal that there is no flip in $T1_N$. To reduce the effect of the T1 flip-flops, all the inputs that are not related to the cipher plaintext are kept constant. It means that $T1_N$ for M0 has the same value of $T1_N$ for M_1. However, the same method cannot be applied to reduce the

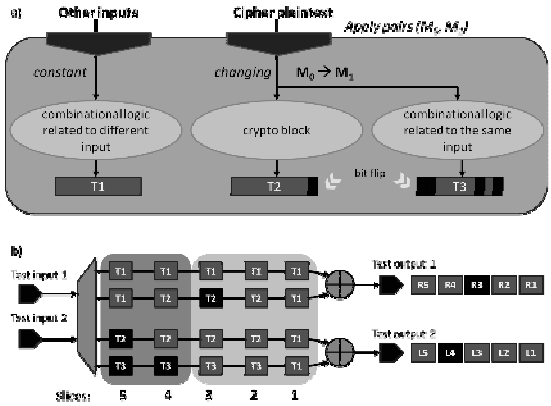


Fig. 3. a. Design illustrating the categories of FFs b. DfT scheme

effects of T3. Since we suppose that the logic associated with T3 is unknown and since its inputs are changing, the value $T3_N$ for M_0 may differ from $T3_N$ for M_1. In our example, let us flip only three bits of T3.

Figure 2.b shows the result of these bit flips in the scan chain and consequently in the test outputs. For didactic reasons, we suppose that the DfT insertion created 4 scan chains, and placed a pattern decompressor at the input and a response compressor with two outputs (R and L). As it may be seen, the slice 1 contains only T1 scan flip-flops, meaning that after the response compressor, the values of R1 and L1 are not supposed to flip (because $T1_N$ has the same value for M_0 and M_1). For slice 2, the same happens. Slice 3 contains the only flipping bit of $T2_N$ and the other flip-flops in the slice do not change. In this case, the bit flip of the first bit of $T2_N$ is observable in R3. It means that an attacker could exploit the information contained in R3 to find the secret key. Hence, this is considered a security leakage and may be exploited by the attack described in Section 3.

Slice 4 and slice 5 contain flip-flop configurations that may complicate an attack. For instance in slice 4, there are FFs of T1 and T2 that are not affected by a change

from M_0 to M_1. However it contains one FF affected in T3. It implies that the L4 value flips, which may confuse the attacker (he expects a single bit flip caused by the LSB of T2). In this case, the attacker is able to identify that the value of L4 is dependent on the plaintext, but is not able to exploit this information, since the T3 related logic is supposed to be unknown. Another complication is shown in the configuration of slice 5. If the LSB of T2 is actually on the same slice as a flipping SFF of T3, the flip is masked and no change is observed in L5. In this case, the attacker is not able to exploit that bit

Next, the attacker repeats this method for each bit of the intermediary register (e.g., 1024 times for RSA). If it detected some useful leakage (like R3), he proceeds with the attack method explained in the Section 3.

4.2 Timing aspects

The scan-based attack on RSA is targeted at finding the decryption key (may be 1024

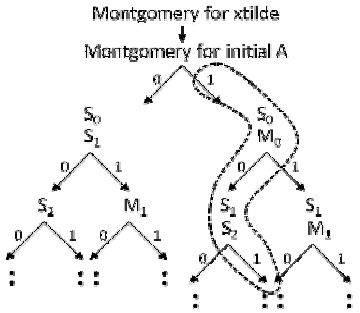


Fig. 4. Timing Estimation Tree

or 2048 bits long). It is very important to find the exact time to scan out the contents of the intermediate registers using the scan chains. The integral timing aspects for the attack are presented pictorially in Figure 4.

Since the same hardware is commonly used for both encryption and decryption in RSA, we can run the hardware with a known encryption key in order to get the timing estimations. For instance, the attacker must find out the number of clock cycles that a Montgomery multiplication operation takes. With a known key, we know the number of Montgomery multiplications required for the square and multiply operations of the RSA modular exponentiation (Algorithm 1). Dividing the total time of execution for this encryption by the number of operations gives the approximate time required for one Montgomery operation. Then using repeated trial-and-error steps of the comparing the actual output with the expected result after one Montgomery (presented in Section 3), it may be possible to find out the exact number of clock cycles required.

This timing is utilized in our attack during the decryption process to find out the decryption exponent. The RSA in hardware is run in functional mode for the exact number of cycles needed to execute a predetermined number of Montgomery opera-

tions. Then the hardware is reset, scan enable is made high and the scan-chain contents are taken out. Depending on whether the key bit was 0 or 1, either a squaring (S) is performed or both square (S) and multiply (M) are performed respectively. In our proposed attack, we always run the software implementation for two Montgomery cycles taking the key bit as 0 and 1 (two hypothesis in parallel). If the first bit was 1, both square (S0) and multiply (M0) operations are performed, otherwise two squarings (S0 & S1) are performed. Then the actual result from the scan-out of the hardware implementation after each key bit execution is checked with the results of the simulation in software. If it matches with the first result (of S0 and M0), then the key bit is 1, otherwise the key bit is 0. Now, for the next step starting with the right key bit, again the decryption is performed in software assuming both 0 and 1 possibilities. This time we run for one or two Montgomery cycles depending on whether the previous key bit was 0 or 1 respectively. If the previous key bit was 0, then Squaring on the next key bit (S2) is performed for key bit 0 and a Multiply on the same key bit is performed (M1) for present key bit 1. On the other hand, if the previous key bit was 1, then Squaring on the same (S1) and next key bit (S2) is performed for present key bit 0 or a Square (S1) and Multiply (M1) on the same key bit is performed (M1). The results are compared with the actual result from the scan-out of the hardware implementation, and the corresponding decision taken. The process is repeated in this way until all the decryption key bits are obtained.

As an example, if the decryption key bits were 101…, the timing decision tree would follow the path denoted within the dotted lines in the figure (S0, M0, S1, S2, M2,…).

5 Attack Tool

In order to apply the attack to actual designs, we developed an attack tool. The main goal of this tool is to apply the attack method proposed in Section 3, as well as the leakage analysis proposed in Section 4, to many different DfT configurations, without modifying the attack.

The scan analysis tool is divided in three main parts: the attack methods and ciphers (implemented in C++), the main controller (Perl), and the simulation part which is composed by a RTL deck and ModelSIM, as it may be seen in Figure 1. In order to use the tool the gate-level netlist must be provided by correctly setting the path for both the netlist and technology files for ModelSIM simulations. Then the design is linked to the RTL deck, which is used as an interface with the tool logic. This connection is automatically done by giving the list of input and output data test pins, as well as the control clock, reset, and test enable pins. Additionally, other inputs such as plaintext and ciphertext must be set in the configuration file.

Once the DUT is linked, the tool may simulate it by calling ModelSIM SE with the values established by the main controller. This interface is achieved by setting environment variables in the Perl script which are read by the ModelSIM Tcl script and then passed on to the RTL deck via generics. For instance, the information being exchanged here is the plaintext (cipher specific), reset and scan enable timing (when to

scan and how long) and the value of the scan input (test specific). In return, the scan output contents are stored in a file and they are processed by the main attack controller in order to run the attacks.

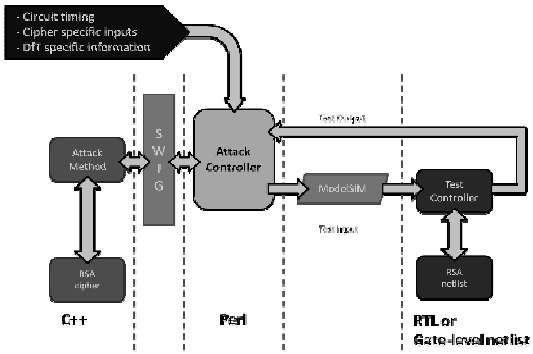


Fig. 5. High-level block diagram of the Attack tool

On the left side of Figure 5, the software part is shown (attack method and cipher description). The new RSA attack is implemented in C++ based on a RSA cipher implemented in the same language. We previewed new attacks against other ciphers, e.g. ECC. Scan-attacks on other similar cryptosystems may be conceived since the tool was built in such a way that adding a new cipher is straight-forward.

The core of the tool is implemented by the attack controller (Perl) which calls the attack method (using a SWIG interface). The attack controller ensures that the settings are initialized and then it launches both the attack and the simulation. As a secondary functionality, the controller handles some design aspects, like timing and multiple test outputs, so that the attack method itself may abstract that information. For instance, the attack method has no information on how many clock cycles it takes to execute a Montgomery multiplication. Also, it finds out the number of scan cycles that the shift operation must be enabled so all the scan length is unloaded.

6 Experimental Results

In order to test the effectiveness of the attack, we implemented a 1024 bits RSA algorithm in hardware with separate datapaths for the Montgomery multiplier, adder/subtractor block and the main controller for the Montgomery exponentiation. Then we envisaged different scenarios to test the attack flexibility. The first scenario is a single chain containing all the FFs of the design. Then, in the next subsection, we used Synopsys DfT Compiler (v2010.12) to insert more complex configurations such as decompression/compaction. Finally, in the last subsection, we implemented some countermeasures proposed in the literature to verify if the attack is able to overcome them. All the runs were performed on a 4 GB Intel Xeon CPU X5460 with four processors.

The total number of FFs in the design is 9260. Out of these, 4500 belong to the T1 type, 1024 consist of the intermediate register (T2 type) and 4096 belong to the T3 type (see Section 3). Therefore using Synopsys DfT Compiler we inserted a single chain with all these FFs, and the design was linked with the tool. Then the leakage analysis was run over this configuration. For identifying each bit of the RSA intermediate register (1024-bit long), the attack tool takes approximately 3.5 minutes per bit. Then the tool proceeds with the attack method, in order to find the secret key. In this phase, the tool takes again approximately 3.5 min per bit of secret key. Both the timing for the leakage analysis and the attack are strongly dependent on the server configuration. Additionally, the C++ code takes approximately 5 seconds from the 3.5 minutes, meaning that the simulation limits the execution time.

For our test case, we required around 11 messages to find out the full 1024-bit RSA exponent. This number is less than that required for the attack presented in [5] (which takes around 30 messages).

6.1 In presence of DfT Methods

In order to test our scan-attack in the presence of industrial DfT methods, Synopsys DFT Compiler was used to insert different DfT configurations in the RSA circuit. In the first case, 120 scan chains were inserted, without compaction/compression. Since the tool analyzes each scan output pin separately and independently, and since the sensitive registers were converted to scan FFs, the attack with the tool was able to find out the secret key. Changing the position of the sensitive FFs do not change the result. The time taken to retrieve the key in this case is almost the same as that of the previous case (with a single chain).

In a second scenario, pattern compaction and response compression were inserted. Different compression ratios were tested, but as proposed in [16], linear response compactors do not lead to any increase in the security. Since the test inputs are not used in the pattern compactor (the plaintext is a primary input), it does not affect the attack method and hence it is not taken into consideration. As the proposed methods are all based on the differential mode, the linear compressors do not impede the attack and also it does not imply significant increase in simulation time.

As a last scenario, the X-tolerant options were activated to add the masking logic that deals with the unknowns present in the design. The masking blocks some scan chains at the instant while the contents are shifted out if the test engineer believes that there is an X that may corrupt the test output. This mask is controlled by the output of the pattern decompressor, which is controller then by the test inputs. Since the mask is controllable, it is just a matter of shifting in the right pattern which does not mask the confidential data. Thus the masking can set when the sensitive data is shifted out. Hence, our proposed scan-attach still works in the presence of masking.

6.2 In presence of proposed countermeasures

In presence of inverters. One of the countermeasures proposed in the literature is the insertion of dummy inverters before some FFs of the scan chain [16]. This technique

aims at confusing the hacker, since the sensitive data observed at the scan chain may be inverted. However, since these inverters are placed always at the same location in the scan chain, they are completely transparent to the differential mode.

The effectiveness of the attack against this countermeasure was validated on the RSA design containing multiple scan chains and compaction/compression module. Two implementations were considered with 4630 and 6180 inverters (50% and 75 % of the overall 9260 FFs in the design respectively) randomly inserted in the scan chains. For both cases, the tool was able to find leakage points and then to retrieve the secret key.

In presence of partial scan. Depending on the design, not all the flip-flops need to be inserted in the scan chain in order to achieve high testability. As proposed in [4], partial scan may be used for increasing the security of a RSA design against scan attacks. However, the authors suppose that the attacker needs the whole sensitive register to retrieve the secret key. As it was described in Section 3, the leakage analysis feature can be used to find out which bits of the sensitive register are inserted in the scan chain. Once these bits are identified, the attack can proceed with only partial information, since each bit of the sensitive register is related to the key.

For evaluating the strength of the partial scan, we configured the DfT tool in such a way so as to not to insert some of the sensitive registers in the scan-chain. In the first case, half of the sensitive flip-flops were inserted in the chain. The tool was able to correctly identify all the leaking bits and then to retrieve the secret key. Also in the worst case situation, i.e., where only one secret bit was inserted in the chain, the tool was still able to find out the correct secret key.

7 Comparison with previous RSA attacks

The approach taken in [4] is for a pure software attack which does not take into account the practical aspects of applying it to an actual cryptographic hardware implementation. The timing aspects are crucial to scan attacks on secure hardware, which has been addressed in this paper. Our scan-attack analysis tool integrates the actual hardware (in the form of a gate-level netlist with inserted DFT) with the software emulation which allows us to perform the attack in real-time. The secret decryption exponent key bits are deciphered on-the-fly using this combined approach.

Left-to-right binary exponentiation (employed in ordinary exponentiation) is used as the target RSA algorithm for the attack in [4]. This is generally not implemented in hardware owing to the expensive division operation involved in modular operations. We target the Montgomery Exponentiation algorithm, which is by far the most popular and efficient implementation of RSA in hardware, as there are no division operations involved (owing to performing the squaring and multiply operations in the Montgomery domain).

Moreover, an inherent assumption in the attack in [4] is that there are no other exponent key-bit dependent intermediate registers which change their value after each square and multiply operation. This may not be the practical case in an actual hard-

ware implementation, where multiple registers are key dependent and change their values together with the intermediate register of interest in the attack (for instance, input and output buffers). These registers may mask the contents of the target intermediate register after XOR-tree compaction (as shown in the leakage analysis in Section 3). Our proposed scan-attack analysis takes the contents of other key-dependent registers present in the scan chain, and presents ways to deal with this problem.

Finally, the attack in [4] cannot be applied to secure designs having test response compaction and masking (which is usually employed in DfT for all industrial circuits to reduce the test volume and cost). Our scan-attack analysis, on the other hand, works in the presence of these scan compression DfT structures.

8 Conclusion

In this paper, we have presented a new scan-based attack on RSA cryptosystems. A scan-chain leakage analysis for the algorithm is presented, along with the practical aspects of mounting the attack on an actual hardware implementation of RSA. A comparison with the previous RSA scan-attack proposal is also made. We present a scan-chain leakage analysis tool and explain its use through the RSA attack. State-of-the-art scan attack countermeasures and industrial test compression techniques, supposed to behave as countermeasures are also evaluated for scan-leakage strength using RSA as a case study. We successfully attacked the RSA implementation in the presence of these countermeasures.

As future work, we plan to extend the scope of this scan-based attack on RSA to El Gamal and other similar public-key implementations based on large number modular exponentiations. We will also extend the scope of our proposed attack on RSA implementations with SPA and DPA countermeasures. This can also be an interesting topic for future contributions in this domain.

Acknowledgement

This work has been supported in part by the European Commission under grant agreement ICT-2007-238811 UNIQUE and in part by the IAP Programme P6/26 BCRYPT of the Belgian State. Amitabh Das is funded by a fellowship from Erasmus Mundus External Cooperation Window Lot 15.

References

1. B. Yang, K. Wu, and R. Karri. Scan Based Side Channel Attack on Dedicated Hardware Implementations of Data Encryption Standard. Proceedings IEEE International Test Conference (ITC) 2004.
2. B. Yang, K. Wu, and R. Karri. Secure Scan: A Design-for-Test Architecture for Crypto Chips. Proceedings ACM/IEEE Design Automation Conference (DAC), June 2005, pp. 135–140.

3. G. Sengar, D. Mukhopadhayay, D. Chowdhury. An Efficient Approach to Develop Secure Scan Tree for Crypto-Hardware, 15th International Conference on Advanced Computing and Communications.
4. M. Inoue, T. Yoneda, M. Hasegawa, and H. Fujiwara. Partial Scan Approach for Secret Information Protection. European Test Symposium, 2009, pp.143-148.
5. Ryuta Nara, Kei Satoh, Masao Yanagisawa, Tatsuo Ohtsuki, Nozomu Togawa. Scan-Based Side-Channel Attack Against RSA Cryptosystems Using Scan Signatures. IEICE Transaction Fundamentals, Vol. E93-A, No. 12, December 2010, Special Section on VLSI Design and CAD Algorithms.
6. Wang, L.-T.; Xiaoqing Wen; Furukawa, H.; Fei-Sheng Hsu; Shyh-Horng Lin; Sen-Wei Tsai; Abdel-Hafez, K.S.; Shianling Wu. VirtualScan: a new compressed scan technology for test cost reduction. Test Conference, 2004. Proceedings. ITC 2004. International , pp. 916- 925, 26-28 Oct. 2004
7. Rajski, J.; Tyszer, J.; Kassab, M.; Mukherjee, N. Embedded deterministic test. Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on , vol.23, no.5, pp. 776- 792, May 2004
8. Mitra S.; Kee Sup Kim. X-compact: an efficient response compaction technique for test cost reduction. Proc. ITC 2002, pp. 311- 320.
9. C. Liu, Y. Huang. Effects of Embedded Decompression and Compaction Architectures on Side-Channel Attack Resistance. 25th IEEE VLSI Test Symposium (VTS) 2007.
10. R. Nara, N. Togawa, M. Yanagisawa, T. Ohtsuki. Scan-Based Attack against Elliptic Curve Cryptosystems, Asia South-Pacific Design Automatic Conference (ASPDAC) 2010.
11. Y. Liu, K. Wu, and R. Karri. Scan-based Attacks on Linear Feedback Shift Register Based Stream Ciphers. ACM Transactions on Design Automation of Electronic Systems (TODAES) 2011.
12. A. Das, M. Knezevic, S. Seys, and I. Verbauwhede. Challenge-response based secure test wrapper for testing cryptographic circuits. IEEE European Test Symposium (ETS) 2011.
13. D. Hély, M. Flottes, F. Bancel, B. Rouzeyre, N. Berard, M. Renovell. Scan Design and Secure Chip. 10th IEEE International On-Line Testing Symposium (IOLTS'04).
14. D. Hély, F. Bancel, M. Flottes, B. Rouzeyre. Test Control for Secure Scan Designs. European Test Symposium (ETS'05).
15. J. Da Rolt, G. Di Natale, M. Flottes, B. Rouzeyre. New security threats against chips containing scan chain structures. Hardware Oriented Security and Trust (HOST) 2011.
16. J. Da Rolt, G. Di Natale, M. Flottes, B. Rouzeyre. Scan attacks and countermeasures in presence of scan response compactors. 16th IEEE European Test Symposium (ETS) 2011.
17. A. Menezes, P. van Oorschot, and S. Vanstone. Chapter 14. Efficient Implementations. Handbook of Applied Cryptography. CRC Press, 1996.
18. Gezel Hardware/Software Codesign Environment, http://rijndael.ece.vt.edu/gezel2/