Stealthy Dopant-Level Hardware Trojans ⋆

Georg T. Becker[1], Francesco Regazzoni[2], Christof Paar[1,3], and Wayne P. Burleson[1]

[1]University of Massachusetts Amherst, USA
[2]TU Delft, The Netherlands and ALaRI - University of Lugano, Switzerland
[3]Horst Görtz Institut for IT-Security, Ruhr-Universität Bochum, Germany

Abstract. In recent years, hardware Trojans have drawn the attention of governments and industry as well as the scientific community. One of the main concerns is that integrated circuits, e.g., for military or critical-infrastructure applications, could be maliciously manipulated during the manufacturing process, which often takes place abroad. However, since there have been no reported hardware Trojans in practice yet, little is known about how such a Trojan would look like, and how difficult it would be in practice to implement one.
In this paper we propose an extremely stealthy approach for implementing hardware Trojans below the gate level, and we evaluate their impact on the security of the target device. Instead of adding additional circuitry to the target design, we insert our hardware Trojans by changing the dopant polarity of existing transistors. Since the modified circuit appears legitimate on all wiring layers (including all metal and polysilicon), our family of Trojans is resistant to most detection techniques, including fine-grain optical inspection and checking against "golden chips". We demonstrate the effectiveness of our approach by inserting Trojans into two designs — a digital post-processing derived from Intel's cryptographically secure RNG design used in the Ivy Bridge processors and a side-channel resistant SBox implementation — and by exploring their detectability and their effects on security.

Keywords: Hardware Trojans, malicious hardware, layout modifications, Trojan side-channel

1 Introduction

Integrated circuits (ICs) are the heart of virtually all modern applications. This includes sensitive and safety critical devices, such as medical devices, automotive, industrial control systems, power management or military devices. Often circuit blocks in a single IC are designed by different parties, manufactured by

⋆ The authors would like to thank Mario Kirschbaum from TU Graz for his helpful comments in implemeting iMDPL. This work was supported in part by the NSF Grants 0916854, 0923313 and 0964641 and by the HHS Grant 90TR0003/01.

an external and possibly off-shore foundry, packaged by a separate company and supplied by an independent distributor.

This increased exploitation of out-sourcing and aggressive use of globalization in circuit manufacturing has given rise to several trust and security issues, as each of the parties involved potentially constitutes a security risk. In 2005 the Defense Science Board of the US Department of Defense published a report in which it publicly voiced its concern about US military reliance on ICs manufactured abroad [4]. One threat in this context is that malicious modifications, also referred to as hardware Trojans, could be introduced during manufacturing. All this raises the question of trust in the final chip, especially if chips for military or safety-critical civilian applications are involved. Even if chips are manufactured in a trusted fab, there is the risk that chips with hardware Trojans could be introduced into the supply chain. The discovery of counterfeit chips in industrial and military products over the last years has made this threat much more conceivable. For instance, in 2010 the chip broker VisionTech was charged with selling fake chips, many of which were destined for safety and security critical systems such as high-speed train breaks, hostile radar tracking in F-16 fighter jets, and ballistic missile control systems [6]. The threat of hardware Trojans is expected to only increase with time, especially with the recent concerns about cyberwar, cf., e.g., [13, 20].

Surprisingly, despite the major research efforts in the general area of hardware Trojans, little is known about how to built stealthy hardware Trojans at the layout level (post place&route). Contrary to the majority of past works, in this paper, we investigate a new family of Trojans that do not need any extra logic resources but merely require a *change in the dopant polarity of a few transistors*. Hence, these Trojans add zero overhead in terms of additional transistors and metal wires. We show that such a change will not be detected by several of the common Trojan testing methods, including optical inspection. A central question that arises is how such minuscule manipulations can result in changes to the target system which are meaningful to an attacker. We address this question using two case studies. First, we show an attack against a design derived from Intel's RNG design used in the Ivy Bridge processors, and second, a dopant Trojan that allows attacking a side-channel resistant SBox implementation. Since the hardware is usually the root of trust in a system, even small malicious modifications of the hardware can be devastating to system security.

1.1 Related work

Research efforts targeting hardware Trojans can be divided into two parts, one related to the design and the implementation of hardware Trojans, and one addressing the problem of detecting hardware Trojans. In this section we summarize some contributions from both areas.

Hardware Trojan designs There have been relatively few research reports addressing the question of creating (as opposed to defeating) hardware Trojans,

with the first hardware Trojans published around 2008. Most proposed hardware Trojans consist of small to mid-size circuits which are added at the HDL level. For example, King et al. [10] presented a hardware Trojan inserted into a CPU that was capable of granting complete control of the system to an external attacker. The attacker can make arbitrary changes to the program code and can get unlimited access to the memory by simply sending a specific malicious UDP package to the processor. This Trojan shows how vulnerable systems can become once the root of trust — the hardware — is compromised. Another class of HDL-level Trojans are those which create a hidden side-channel to leak out secret keys by adding only a few additional gates [12]. Perhaps most of the Trojans proposed so far were shown at the annual hardware Trojan challenge hosted by NYU-Poly, where students insert hardware Trojans into a target FPGA design with the goal of overcoming hardware detection mechanisms [18].

All these Trojans have in common that they are inserted at the HDL level. The attack scenario here is that malicious circuitry is introduced into the design flow of the IC. However, these Trojans are difficult to realize by a malicious foundry which usually only has access to the layout masks. In this context, finding the needed space and adding extra connections to place & route the Trojan gates can be impractical. Furthermore, adding additional gates to the design after place & route can easily be detected using optical reverse-engineering. How realistic these Trojans are in a foundry-based attack model is therefore still unanswered.

A more realistic scenario for a foundry-based Trojan insertion are malicious modifications carried out at the layout level. An example of such a Trojan is the Trojan proposed by Shiyanovskii et.al. [21]. In this work the dopant concentration is changed in order to increase the effects of aging on the circuit, with the ultimate goal of reducing the expected lifetime of the device. However, these Trojans have limited usability, since it is hard to predict the exact time the ICs will fail and they can usually only serve as a denial-of-service type of Trojan.

Hardware Trojan detection Hardware Trojan detection mechanisms can be divided into post-manufacturing and pre-manufacturing detection mechanisms. The input to pre-manufacturing Trojan detection is usually the gate netlist or HDL description of the design under test. Pre-manufacturing Trojan detection tries to detect Trojans that have been inserted at the HDL level into the design flow, e.g. by third party IPs, design tools or untrusted employees. Usually the Trojan detection is based on functional testing or formal verification. There have also been proposals of how to defend against rather than detect hardware Trojans at the HDL level. One approach is to replace part of the hardware design that was not covered by functional testing with software [8]. Another approach is to add redundancy or a control circuitry between untrusted IPs that will make Trojan activation based on counters and inputs difficult [23]. However, these proposed Trojan detection and prevention mechanisms cannot prevent Trojans inserted at the sub-gate level, including the ones proposed in this paper.

Post-manufacturing Trojan detection mechanisms primarily attempt to detect Trojans inserted during manufacturing. They can be divided into two categories based on whether or not they need a "golden chip" (also referred to as golden model). A golden chip is a chip which is known to not include malicious modifications. The standard approach proposed to detect layout-level hardware Trojans and to find a golden chip is the use of optical reverse-engineering. The idea is to decap the suspected chip and make photos of each layer of the chip with e.g. a scanning electron microscope (SEM). These photos are then compared to the layout mask to detect additional metal or polysilicon wires. Additional metal wires and transistors can usually be detected very reliably. However, the overall process is expensive, time consuming and also destroys the chip under test. Hence, this method can only be used on a small number of chips. Also, optical reverse-engineering does not usually allow to detect changes made to the dopant, especially in small technologies. A dedicated setup could eventually allow to identify the dopant polarity. However, doing so in a large design comprising millions of transistors implemented with small technologies seems impractical and represents an interesting future research direction. We exploit this limitation to make our Trojans resistant against optical reverse-engineering.

A different approach to test for hardware Trojans without a golden chip is functional testing of the chip. Functional testing is standard procedure in the IC design flow and to some degree will always be performed. However, detecting Trojans is different from detecting manufacturing defects. Creating efficient test cases for hardware Trojan detection is difficult since the tester does not know how the Trojan gates look like. As a result, these Trojan gates are not taken into account during the test case generation which usually tries to optimize gate coverage. This leads to an inefficient functional testing procedure in contrast to functional testing at the netlist level, since in this case the Trojan gates will be part of the input to the test case algorithms.

Trojan detection mechanisms that require a golden chip are usually based on comparing side-channel information of the golden chip and the suspected chip. The most popular method is using the power side-channel for Trojan detection [1] but other side-channels such as time [11, 25], electro-magnetics(EM) and heat have been proposed as well. Typically these detection mechanisms can only detect Trojans that are at most three to four orders of magnitude smaller than the target design [1]. Small Trojans on the other hand are likely to stay undetected. Another approach to detect Trojans is to add specific Trojan detection circuitry into the design that can detect if the design was changed during manufacturing. For example, Rajendran et.al. [19] proposed to add additional gates that transform parts of the design into ring-oscillators. During testing, the frequencies of these ring-oscillators are compared with a golden chip to detect if the design was changed. These methods usually require a golden chip to determine the expected output of the detection circuitry, since circuit simulations are often not accurate enough. One big disadvantage of Trojan detection circuitry is that the circuitry itself can be subject to Trojan modifications. For similar reasons, the build-in-self-tests (BIST) that are employed in some designs to automatically

detect manufacturing and aging defects, are of limited use when applied to Trojan detection. This is not only due to the fact that a Trojan can be inserted into the BIST itself but also because the Trojan can be designed to not trigger the BIST, since BISTs are usually designed to only detect a sub-set of all possible errors.

1.2 Our goal and contribution

One of the major concerns are Trojans inserted during manufacturing e.g. by an untrusted foundry, but most of the published hardware Trojans are implemented at the HDL level. In this paper, we will therefore focus on Trojans inserted into designs at the layout level, after the place & route phase. We concentrate on constructing Trojans that can easily be added by a foundry and that defeat Trojan detection mechanisms. Especially, we propose layout-level hardware Trojans that can resist optical inspection, which is believed to be a reliable way to detect layout-level hardware Trojans. The proposed Trojans are inserted by modifying only the polarity of dopant in the active area and are therefore practically invisible to optical reverse-engineering. From a technical point of view, such modifications are certainly feasible in practice: A very similar approach is already used commercially for hardware-obfuscation in which optical reverse-engineering needs to be defeated as well [22]. By using two case studies, a side-channel resistant SBox implementation and an implementation of a secure digital random number post-processing design derived from Intel's new RNG used in the Ivy Bridge processors, we prove that the proposed dopant-based Trojans can be used efficiently in practice to compromise the security of the underlying target design. To the best of our knowledge, our dopant-based Trojans are the first proposed, implemented, tested, and evaluated layout-level hardware Trojans that can do more than act as denial-of-service Trojans based on aging effects.

The remainder of the paper is organized as followed. In the next section we will introduce the basic concept of our dopant-based Trojans. In Section 3, the first case study, a Trojan inserted into a design derived from Intel's new RNG design, is discussed. The second case study is presented in Section 4, showing how a side-channel resistant SBox implementation can be modified to establish a hidden side-channel using the dopant Trojans. In the last section the results are summarized and conclusions are drawn.

2 Dopant-Trojans

In this section an efficient way to design hardware Trojans without changing any metal or polysilicon layer of the target design is introduced. The main idea of the proposed Trojan is as follows: A gate of the original design is modified by applying a different dopant polarity to specific parts of the gate's active area. These modifications change the behavior of the target gate in a predictable way and are very similar to the technique used for code-obfuscation in some commercial designs [22]. Using a simple inverter as an example, we explain these dopant

modifications by changing the behavior of the target inverter gate in a way that it always outputs V_{DD}. However, the proposed techniques are sufficiently general to be applied to other types of gates in a similar way.

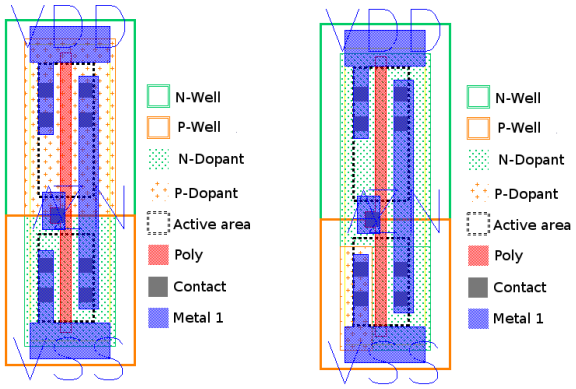
An inverter consists of a p-MOS and an n-MOS transistor whose drain contacts are connected via a metal layer as shown in Figure 1(a). The upper part of Figure 1(a) shows a p-MOS transistor. A p-MOS transistor consists of an n-well, the positively doped source and drain region and the gate region. The active area defines the area in which the dopant masks apply and hence also defines the source and drain area of the transistor. The polysilicon wire defines the gate area of the transistor[1].

To create an inverter Trojan that constantly outputs V_{DD}, the positively doped p-dopant mask of this p-MOS transistor is exchanged with the negatively doped n-dopant mask. Doping an active area within an n-well with n-dopant basically creates a connection to the n-well. N-wells are usually always connected to V_{DD} in a CMOS design. Since the n-dopant is applied to the entire active area of the p-MOS transistor, including the metal contacts, a direct connection from these contacts to the n-well is created. The upper part of Figure 1(b) shows the resulting p-MOS transistor Trojan. The source contact, which is connected to V_{DD}, has been transformed into an n-well tap, creating an additional connection from the n-well to V_{DD}. The drain contact is also connected to the n-well and thereby to V_{DD}. Hence, we have created a constant connection between V_{DD} and the drain contact without modifying the metal, polysilicon, n-well or active area. In the second step the connection between the n-MOS transistor's drain contact and GND is constantly disabled. This is achieved by applying p-dopant to the source contact of the n-MOS transistor while leaving the drain contact untouched. Applying p-dopant to the source contact of the n-MOS transistor transforms it into a well tap again and cuts of any connection between the source contact and the negatively doped source area of the n-MOS transistor. Therefore, the n-MOS transistor is no longer connected to GND regardless of its gate input. The resulting Trojan inverter can be seen in Figure 1(b). The metal, polysilicon, active and well layers are identical with the original inverter in Figure 1(a), but the Trojan gate always outputs V_{DD} regardless of its input.

Besides fixing the output of transistors to specific values, it is also possible to change the strength of transistors in a similar way. The strength of a transistor in CMOS is defined by its width. Usually the entire active area of a transistor is doped and therefore the width of a transistor is defined by the active area. However, by decreasing the area which is doped positively in a p-MOS transistor, it is possible to reduce the effective width of the transistor. Hence, to decrease the strength of a transistor it is sufficient to apply p-dopant to an area smaller than the active area of the transistor.

We want to stress that one of the major advantages of the proposed dopant Trojan is that they cannot be detected using optical reverse-engineering since we only modify the dopant masks. The introduced Trojans are similar to the

[1] The silicon area below the polysilicon wire is not subject to the dopant mask and hence remains the same polarity as the underlying well.



(a) Original (b) Trojan

Fig. 1. Figure of an unmodified inverter gate (a) and of a Trojan inverter gate with a constant output of V_{DD} (b).

commercially deployed code-obfuscation methods [22] which also uses different dopant polarity to prevent optical reverse-engineering. This suggests that our dopant Trojans are extremely stealthy as well as practically feasible.

3 Case-study 1: Intel's Ivy Bridge RNG

In this section we apply the concepts of our dopant Trojans to a meaningful, high-profile target to demonstrate the danger and practicability of the proposed Trojans. Our first target is a design based on Intel's new cryptographically secure RNG. Most prominently, it is used in the Ivy Bridge processors but will most likely be used in many more designs in the future. We chose this target because of its potential for real-world impact and because there is detailed information available about the design and especially the way it is tested [7, 9, 24].

The cryptographically secure RNG generates unpredictable 128-bit random numbers. The security has been verified by an independent security company [7] and is NIST SP800-90, FIPS 140-2, and ANSI X9.82 compliant. We will modify the digital post-processing of the design at the sub-transistor level to compromise the security of keys generated with this RNG. Our Trojan is capable of reducing the security of the produced random number from 128 bits to n bits, where n can be chosen. Despite these changes, the modified Trojan RNG passes not only the Built-In-Self-Test (BIST) but also generates random numbers that pass the NIST test suite for random numbers.

In the following section we first summarize the design of Intel's RNG and then discuss our malicious modifications.

3.1 Intel's TRNG design

Like most modern RNGs, Intel's RNG design consists of an entropy source (ES) and digital post-processing. The design also features a Built-In-Self-Test (BIST) unit that checks, at each power up, the correct functioning of the entropy source and the digital post-processing.

The ES is a metastable circuit based on two cross coupled inverters with adaptive feedback. The digital post-processing consists of a Online Health Test (OHT) unit and a cryptographically secure Deterministic Random Bit Generator (DRBG). The OHT monitors the random numbers from the entropy source to ensure that the random numbers have a minimum entropy.

The Deterministic Random Bit Generator itself consists of two parts, a conditioner and a rate matcher. The conditioner is used to compute new seeds for the rate matcher. Based on the current state, the rate matcher computes 128-bit random numbers. Reseeding is done whenever the conditioner has collected enough random numbers from the entropy source, or if at most 512 128-bit random numbers have been generated by the rate matcher. The conditioner as well as the rate-matcher are based on AES.

The rate matcher generates the 128-bit output r of the RNG and takes the seed (s, t) generated by the conditioner unit as input. The rate matcher has two internal state registers: a 128-bit register K and a 128-bit register c. During normal operation, the rate matcher generates 128 random bits r and updates the state registers in the following way (r, c, K)=Generate(c,K):

1. $c = c + 1$, $r = AES_K(c)$
2. $c = c + 1$, $x = AES_K(c)$
3. $c = c + 1$, $y = AES_K(c)$
4. $K = K \oplus x$
5. $c = c \oplus y$

Whenever the conditioner has a new seed, consisting of the 128-bit values s and t, available the internal states c and K are reseeded using the (c,K)=Reseed(s,t,c,K) function:

1. $c = c + 1$, $x = AES_K(c)$
2. $c = c + 1$, $y = AES_K(c)$
3. $K = K \oplus x \oplus s$,
4. $c = c \oplus y \oplus t$

Under low load, the rate matcher reseeds after each output of r. Under heavy load, the rate matcher generates several random numbers r before it reseeds, up to a maximum of 512. However, even under heavy load the rate matcher should reseed long before reaching its maximum of 512 [7].

3.2 Dopant-Trojan for Intel's DRBG

A 128-bit random number r generated by the rate matcher is the result of an AES encryption with an unknown 128-bit random input c and an unknown,

random key K. The attacker has a chance of $1/2^{128}$ to correctly guess a random number resulting in an attack complexity of 128-bits. The goal of our Trojan is to reduce the attack complexity to n bits, while being as stealthy as possible. This is achieved by cleverly applying our dopant-based Trojan idea described in Section 2 to internal flip-flops used in the rate matcher. In the first step we modify the internal flip-flops that store K in a way that K is set to a constant. In the second step the flip-flops storing c are modified in the same way, but n flip-flops of c are not manipulated. Hence, only $(128-n)$ flip-flops of c are set to a constant value. This has the effect that a 128-bit random number r depends only on n random bits and $128+(128-n)$ constant bits known to the Trojan designer. The owner of the Trojan can therefore predict a 128-bit random number r with a probability of $1/2^n$. This effectively reduces the attack complexity from 128-bit down to n bits. On the other hand, for an evaluator who does not know the Trojan constants, r looks random and legitimate since AES generates outputs with very good random properties, even if the inputs only differ in a few bits.

Our Trojan can be implemented by only modifying the flip-flops storing c and K, while all other parts of the target design remain untouched. Two different Trojan flip-flops are needed: one which sets the flip-flop output to a constant '1' and one which outputs a constant '0' regardless of the inputs. The DFFR_X1 flip-flop of the used Nangate Open Cell library [15] has two outputs, Q and its inverse QN. To implement our Trojan, the drain contact of the p-MOS transistor that generates signal Q is shortened to V_{DD} by applying n-dopant above the drain contact, as explained in Section 2. Simultaneously, the source contact of the n-MOS transistor for signal Q is disabled by applying p-dopant to the source contact. Hence, the output signal Q generates a constant output of V_{DD} regardless of its input. The inverse output QN is modified in the same way, only that this time the drain contact of the n-MOS transistor is shortened to GND and the source contact of the p-MOS transistor is disabled. This leads to a constant output of '0' for QN. The same modifications are used to generate a flip-flop Trojan to constantly provide an output of Q='0' and QN='1' by switching the roles of the n-MOS and p-MOS transistors. Note that only four of the 32 transistors of the DFFR_X1 flip-flop are modified as can be seen in Figure 2. But 28 transistors on the other hand stay untouched and therefore will still switch according to the input. This results in a smaller but still similar power consumption for a Trojan flip-flop compared to a Trojan-free flip-flop.

3.3 Defeating functional testing and statistical tests

It is a standard procedure to test each produced chip for manufacturing defects. In addition to these tests, the produced RNGs will also be tested against a range of statistical tests in order to be NIST SP800-90 and FIPS 140-2 compliance. Furthermore, to be compliant with FIPS 140-2, the RNG needs to be tested at each power-up to ensure that no aging effects have damaged the RNG. For this purpose Intel's RNG design includes a Built-In-Self-Test unit that checks the correct functioning of the RNG in two steps after each power-up. In the first step, the entropy source is disabled and replaced by a 32-bit LFSR that

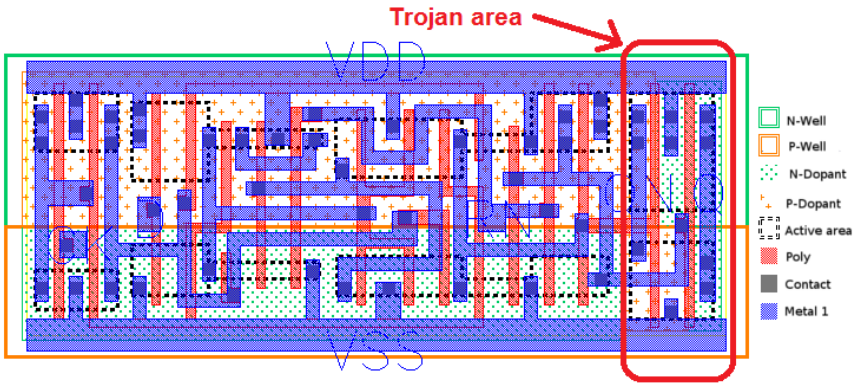


Fig. 2. Layout of the Trojan DFFR_X1 gate. The gate is only modified in the highlighted area by changing the dopant mask. The resulting Trojan gate has an output of $Q = V_{DD}$ and $QN = GND$.

produces a known stream of pseudo-random bits. The BIST uses this pseudo-random bit stream to verify the correct functioning of the OHT and feeds this bitstream to the conditioner and rate matcher. A 32-bit CRC checksum of the 4 x 128-bit output buffer that stores the last four outputs r_1,...,r_4 of the rate matcher is computed. This 32-bit CRC checksum is compared against a hard-coded value to verify the correct functioning of the conditioner and rate matcher. If the checksum matches, the RNG has passed the first part of the BIST. In the second part of the BIST the conditioner, rate matcher and output buffer are reset and the entropy source is connected again. The OHT tests the entropy of the entropy source and simultaneously seeds the conditioner and rate matcher. If the OHT signals the BIST that the entropy of the entropy source is high enough, the BIST is passed and the RNG can generate random numbers.

In [9] it is stated that "This BIST logic avoids the need for conventional on-chip test mechanisms (e.g., scan and JTAG) that could undermine the security of the DRNG." This fact is also mentioned in an Intel presentation in which it is argued that for security reasons the RNG circuitry should be free of scan chains and test ports [24]. Therefore, to prevent physical attacks, only the BIST should be used to detect manufacturing defects. From an attacker's point of view, this means that a hardware Trojan that passes the BIST will also pass functional testing. Although Intel's BIST is very good at detecting manufacturing and aging defects, it turns out that it cannot prevent our dopant Trojans. One simple approach to overcome the BIST would be to add a dopant Trojan into the BIST itself to constantly disable the error flag. However, it could be very suspicious if the BIST never reports any manufacturing defects.

To pass the BIST, the Trojan rate matcher needs to generate outputs r'_1,...,r'_4 during the BIST that have the same 32-bit CRC checksum as the correct outputs r_1,...,r_4. Since the input to the rate matcher during the BIST is known, the

Trojan designer can compute the expected 32-bit CRC checksum. He then only needs to find a suitable value for the Trojan constants $c[1:128]$ and $K[1:128-n]$, which generate the correct CRC checksum for the inputs provided during the BIST. Since the chance that two outputs have the same 32-bit CRC is $1/2^{32}$, the attacker only needs $2^{32}/2$ tries on average to find values for c and K that result in the expected 32-bit CRC. This can easily be done by simulation. By cleverly choosing c and K the Trojan now passes the BIST, while the BIST will still detect manufacturing and aging defects and therefore raises no suspicion.

Since the Trojan RNG has an entropy of n bits and uses a very good digital post-processing, namely AES, the Trojan easily passes the NIST random number test suite if n is chosen sufficiently high by the attacker. We tested the Trojan for $n = 32$ with the NIST random number test suite and it passed for all tests. The higher the value n that the attacker chooses, the harder it will be for an evaluator to detect that the random numbers have been compromised.

Detecting this Trojan using optical reverse engineering is extremely difficult since only the dopant masks of a few transistors have been modified. As discussed, detecting modifications in the dopant mask is extremely difficult in a large design, especially since only a small portion of a limited number of gates were modified. Since optical reverse-engineering is not feasible and our Trojan passes functional testing, a verifier cannot distinguish a Trojan design from a Trojan-free design. This also means that the verifier is not able to reliably verify a golden chip. But without such a verified golden chip, most post-manufacturing Trojan detection mechanisms do not work.

4 Case-study 2: Side-channel Trojan

In the first case study we showed how our dopant Trojan can be used to compromise the security of a real world system by shorting specific signals to GND and V_{DD}. With the second case study we want to emphasize the flexibility of the dopant Trojan. Instead of modifying the logic behavior of a design, the dopant Trojan is used to establish a hidden side-channel to leak out secret keys. We prove this concept by inserting a hidden side-channel into an AES SBox implemented in a side-channel resistant logic style.

We chose the side-channel resistant logic style iMDPL for our target implementation despite the fact that it has some known weaknesses, namely imbalanced routing, that can enable some side-channel attacks [14]. Our target iMDPL SBox is reasonably secure and we would like to stress that the focus of this work is hardware Trojans and not side-channel resistant logic styles. Our point here is that our Trojan modifications do not reduce the side-channel resistance against common side-channel attacks while enabling the Trojan owner to recover the secret key. In the following Section a brief introduction of iMDPL is given and then the dopant based side-channel Trojan is explained.

4.1 iMDPL

The *improved Masked Dual Rail Logic* (iMDPL) was introduced in [16] as an improvement of the Masked Dual-Rail Logic (MDPL) [17]. There are three main ideas incorporated in iMDPL:

1. Dual-Rail: for every signal a, both the true and the complementary signal (indicated with $\bar{a}$) are computed. Therefore the same number of 1's and 0's are computed regardless of the input. This prevents attacks based on the Hamming weight.
2. Precharge phase: Between two clock cycles, there is always a precharge phase in which all iMDPL gates (besides registers which have to be treated differently) are set to 0. This prevents attacks based on the Hamming distance.
3. Mask bit: Due to imbalances in routing inverse signals and process variations, the power consumption of a signal a might differ from that of its inverse signal $\bar{a}$ which can lead to side-channel attacks. In iMDPL a random mask bit is used to randomly choose between a and $\bar{a}$ to mask the power consumption.

In an iMDPL gate, every input and output bit as well as its inverse is masked with a mask bit m. An iMDPL-AND gate performing the operation $q = a \,\&\, b$ has six inputs: The masked input values $a_m = a \oplus m$, $\bar{a}_m = a \oplus \bar{m}$, $b_m = b \oplus m$, $\bar{b}_m = b \oplus \bar{m}$ and the mask bit m and its inverse $\bar{m}$. The two outputs of an iMDPL-AND are $q_m = q \oplus m$ and $\bar{q}_m = q \oplus \bar{m}$.

The schematic of an iMDPL-AND gate is shown in Figure 3. It consists of a detection stage, an SR-latch stage and two majority gates with complementary inputs. If one input of a 3-input majority gate is set to 0, the majority gate behaves like an AND gate. If one input is set to 1, the majority gate behaves like an OR gate. For the mask bit $m = 0$, the lower Majority gate with the inputs a_m, b_m and m computes $q = q_m = a \,\&\, b$ and the upper majority gate computes $\bar{q} = \bar{q}_m = \bar{a} \mid \bar{b}$. For the mask bit $m = 1$ on the other hand the lower majority gate computes $\bar{q} = q_m = \bar{a} \mid \bar{b}$ and the upper majority gate computes $q = \bar{q}_m = a \,\&\, b$. Hence, the current mask bit decides which inputs and outputs are the correct ones and which the inverse. It is also possible to create an iMDPL-OR and iMDPL-NOR gate using the same structure by switching the outputs and/or inputs. In iMDPL all combinational logic is build using these four basic operations (AND, NAND, OR and NOR). The detection and SR-latch stage was introduced in iMDPL to prevent the early propagation effect and glitches by making sure that all inputs are in a complementary stage before evaluating. A more detailed description of iMDPL can be found in [16].

As in the previous sections, the 45nm Nangate Open Cell library was used for our implementation of an area optimized Canright [3] AES SBox in iMDPL. Since the target library does not have a 3-input majority gate, we used a six input AND-OR-INVERTER (AOI) gate configured as a 3-input not-majority gate together with an inverter to build the majority gate[2].

[2] We would like to note that the layout of a majority gate is very similar to an AOI gate and we verified that the Trojan also works with a standard majority gate.

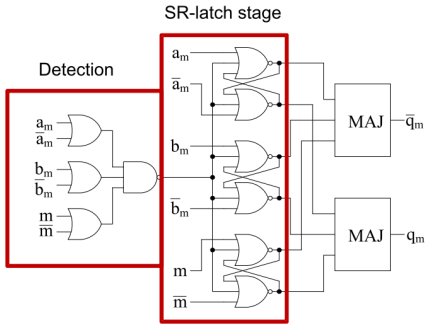


Fig. 3. Schematic of an iMDPL-AND gate consisting of two Majority gates, a detection logic and an SR-latch stage[16].

4.2 iMDPL-Trojan

To insert a Trojan into the iMDPL SBox implementation, we replace two AOI gates from a single iMDPL gate with Trojan AOI gates that create a predictable, data-dependent power consumption independent from the mask bit. Modifying only single gates makes inserting the Trojan into the design after place & route very simple, since we do not need to worry about any additional routing or find empty space in the design. Figure 4 shows the schematic of the used AOI gate configured as a 3-input not-majority gate. Two changes are made to this not-majority gate to create a large data-dependent power consumption. First, the two topmost p-MOS transistors are removed by shorting their output contacts to VDD. Secondly, the strength of the remaining p-MOS transistors is decreased by decreasing their effective width. These changes are depicted on the right side of Figure 4.

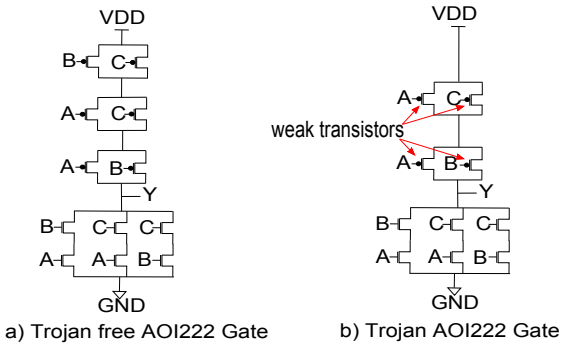


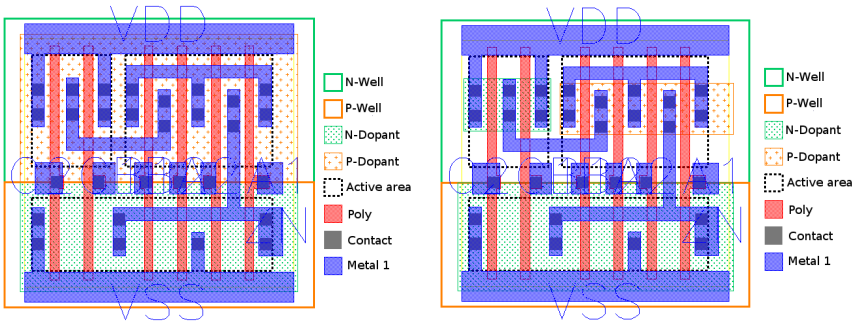
Fig. 4. Schematic of the Trojan-free and Trojan AOI222_X1 gate configured as a 3-input not-majority gate.

The Trojan not-majority gate behaves like the Trojan-free gate except for the input pattern $A = 0$, $B = 1$, and $C = 1$. In the unmodified not-majority gate the pull-up network is inactive and the pull-down network is active, resulting in an output value of 0. However, in the Trojan gate the pull-up as well as the pull-down network are both active for this input pattern. Due to the reduced size of the p-MOS transistors, the pull-up network is much weaker than the pull-down network and the resulting output voltage is therefore still close to 0. In a sense we have turned the not-majority gate into a pseudo-n-MOS gate for this input pattern. Hence, the output values of both the Trojan-free and Trojan gate are the same, but there is a large power consumption in the Trojan gate for this input pattern due to the connection between GND and V_{DD}. For all other inputs only the pull-up or pull-down network is active for the Trojan gate as well as the Trojan-free gate.

If the two not-majority gates of the iMDPL gate are exchanged with this Trojan gate, a high power consumption is generated whenever one of the two AOI gates has the input $A = 0$, $B = 1$, and $C = 1$. In our configuration this is the case if $a_m = 0$, $b_m = 1$, $m = 1$ or if $\bar{a}_m = 0$, $\bar{b}_m = 1$, $\bar{m} = 1$ which turns out to be the case for $a = 1$, $b = 0$ regardless of the value of m. Hence, the Trojan iMDPL gate has a data-dependent power consumption that is independent of the mask bit m.

We used the technique of dopant Trojans described in Section 2 to realize our Trojan AOI gate. The modifications were done using Cadence Virtuoso Layout editor and are shown in Figure 5(b). The Trojan gate passed the DRC check and we used Calibre PEX in Virtuoso to do the netlist and parasitic extraction. The Trojan and Trojan-free gate were simulated in HSpice. The propagation delay, rise and fall time of a Trojan iMDPL gate are very similar to the Trojan-free iMDPL implementation. This makes it possible to place our Trojan gates even in the critical path without creating timing violations. The additional power consumption when the Trojan activates depends on the used clock frequency, since the majority of power consumption of the Trojan is static current due to the connection between V_{DD} and GND. Even at a very high frequency such as 10 GHZ, the Trojan gate consume roughly twice as much power when the Trojan activates compared to the Trojan-free counterpart.

To insert our Trojan iMDPL gate in the layout of the target SBox implementation after place & route we need to identify an iMDPL gate that serves as a suitable Trojan location and replace the AOI gates of this target iMDPL gate with the Trojan AOI gate. Finding a suitable location does not require a detailed knowledge of the target SBox. In fact, the right location can be identified using simulation. The individual iMDPL gates can easily be identified by searching for AOI gates connected with inverse inputs. In the first step, we simulated the SBox for all 512 possible inputs (for each mask there are 256 different inputs) and stored the inputs and outputs for the tested AOI gates. Then, a matlab script was used to test the performance of possible Trojan target locations. We chose a target location that (1) had a small correlation with the Trojan power model for all false key guesses to make it easy for the owner of the Trojan to use it and



(a) Trojan-free AOI gate (b) Trojan AOI gate

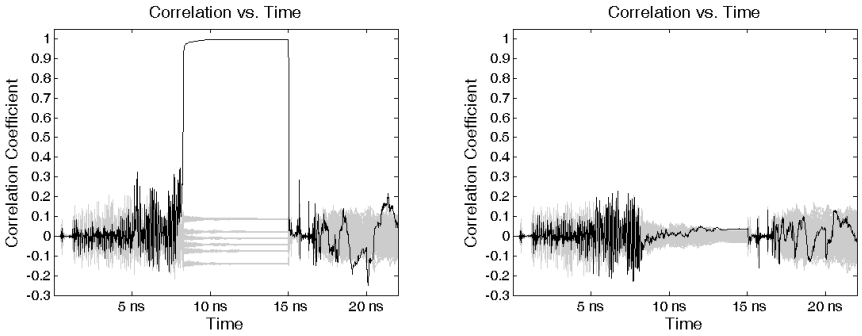
Fig. 5. On the left (a) the layout of the unmodified AOI222_X1 gate and on the right (b) the Trojan AOI222_X1 gate is shown. In the Trojan gate the p-MOS transistors in the upper left active area have been shorted with the n-well by replacing the p-implant with n-implant. The strength of the remaining p-MOS transistors in the upper right active area have been reduced by decreasing the p-implant in this area.

(2) a location which did not increase the vulnerability against the considered side-channel attacks. We tested (2) by performing the considered side-channel attacks on hypothetical power traces based on the Trojan power model. Once we located a good Trojan location we simply replaced the corresponding AOI gates with the Trojan AOI gate.

4.3 Result

To verify the correct functioning of our Trojan we performed a side-channel attack with the Trojan power model using the Trojan Sbox implementation and the Trojan-free implementation on simulated power traces. Figure 6(a) shows the result of the attack on the Trojan SBox and Figure 6(b) shows the result of performing the same attack on the Trojan-free implementation. The correct key can clearly be distinguished for the Trojan SBox with a correlation close to 1. It is also interesting to note that the Trojan generates static current compared to switching current. Hence, one can make power measurements after most switching activity has occurred and use integration to increase the signal-to-noise ratio. This makes using the Trojan easy in a practical setting. As expected, the Trojan power model does not reveal the key in the Trojan-free implementation, which shows that the side-channel was indeed produced by the added Trojan.

We then compared the side-channel resistance of the Trojan implementation with the Trojan-free implementation. Covering all possible side-channel attacks is far out of the scope of this paper. We therefore only considered the most common side-channel attacks, namely 1- and 8-bit CPA [2] and MIA [5]. We found a small vulnerability in the Trojan-free design, which is in line with the results from [14]. However, the Trojan did not increase this weakness and the Trojan design is as side-channel resistant as the Trojan-free design against the



(a) Trojan design (b) Trojan-free design

Fig. 6. 1-Bit CPA on (a) the Trojan design and (b) the Trojan-free design using the Trojan power model with the evaluation phase starting at 0ns and the precharge phase starting at 15ns. The correct key is shown in black and the false keys are shown in gray. The correlation for the correct key in the Trojan design goes up to 0.9971.

considered side-channel attacks. The side-channel analysis showed that we have successfully established a hidden side-channel that can leak out secret keys very reliably while not decreasing the side-channel resistance against the most common side-channel attacks. Hence, the newly introduced Trojan side-channel can only be used by the owner of the Trojan who knows the secret Trojan power model.

Since we did not change the logic behavior of any gate, no kind of functional testing can detect the Trojan. As discussed in Section 2, optical inspection cannot detect the Trojan since we only modified the dopant masks. Without being able to detect the Trojan using functional testing or optical inspection, an attacker cannot distinguish a Trojan chip from a Trojan-free chip. Hence, an evaluator cannot verify a golden chip and therefore methods that rely on a golden chip have only limited use in detecting the Trojan. This shows that detecting a dopant-based side-channel Trojan would be really challenging in practice using known methods.

5 Conclusions

In this paper we introduced a new type of sub-transistor level hardware Trojan that only requires modification of the dopant masks. No additional transistors or gates are added and no other layout mask needs to be modified. Since only changes to the metal, polysilicion or active area can be reliably detected with optical inspection, our dopant Trojans are immune to optical inspection, one of the most important Trojan detection mechanism. Also, without the ability to use optical inspection to distinguish Trojan-free from Trojan designs, it is very difficult to find a chip that can serve as a golden chip, which is needed by most post-manufacturing Trojan detection mechanisms.

To demonstrate the feasibility of these Trojans in a real world scenario and to show that they can also defeat functional testing, we presented two case studies. The first case study targeted a design based on Intel's secure RNG design. The Trojan enabled the owner of the Trojan to break any key generated by this RNG. Nevertheless, the Trojan passes the functional testing procedure recommended by Intel [9, 24] for its RNG design as well as the NIST random number test suite. This shows that the dopant Trojan can be used to compromise the security of a meaningful real-world target while avoiding detection by functional testing as well as Trojan detection mechanisms. To demonstrate the versatility of dopant Trojans, we also showed how they can be used to establish a hidden side-channel in an otherwise side-channel resistant design. The introduced Trojan does not change the logic value of any gate, but instead changes only the power profile of two gates. An evaluator who is not aware of the Trojan cannot attack the Trojan design using common side-channel attacks. The owner of the Trojan however can use his knowledge of the Trojan power model to establish a hidden side-channel that reliably leaks out secret keys.

Detecting this new type of Trojans is a great challenge. They set a new lower bar on how much overhead can be expected from a hardware Trojan in practice (i.e. zero!). Future work should include developing new methods to detect these sub-transistor level hardware Trojans.

References

1. D. Agrawal, S. Baktir, D. Karakoyunlu, P. Rohatgi, and B. Sunar. Trojan Detection using IC Fingerprinting. In *IEEE Symposium on Security and Privacy (SP 2007)*, pages 296–310, 2007.
2. E. Brier, C. Clavier, and F. Olivier. Correlation Power Analysis with a Leakage Model. In *Cryptographic Hardware and Embedded Systems - CHES 2004*, LNCS, pages 16–29. Springer, 2004.
3. D. Canright. A very compact S-box for AES. In *Cryptographic Hardware and Embedded Systems - CHES 2005*, LNCS, pages 441–455. Springer, 2005.
4. Defense Science Board. Report of the Defense Science Board Task Force on High Performance Microchip Supply. US DoD, February 2005.
5. B. Gierlichs, L. Batina, P. Tuyls, and B. Preneel. Mutual Information Analysis. In *Cryptographic Hardware and Embedded Systems - CHES 2008*, LNCS, pages 426–442. Springer, 2008.
6. C. Gorman. Counterfeit chips on the rise. *IEEE Spectrum*, 49(6):16–17, june 2012.
7. M. Hamburg, P. Kocher, and M. E. Marson. Analysis of Intel's Ivy Bridge Digital Random Number Generator. Technical Report, Cryptography Research INC., March 2012.
8. M. Hicks, M. Finnicum, S. T. King, M. M. Martin, and J. M. Smith. Overcoming an untrusted computing base: Detecting and removing malicious hardware automatically. In *IEEE Symposium on Security and Privacy (SP 2010)*, pages 159–172, 2010.
9. Intel. Intel Digital Random Number Generator (DRNG) Software Implementation Guide. `http://software.intel.com/sites/default/files/m/d/4/1/d/8/441_Intel_R__DRNG_Software_Implementation_Guide_final_Aug7.pdf`, Aug 2012. Revision 1.1.

10. S. T. King, J. Tucek, A. Cozzie, C. Grier, W. Jiang, and Y. Zhou. Designing and implementing malicious hardware. In *Proceedings of the 1st USENIX Workshop on Large-scale Exploits and Emergent Threats (LEET 08)*, pages 1–8, 2008.
11. J. Li and J. Lach. At-speed delay characterization for IC authentication and Trojan horse detection. In *IEEE International Workshop on Hardware-Oriented Security and Trust (HOST 2008)*, pages 8–14, 2008.
12. L. Lin, M. Kasper, T. Güneysu, C. Paar, and W. Burleson. Trojan Side-Channels: Lightweight Hardware Trojans through Side-Channel Engineering. In *Cryptographic Hardware and Embedded Systems - CHES 2009*, LNCS, pages 382–395. Springer, 2009.
13. S. Markoff. Cyberwar — Old Trick Threatens the Newest Weapons. *New York Times*, October 2009.
14. A. Moradi, M. Kirschbaum, T. Eisenbarth, and C. Paar. Masked Dual-Rail Precharge Logic Encounters State-of-the-Art Power Analysis Methods. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, PP(99):1–13, 2011.
15. Nangate Inc. Nangate Open Cell Library, version PDKv1_3_v2010_12. `http://www.si2.org/openeda.si2.org/projects/nangatelib`, August 2011.
16. T. Popp, M. Kirschbaum, T. Zefferer, and S. Mangard. Evaluation of the Masked Logic Style MDPL on a Prototype Chip. In *Cryptographic Hardware and Embedded Systems - CHES 2007*, LNCS, pages 81–94. Springer, 2007.
17. T. Popp and S. Mangard. Masked Dual-Rail Pre-charge Logic: DPA-Resistance Without Routing Constraints. In *Cryptographic Hardware and Embedded Systems - CHES 2005*, LNCS, pages 172–186. Springer, 2005.
18. J. Rajendran, V. Jyothi, and R. Karri. Blue team red team approach to hardware trust assessment. In *IEEE 29th International Conference on Computer Design (ICCD 2011)*, pages 285–288, oct. 2011.
19. J. Rajendran, V. Jyothi, O. Sinanoglu, and R. Karri. Design and analysis of ring oscillator based Design-for-Trust technique. In *29th IEEE VLSI Test Symposium (VTS 2011)*, pages 105–110, 2011.
20. D. Sanger, D. Barboza, and N. Perlroth. Chinese Army Unit Is Seen as Tied to Hacking Against U.S. *New York Times*, February 2013.
21. Y. Shiyanovskii, F. Wolff, A. Rajendran, C. Papachristou, D. Weyer, and W. Clay. Process reliability based trojans through NBTI and HCI effects. In *NASA/ESA Conference on Adaptive Hardware and Systems (AHS 2010)*, pages 215–222, 2010.
22. SypherMedia International. Circuit Camouflage Technology - SMI IP Protection and Anti-Tamper Technologies. White Paper Version 1.9.8j, March 2012.
23. A. Waksman and S. Sethumadhavan. Silencing hardware backdoors. In *IEEE Symposium on Security and Privacy (SP 2011)*, pages 49–63, 2011.
24. J. Walker. Conceptual Foundations of the Ivy Bridge Random Number Generator. Presentation at ISTS Computer Science Department Colloquium at Dartmouth College, Nov 2012. http://www.ists.dartmouth.edu/docs/walker_ivy-bridge.pdf.
25. J. Yier and Y. Makris. Hardware Trojan detection using path delay fingerprint. In *IEEE International Workshop on Hardware-Oriented Security and Trust (HOST 2008)*, pages 51–57, 2008.