

CSAW 2008 Team Report

Yale University

Yier Jin, Nathan Kupp

Department of Electrical Engineering, Yale University, 10 Hillhouse Avenue, New Haven, CT 06520, USA

Abstract

Hardware Trojans are an increasingly significant threat to the integrated circuit (IC) industry in both commercial and military fields due to the progress of globalization and the rapidly improving IC manufacturing technology in various countries. Because of global economic pressures, the development and fabrication of advanced ICs are often outsourced offshore in order to reduce cost. As a result, the entire IC supply chain that was once located in one country can now often be globally distributed. To control every point of this manufacturing process, including all manufacturing facilities is almost impossible. On the other hand, compromising the IC supply chain for sensitive commercial and defense applications has become significantly easier, as control is ceded over these critical segments. In this work, we present several attack techniques, employing hardware trojans to break the security of an Alpha encryption module implemented on a Digilent BASYS Spartan-3 FPGA board.

1. Introduction

As the threat of hardware trojans becomes greater, some key areas of circuit design present themselves as likely targets for malicious attackers. We note that while IC design and final product shipping are often performed internally, the growing industry trend is to export chip fabrication, packaging, and other low-level tasks, driven by the economics of the IC fabrication industry. Clearly, this is a security issue, as the Register Transfer Level (RTL) design, fabrication masks, and packaged parts may no longer be under complete control.

Perhaps the most straightforward attack vector for a malicious attacker is to modify the RTL design. Due to the high complexity of modern ICs, auto-placement and auto-routing tools are widely used in order to accelerate development time. These tools, however, are not ideal, and often leave a portion of chip space unused due to suboptimal placement and routing. This situation enables attackers to embed malicious circuits, referred to as Trojan circuits, in the unused space, or otherwise modify design parameters without changing the area of the whole chip.

The purpose of this work is to investigate different attacks on a design at the RTL level. Specifically, we examine the possibility of designing hardware Trojans that are able to evade state-of-the-art detection methodologies, as well as pass functional test. The *Alpha* device is considered as a target for these attacks, shown in Fig. 1. This device contains a 128-bit AES block encryption module, as well as an RS-232 communication channel over which ciphertext is transmitted.

2. Hardware Trojans

In order to demonstrate the efficacy of hardware Trojans with a range of sophistication, we designed 8 types of hardware Trojans (named tro1, tro2,..., tro8 in the rest of the report) and implemented them

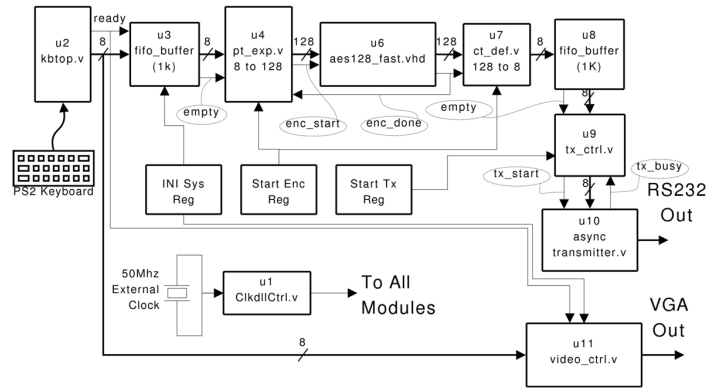


Fig. 1. Alpha architecture

in the *Alpha* architecture. To comply with the requirements of the competition, all of the Trojans were designed on the RTL level, that is, only the HDL code has been modified. In the following sections, we examine different characteristics of hardware Trojans over a range of sophistication, and discuss the Trojans implemented for this work.

2.1. Payload

For a hardware Trojan to be useful, it must carry a *payload*. In other words, we attempt to link the activation of the Trojan with some deterministic event that is favorable to the attacker. There are several categories into which hardware Trojans can be classified according to these payloads:

- 1) Broadcast to the attacker some internal signals, which may be secret (e.g. the encryption key).
- 2) Compromise the function of the circuits (e.g. to replace the plaintext with other preset information).
- 3) Destroy the chip.

Our designs, tro1, tro4, tro5, tro7, and tro8 belong to the first group. Clearly, to an attacker, the most valuable information is the encryption key. Thus, when the Trojan is triggered, the key will be transmitted along with the cipher text. The attacker, by appropriately listening to the transmission channel, can thus acquire the key and break the system.

Tro3 belongs to the second group, the replacement of plaintext. With this Trojan, the legitimate receiver will always receive reasonable plaintext. For example, in Tro3, whenever we input “Moscow”, it will be changed to “Boston”, i.e., the plaintext “launch the missile targeting Moscow” will be changed to “launch the missile targeting Boston”.

Tro2, tro6 belong to the third group, and are designed to destroy the whole chip. This approach has the advantage of being very difficult to detect during functional test, as the configured sleep time of the Trojan will often be much longer than the testing time. Here, a counter plays the role of a trigger.

2.2. Triggers

Similarly, we can classify the Trojans according to their *trigger*, an event which enables the Trojan. This event is designed to evade functional test by either conditioning on rare/atypical events, such as undefined input sequences, or by using a counter with a very long period. For triggers, we have the following categories:

- 1) The attacker can physically access the device and can give special input to trigger the Trojan directly.
- 2) The Trojan is triggered internally: by a specific input event, counter, or other signal change.
- 3) No trigger; the trojan is always activated.

Tro1, tro2, and tro8 belong to the first group in which the trigger methods can be quite obscure and hard to detect during testing. The most compact but useful one is to redefine an unused key in the keyboard to trigger the Trojan. Since only the alphanumeric characters are officially supported by *Alpha*, this trigger can reasonably be expected to escape functional test. An alternative is to use a specially designed input string. Using a string such as this should easily escape functional testing since the input space is very large. Clearly, however, a trigger that relies on physical access is limited in application.

The second class of triggers (tro3, tro4, tro5, tro6, tro7) is for devices which cannot be physically accessed by an attacker. In this case, we note that some type of internal counter can be used as an activation point for the Trojan. For example, we can implement a counter to track the number of transmission times by *Alpha*, and trigger after it exceeds a pre-specified number. A real-time counter can also be used as a trigger. Finally, the change of internal signals can also be used as the trigger, such as in tro5. In this design, whenever the key has been changed, the Trojan will be activated. An even more sophisticated method inserts a RS232 receiver in order to let the attacker control the whole chip through RS232 channel.

The third group is more aggressive, and configures the Trojan to be always activated. This requires the payload of the Trojan to be well hidden, i.e., by broadcasting secret information in ways undetectable by functional test. A shortcoming of configuring a Trojan this way is that the power usage will be measurably higher, potentially raising suspicions during test. A modified version of tro5 belongs to this group but it is not included in our final report.

2.3. Code Optimization

Along with the trigger/payload-based Trojans described above, there are two Trojan targets which can be quite effective and hard to detect on the RTL level¹.

The first is targeting unoptimized HDL code. For example, some modules may be written in an unoptimized way and even with the help of advanced synthesis tools, the generated gate-level design will have significant amounts of redundant logic. The attacker who has already mastered the timing sequence and module function can rewrite the module in a significantly more compact way while performing the same functionality. The on-chip resources saved by this optimization can then be allocated to a Trojan. For example, in our work we rewrite the `pt_exp.v` module to save 128 Flip-flops. In tro2, tro7 and tro8, the original `pt_exp.v` file is replaced by our own designed module to balance the total on-chip area usage and power consumption.

The second is targeting standard or commercial modules (or IP cores). IP cores are now widely used both in academic and commercial designs to minimize development time. As a reusable module, IP cores are typically designed to complete some generic function in order to suit many different usages. However, in most applications, not all of these functions will be used. This leads to redundant logic which can be optimized away by modifying the code to eliminate unused functions. Again, the chip area saved by this modification can be used to insert Trojans. For example, we changed the `Ps2interface.vhd` module which saved 13 Flip-flops and 10 4-input LUTs.

In tro2, and tro3, the original `Ps2interface.vhd` file is replaced by our optimized module to reduce the total on-chip area usage and power consumption.

3. Trojan Implementations

To evaluate the different types of trigger and payload combinations in a real-world system, a total of 8 Trojans were implemented into the *Alpha* architecture. These implementations were performed within the Xilinx ISE Webpack 10.1 environment, using the target chip XC3S250e-4tq144 on a Digilent Basys

¹Note that these two kinds of Trojans platforms could be much more difficult to be implemented in Gate level or transistor level.

development board. There is a conclusion table giving basic characteristics of all these eight Trojans at the end of this report.

3.1. Trojan type I

Description and trigger mechanism: This Trojan is inserted at the front-end of the design and monitors the keyboard input. If the phrase “New Haven” is detected at the beginning of plaintext, the Trojan will be triggered. Whenever the Trojan is triggered, the first block (128 bits) of the output ciphertext will be replaced by the encryption key. The majority of the Trojan code is located in `alphatop.v` file where a FSM is inserted. The remainder of the Trojan source code can be found in file package `alpha_Tro1.rar`.

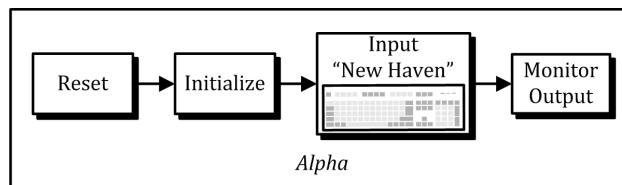


Fig. 2. Trojan type I architecture

Platform and area consumption: The original code without any modification was used as the platform for this Trojan. Altogether, 1486 Flip-flops and 4320 LUTs are used which correspond to +0.8% and +6.8% more usage than original design, respectively.

Implementation efficacy: The attacker who has access to the input (keyboard) and is listening to the output (RS232 serial port) can easily acquire the encryption key. A remote attacker who cannot access the device physically but can listen to the communication channel can still acquire the key, but is reliant on a user entering the special text as plaintext.

Limitations: As mentioned above, a special input string is required to trigger the Trojan. Therefore, either the attacker must have physical access to the system, or depend on the end user entering this special string. Of course, the attacker must also have the ability to monitor the output. Moreover, the chip area overhead will probably lead to more power consumption, raising it to suspicious levels during test. Although it is extremely unlikely, there is a non-zero probability that the Trojan will be discovered during functional test, if the special keyword is used to test the system. Furthermore, when the Trojan is triggered, the legitimate receiver will still attempt to decrypt the ciphertext which has been replaced by the encryption key. This will result in the receiver acquiring garbage data instead of the correct plaintext, which could also lead to the detection of the Trojan.

3.2. Trojan type II

Description and trigger mechanism: This Trojan was designed to reduce power usage significantly to avoid suspicious increases in power usage due to a Trojan. Two verilog files were edited to remove extraneous components and reduce space usage, and then the Trojan was implemented. For this Trojan, the trigger is an originally undefined key “F12”. Whenever, the key is pressed, the triggered Trojan will lock and ignore any input unless the FPGA is reprogrammed. The primary Trojan code is located in the `kb2ascii.v` file where the “F12” key is defined. The Trojan source code can also be found in file package `alpha_Tro2.rar`.

Platform and area consumption: The code with optimized `pt_exp.v` and `kb_top.v` modules was used as a platform. Altogether, 1336 Flip-flops and 4198 LUTs are used which are 9.4% less than the original design and 0.024% more than original design, respectively. Clearly, using too few Flip-flops is not problematic, as we can insert dummy FFs to match the original design at will.

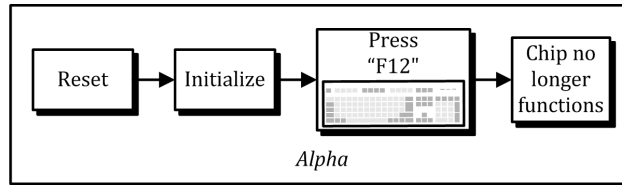


Fig. 3. Trojan type II architecture

Implementation efficacy: The attacker who can access the input can easily trigger the Trojan. A remote attacker who cannot access the device physically is reliant on a user mistakenly pressing the undefined key to trigger the Trojan.

Limitations: Due to the limitations of the synthesis tools, we cannot destroy the chip physically through RTL code, i.e., by making a connection from V_{DD} to GND . As the Trojan can be deactivated by simply reprogramming the device, this type of Trojan would likely not incur serious long-term negative effects for the user.

3.3. Trojan type III

Description and trigger mechanism: This Trojan is designed for the special case where the attacker knows the usage of the circuit and can architect a plaintext-replacement scheme appropriately. In this Trojan, whenever the word “Moscow” is detected in the plaintext, it will be replaced by “Boston”. The phrase of interest and the replacement phrase can be changed according to the usage of the Trojan and the chip. The Trojan code is located in the `alphatop.v` file where a FSM is inserted, as well as in file package `alpha_Tro3.rar`.

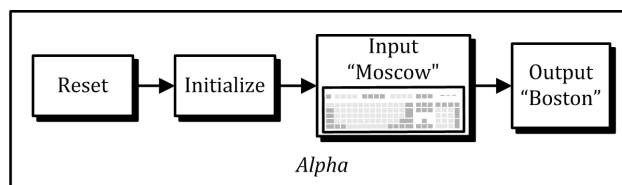


Fig. 4. Trojan type III architecture

Platform and area consumption: The code with optimized `kb_top.v` module. Altogether, 1523 Flip-flops and 4266 LUTs are used which are +3.3% and +1.6% more than original design.

Implementation efficacy: The attacker does not need to access the input equipment nor does he need to monitor the output. The selection of words to replace is critical in the ability of the Trojan to modify the function of the system in some way useful to the attacker.

Constraints: As mentioned above, the selection of keywords is critical when designing the Trojan. Again, however, this type of Trojan would succumb to exhaustive testing of the input space during functional test. Furthermore, if TMR (Triple Modular Redundant) is used with three different encryption modules (the others without the hardware Trojan) to encrypt and transfer data, the modified result will be discarded and the effect of the Trojan will be nullified.

3.4. Trojan type IV

Description and trigger mechanism: The Trojan is custom-designed targeting the mismatched buffer sizes in the system. The plaintext buffer size is only 1 KB, while the video buffer is 4 KB. This makes

it difficult for the operator to know whether he or she has overflowed the input buffer with too much plaintext. When the overflow of input buffer occurs, the Trojan is triggered and the payload is to replace the last block with the encryption key. This approach takes advantage of a detail in the original design: according to the design, if the input is 1KB, the last byte will be discarded in order to transmit the key index. So a legitimate receiver will discard the last block as garbage. Since the receiver will discard this block, the Trojan can simply replace the block with the entire encryption key. The Trojan code is located in the `alphatop.v` file, as well as the file package `alpha_Tro4.rar`.

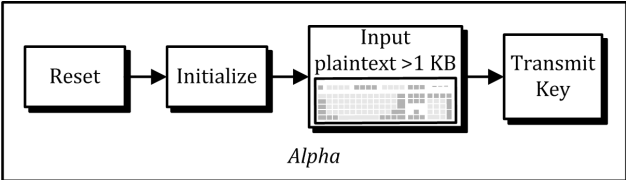


Fig. 5. Trojan type IV architecture

Platform and area consumption: The original code without any modification was used as a platform. Altogether, 1475 Flip-flops and 4273 LUTs are used which are +0.068% and +1.8% more than original design.

Implementation efficacy: The attacker need not physically access the device, but should have access to the communications channel to listen for the encryption key after a buffer overflow has occurred.

Constraints: The attacker must continuously monitor the communications channel, and check the last block of each transmission burst. If a buffer-overflow test is included in the functional test plan, it may detect this type of Trojan.

3.5. Trojan type V

Description and trigger mechanism: This Trojan is designed to compromise the communications channel by hiding information in the transmissions over the RS-232 channel. In our design, the baud rate of the serial port is changed from 9600 bps to 19200 bps. In the original design, each RS-232 packet contains 1 start bit, 8 data bits, and 2 stop bits. At the 19200 bps baud rate, these parameters were changed to 1 start bit, 8 data bits and 1 stop bit. This enables the Trojan to interleave other information, i.e., the encryption key, along with the legitimate data. For each legitimate 8-bit packet, as many as 7 bits can be inserted. In our design, we use four of these bits. When the key index is changed, the Trojan is triggered to insert the encryption key into the hidden channel. The legitimate user or functional test equipment will not observe any difference in the output if the baud rate on the receiver is set at 9600 bps. Thus, the only possible means for detecting this Trojan is through comparing power consumption. The Trojan code is located in `async_transmitter.v` file, as well as in the file package `alpha_Tro5.rar`.

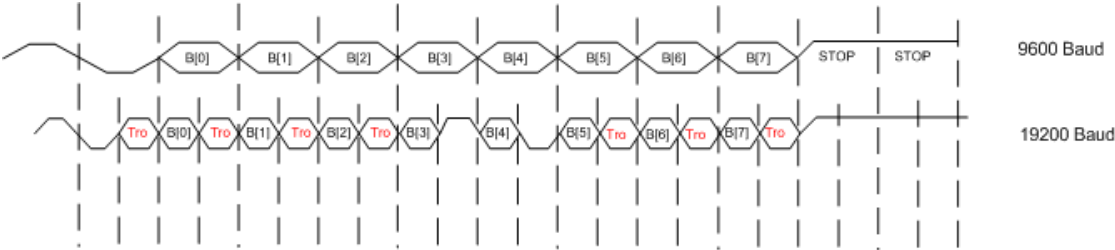


Fig. 6. Compromised RS232 channel

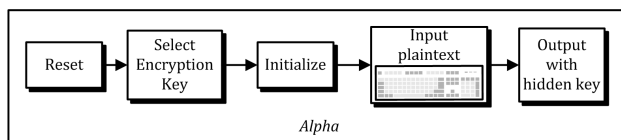


Fig. 7. Trojan type V architecture

Platform and area consumption: The original code without any modification was used as a platform. Altogether, 1485 Flip-flops (0.75% more than original design) and 4255 LUTs (+1.4% more than original design) are used.

Implementation efficacy: This is a sophisticated Trojan which is immune to functional testing within specification parameters, despite being triggered frequently. The attacks only need to monitor the RS-232 transmission channel at the 19200 Baud rate to acquire both the key and the ciphertext.

Constraints: Since the area overhead is insignificant, the Trojan cannot be detected by functional testing or by power trace testing for the majority of the time. However, when the Trojan is triggered, the power consumption at transmission stage can be higher than normal due to high Baud rate. Therefore, an on-line power profile monitoring method could potentially detect the Trojan.

3.6. Trojan type VI

Description and trigger mechanism: This Trojan is designed as a timing bomb. An inserted counter will increment for each character transmitted until it exceed a predefined number, N . When the Trojan is triggered, the output will be locked to a binary 1. The Trojan code is located in `alphatop.v` file, as well as in file package `alpha_Tro6.rar`.

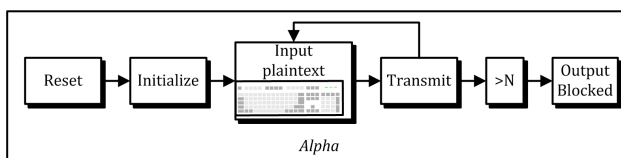


Fig. 8. Trojan type VI architecture

Platform and area consumption: The original code without any modification was used as a platform. Altogether, 1479 Flip-flops and 4204 LUTs are used which are +0.34% and +0.17% more than original design.

Implementation efficacy: The Trojan is solely a time-bomb which will not be activated until the count of transmissions exceeds a predefined number. The attack should carefully set the number N .

Constraints: For this Trojan, the setting of N is critical to the whole design. A poor choice could result in detection during functional test. Due to the limitation of synthesis tools, we cannot destroy the chip physically through RTL code, which means that when the Trojan is triggered, re-programming is an easy way to repair the chip. However, in a real environment, the resulting payload could result in destroying the chip, for example by activating a path that shorts V_{DD} to ground.

3.7. Trojan type VII

Description and trigger mechanism: This Trojan is very aggressive. In this design, the originally unused RxD port of Basys Board is configured as the trigger/control signal and one `async_receiver.v` module is added to the design. Along with a Trojan trigger signal, two more control signals are inserted in the original

design: `tro7_trigger_rst`, and `tro7_trigger_tx`. These two signals are controlled through RxD port and act as remote versions of the Reset and Transmit buttons on the Basys Board. A new encryption key is also inserted (128'h000102030405060708090a0b0c0d0e0f), which is used to encrypt the information the Trojan transmits over the RS-232 communications channel. For example, in our implementation, the original encryption key is encrypted with this key, and then broadcast across the communications channel, enabling the attacker to acquire the encryption key. At the same time, the legitimate user is left unaware that the system has been compromised by the broadcast of the key, as the encrypted key looks like a garbage block. Table 1 displays a tabulation of the commands our implementation handles over the RS-232 RxD port. The Trojan code is located in `alphatop.v` and the extra `async_receiver.v` files, as well as in file package `alpha_Tro7.rar`.

Transmit to the RS-232 RxD port	Effect
<code>Trojanefttri</code>	Reset the system
<code>Trojanabtri</code>	Encrypt the encryption key with the Trojan key
<code>Trojancdtri</code>	Transmit encrypted key on RS-232 TxD port

TABLE 1. Trojan Command Keys

Platform and area consumption: The code with an optimized `pt_exp.v` module was used as a platform. Altogether, 1409 Flip-flops and 4401 LUTs are used which are 4.4% less than original design and 4.9% more than original design, respectively.

Implementation efficacy: Since the RxD channel in the original design is idle, the attacker must control this channel and monitor the TxD channel. And the `tro7_trigger_rst` signal can remove any traces the attackers left to make it even impossible to detect the Trojan in daily use.

Constraints: To control the chip and trigger the Trojan, the attacker should acquire access to the RxD port of the device, which is more difficult than simply monitoring the output. Of course, if the physical connection to the RxD port does not exist, this Trojan will never be activated.

3.8. Trojan type VIII

Description and trigger mechanism: This Trojan was designed for the occasion when the attacker does not have access to the communications channel, but does have access to the input device—the keyboard. The technique we demonstrate here is one of many ways of extracting information from the device without using the RS-232 communications channel. For this Trojan, the trigger is the “Caps Lock” key, or any other undefined key on the keyboard. After a system reset, when the “Caps Lock” key is pressed, the Caps Lock LED will be on/off to indicate a '1' or '0' as the least-significant bit of the encryption key. The attacker need only to press the “Caps Lock” 128 times to acquire the entire key, progressing to the most-significant bit. The Trojan code is located in `kbt0p.v` and `kb2ascii.v` files, as well as in `alpha_Tro8`.

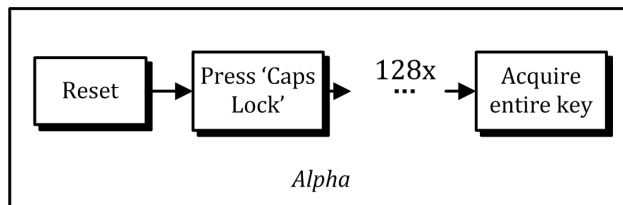


Fig. 9. Trojan type VIII architecture

Platform and area consumption: The code with an optimized `pt_exp.v` module was used as a platform. Altogether, 1396 Flip-flops and 4305 LUTs are used which are 5.3% less than original design and 2.6% more than original design, respectively.

Implementation efficacy: This Trojan removes the need for attackers to monitor the communication channel during an attack. The attacker need only monitor the keyboard LED to determine the encryption key and thus break the system.

Constraints: As mentioned above, the attackers should acquire access to the input device, which may not be feasible in many situations.

4. Lessons Learned

This competition has afforded our team a good opportunity to have an in-depth look at hardware Trojan implementations with RTL code. We investigated many characteristics a successful Trojan should have: the ideal of perfectly passing functional test, while not substantially altering the footprint of the design or the power usage. Of the eight types of Trojans we designed, each of them targets one category of the various types of Trojans that are possible. This design experience persuaded us that current RTL designs are indeed quite vulnerable to a hardware Trojan attack, under the right conditions. Whether the attack is targeted at the inputs, outputs, or internal components of a RTL system, there are many points of vulnerability that can be exploited for malicious purposes.

The designed Trojans also demonstrated that traditional functional testing can often be useless in detecting and preventing hardware Trojan attacks. This provides significant impetus to improve methods for combating hardware Trojans.

5. Conclusions

In this work, we have explored eight different implementations of hardware Trojans. Each of these have different combinations of triggers, payloads, as well as unique parts of the architecture that each Trojan attacks. Most importantly, the Trojans explored herein are designed with varying levels of sophistication, allowing the attacker to tradeoff design time, ability to evade detection, and payload.

Acknowledgements

We would like to thank our advisor Yiorgos Makris for his input, as well as the rest of the TRELAb lab for advice and support. We would also like to thank the competition sponsors, NYU-Polytechnic, for their provision of the hardware and *Alpha* source code.

Trojan Summary Table

Type	Trigger		Payload		Changed Mod.
	By Whom	How	How	Leaking Channel	
Tro1	Attacker with access to input device	Input "New Haven"	First block of ciphertext replaced by key	RS-232 TxD	N/C
Tro2	Attacker/legitimate user	Press "F12" key	The chip stops working	--	<i>pt_exp, kb_top</i>
Tro3	Legitimate user	Input "Moscow"	"Moscow" is replaced by "Boston" at output	RS-232 TxD	<i>kb_top</i>
Tro4	Legitimate user	Input > 1KB data	Last block of ciphertext replaced by key	RS-232 TxD	N/C
Tro5	Legitimate user	When key is changed	New key is hidden in the output	RS-232 TxD	N/C
Tro6	Legitimate user	Transmit > N	The chip stops working	--	N/C
Tro7	Attacker (using the RxD port)	Control w/ RxD port	Chip controlled by the attackers	RS-232 TxD	<i>pt_exp</i>
Tro8	Attacker with access to input device	Press "Caps Lock" key	The "Caps Lock" LED will reveal the key	Keyboard	<i>pt_exp</i>

Type	Area Overhead		Testing Detection Likelihood	
	Flop-flops	4-input LUTs	Functional	Power
Tro1	1486(+0.8%)	4320(+6.8%)	Unlikely ¹	Likely
Tro2	1336(-9.4%)	4198(+0.024%)	Unlikely	Unlikely
Tro3	1523(+3.3%)	4266(+2.4%)	Unlikely ¹	Likely
Tro4	1475(+0.068%)	4273(+1.8%)	Unlikely ²	Likely
Tro5	1485(+0.75%)	4255(+1.4%)	Nearly impossible	Unlikely ³
Tro6	1479(+0.34%)	4204(+0.17%)	Nearly impossible	Unlikely
Tro7	1409(-4.4%)	4401(+4.9%)	Nearly impossible	Unlikely
Tro8	1396(-5.3%)	4305(+2.6%)	Nearly impossible	Unlikely

¹Except with exhaustive input patterns

²Except with buffer-overflow test

³Except with transmit power testing