Detecting Hardware Trojans with Gate-Level Information-Flow Tracking

Wei Hu, Baolei Mao, Jason Oberg, and Ryan Kastner, *Member, IEEE*

Abstract—Hardware trust is an emerging security threat due to the globalization of hardware supply chain. A major security concern is Trojan horses inserted by an untrusted party. Hardware Trojans are carefully crafted to protect them from being identified, and detecting them in third party intellectual property (IP) cores requires significant effort. This work employs information flow tracking to discover hardware Trojans. It works by identifying Trojans that violate the confidentiality and integrity properties of these hardware components. Our method is able to formally prove the existence of such types of Trojans without activating them. We demonstrate our techniques on *trust-HUB* benchmarks and show that our method precisely detects the hardware Trojans that violate the information flow security properties related to confidentiality and integrity.

Index Terms—Hardware security, hardware trust, Trojan horse, information flow tracking, formal methods

1 Introduction

The hardware design and supply chain is now largely a global undertaking. The design process typical involves multiple teams spread around the world. Often it involves integrating third party intellectual property (IP) products from untrusted entities. As a result, hardware may be intentionally or unintentionally built with unspecified functionality. Such undocumented modifications may provide a hidden channel to leak sensitive information or a back door for attackers to compromise a system.

Hardware Trojans are a major security threat originating from malicious design modifications. These are carefully designed lightweight components that are activated under rare conditions, which protects them from being detected during the design phase. As a consequence, these hard-to-detect hidden time bombs are often identified only after severe damage has been inflicted.

Researchers have developed numerous methods for hardware Trojan detection. Initial efforts focused on exhaustive testing, which becomes intractable even for moderate scale designs. More intelligent methods utilize integrated circuit test methodologies to increase the transition probability of the Trojan trigger [1] or to identify redundant circuit with low switching probabilities [2]. However, testing is a hard problem even when not considering intentionally difficult to activate logic. A number of methods seek to capture the Trojan behaviors using side channel signal analysis [3], [4], [5], [6], e.g., they attempt to detect transient power and spurious delays added to the design due to the Trojan design. The increasing amount of hardware manufacturing process variation and decreases in the size of the Trojan payload can mitigate the effectiveness of these techniques.

While a significant amount of work focuses on detecting Trojans in fabricated hardware, this can be too late in the design process. Ideally we catch any potentially vulnerabilities during the design phase as this makes eliminating or mitigating them much easier. Modern hardware design is a massive integration of in-house and external IP cores. External vendors may provide IP cores with Trojans built in. Even internal IP cores could have Trojans due to a rogue employee. And in both cases, an IP core could have non-malicious, but equally dangerous unintended functionality that could be exploited by a hacker.

Detecting Trojans in IP cores is an extremely challenging task [7]. Many existing methods for detecting Trojans in IP cores rely on testing or verification methods to identify suspicious signals, e.g., those with extremely low transition probability [2]. However, these methods may still miss certain types of Trojans, e.g., a Trojan without a trigger signal. Some methods detect Trojans by formally proving security related properties. They indicate the existence of a Trojan when a security property is violated [8]. However, these methods typically require rewriting the hardware design in a formal language, which comes at significant design cost. Additionally, most of the existing methods may not provide clues (e.g., revealing Trojan behavior) that will help pinpoint the Trojan from the entire design.

In this work, we describe the use of information flow tracking (IFT) for hardware Trojan detection. IFT is a formal method that can be used to prove important security properties related to confidentiality and integrity. Violation of such security properties may indicate potential existence of Trojans in hardware components; regardless if it was a maliciously inserted Trojan, it tells the designer that there is a security issue, which is equally important. IFT has been widely deployed across the system stack including at the programming language/compiler, operating system, and instruction set architecture. Gate level information flow tracking (GLIFT) [9] precisely measures and controls all logical flows from the level of Boolean gates. Previous work demonstrated the use of GLIFT in crafting secure hardware architectures [10]. And it can detect security violations due to timing side channels [11].

In this work, we show how GLIFT can be used to detect hardware Trojans that violate confidentiality and integrity of critical data. Our techniques provide a measure to capture harmful flows of information and reveal the malicious Trojan behaviors, which can assist backtracking analysis in order to identify the location of Trojan design. Our method

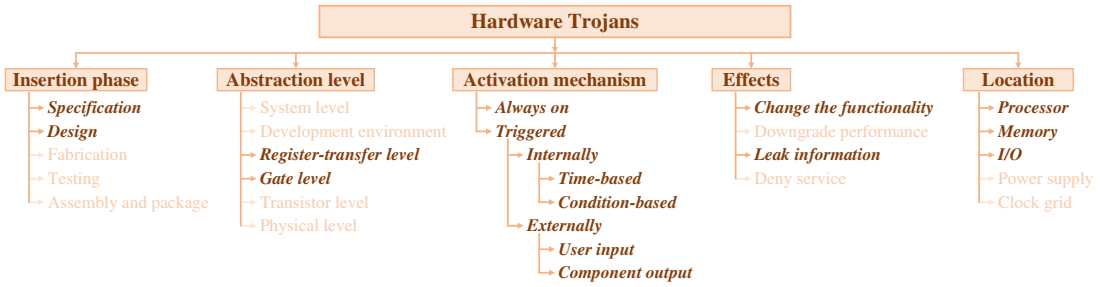

Fig. 1. A taxonomy of hardware Trojans [12]. The techniques described in this work can detect those from the ***bold face*** categories.

can detect the specific types of Trojans that can cause leakage of sensitive information and violation of data integrity. To the best of our knowledge, this is the first work that employs GLIFT for hardware Trojan detection. The contributions of this paper are:

- Proposing a formal method for detecting hardware Trojans that cause information flow security property violations;
- Providing an approach that reveals Trojan behavior and identifies the location of the Trojan in the design;
- Demonstrating the effectiveness of our techniques on the *trust-HUB* benchmarks.

2 Background

Existing hardware Trojan detection methods generally fall into two categories: invasive and non-invasive. Invasive methods either insert test points in the design for increased observability or use reverse engineering techniques to check for malicious design modification at the physical level. These methods are relatively expensive since they require highly specialized tool for physical access to the chip layout. Non-invasive methods do not need to modify the design. They look for clues, e.g., faulty output, downgraded performance, and increased power consumption, which may reveal the existence of a Trojan. Some methods try to capture these clues by functional testing [1], [2], [8] while others perform circuit parameter characterization [3], [4], [5], [6].

Salmani et al. employ a dummy scan flip-flop insertion procedure to aid circuit transition generation and reduce the Trojan activation time [1]. Zhang et al. take an alternative approach to detect Trojans by identifying redundant circuit with low transition probability [2]. Wang et al. propose a current integration methodology to reveal Trojan activity and use localized current analysis for identifying the Trojan [3]. Other work performs circuit parameter characterization to generate fingerprints or watermarks for the hardware design [4]. These fingerprints or watermarks are then compared with those for a Trojan-free reference chip for Trojan detection. However, side-channel signal analysis methods are usually affected by manufacturing process variation. Narasimhan et al. [5] propose a multiple parameter side channel analysis method, which makes use of the relationship between dynamic current and maximum operating frequency in order to minimize the effect of process noise. Jin et al. [8] propose a method for detecting Trojan in cryptography hardware by formally proving information flow related security properties. While this method can be promising in detecting Trojans in IP cores acquired from a third party, it requires careful reasoning about where information can be declassified so that the security label from the Trojan payload can be revealed. This can be a challenging task for hardware designers due to the lack of expertise in security.

In this work, we present a new formal method for detecting hardware Trojans by proving security properties related to confidentiality and integrity. Our method leverages a precise gate level information flow model that can be described with standard hardware description language (HDL) and verified using off-the-shelf electronic design automation (EDA) tools, which minimizes the additional design cost.

3 Threat Model

We focus on third party IP cores and assume that they may contain hardware Trojans that are activated only under rare conditions to leak sensitive information (such as the plaintext) or violate the integrity of critical data (e.g., the secret key); otherwise they run normally and produce correct results. We also assume that these Trojans are carefully designed and are hard to activate through pure functional testing. Our analysis requires access to the RTL code or gate level netlist of the IP cores. We assume no knowledge about the implementation details of the trigger condition or payload of the Trojans.

Figure 1 shows the types of Trojans we target. We adopt the Trojan taxonomy from [12]. We focus on Trojans inserted in the specification and design phases at the register transfer or gate level. The Trojans can be either always on or triggered under specific conditions, e.g., single input, input sequence, or counter. They can cause violation of confidentiality or integrity properties of critical data. We assume the attacker's primary goal is to learn sensitive information and do not account for Trojans that cause a denial of service or downgrade performance. We focus on Trojans that perform logical attacks and thus do not consider those that leak information through power, electromagnetic, and other side channels.

Our techniques will uncover cases where IP cores have intended or unintended functionality that can be exploited in an attack against the specified security property. We primarily focus on malicious insertions of Trojans in our examples and experiments. However, we note that design errors can be just as dangerous in terms of the security of the

hardware. Our techniques are applicable for both malicious (Trojans) and non-malicious (design flaw) scenarios.

4 Modeling Security Properties with Information Flow

We focus on the confidentiality and integrity properties of critical data in hardware designs. The confidentiality property requires that secret information can never leak to an unclassified domain. The integrity property requires that untrusted data should never be written to a trusted location. Hardware description languages are inadequate for enforcing such security properties since they only specify functionality. By contrast, information flow analysis provides a better approach for modeling these security properties since they are related to the movement of data.

To argue about security, additional sensitivity information needs to be associated with data objects so that we know what should be protected. In practice, data objects can have multiple levels of security labels according to their sensitivity. For example, data can be labeled as `Unclassified`, `Confidential`, `Secret` and `Top Secret` in a military information system. The partial order between different security classifications can be defined using a security lattice. Let $\mathcal{L}(\cdot)$ denote the function that returns the security label of a variable. This can be formalized as follows:

$$A \rightsquigarrow B \iff \mathcal{L}(A) \sqsubseteq \mathcal{L}(B) \tag{1}$$

Equation (1) models the confidentiality and integrity properties with allowed flows of information. Specifically, information is allowed to flow from A to B *if and only if* the security level of A is lower than or equal to that of B. Under such a notion, both the confidentiality and integrity properties can be modeled in a unified manner.

In this paper, we use a two level security lattice `LOW` $\sqsubseteq$ `HIGH`. In confidentiality analysis, we label the sensitive data as `HIGH` and unclassified data as `LOW`; in integrity analysis, we mark the critical data as `LOW` and normal data as `HIGH` (because integrity is a dual property of confidentiality). For example, we label the key as `HIGH` in confidentiality analysis while `LOW` in integrity analysis.

5 Methodology

5.1 Gate Level Information Flow Tracking

Gate level information flow tracking (GLIFT) is a hardware IFT technique that models digital information flows at the logical (Boolean) level [9]. GLIFT assigns each bit of data in the hardware design a label (often called *taint*). GLIFT provides a model for understanding how data propagates through a design. We can write security properties about the system, and use GLIFT to test or verify if the design adheres to these properties. For example, if we wanted to understand where information about the cryptographic key can flow, we would label the bits of the key as `Confidential`. We could then write a property that some part of the design (e.g., an untrusted memory location) should never be able to ascertain any confidential information (i.e., information from the key). This is equivalent to proving that the label of the untrusted memory location(s) can never be set as `Confidential`.

GLIFT is a fine granularity IFT method. It precisely accounts for the flow of information based upon the idea that information flows from bit A to bit B if and only if the value of A has an influence on B. GLIFT differs from other IFT methods primarily in two aspects. First, it associates each data bit with a security label while previous methods typically use byte or word level labels. Second, GLIFT takes into account the input data values when calculating the label for the output. Other IFT methods mark the output as `HIGH` if there is at least one `HIGH` input regardless of the data values. Therefore, GLIFT provides a more precise approach to determine the security label of the output. This allows a more accurate measurement of the actual flows of information.

For a better understanding of the ideas behind GLIFT, consider the two-input AND gate (AND-2), whose Boolean function can be described as $O = A \cdot B$. Let A_t, B_t and O_t denote the security labels (taints) of A, B and O respectively. Here, $A, B \in \{\texttt{0}, \texttt{1}\}$ and $A_t, B_t, O_t \in \{\texttt{LOW}, \texttt{HIGH}\}$. Under an encoding scheme (e.g., `LOW = 0` and `HIGH = 1`) where

$$\texttt{LOW} \cdot \texttt{HIGH} = \texttt{LOW}, \;\; \texttt{LOW} + \texttt{HIGH} = \texttt{HIGH} \tag{2}$$

Previous conservative IFT methods typically use the label propagation policy shown in (3) for AND-2 and set O_t to `HIGH` when either A or B is labeled as `HIGH`.

$$O_t = A_t + B_t \tag{3}$$

This is safe since it accounts for all possible flows of `HIGH` information. However, it can cause a significant amount of false positives (non-existent flows) in information flow measurement. Let $Secret$ be a 32-bit `HIGH` value. After performing the operation in (4), conservative methods will mark the entire $Public$ as `HIGH`, indicating there are 32 bits of information flowing from $Secret$ to $Public$.

$$Public = Secret \cdot \text{0x01} \tag{4}$$

GLIFT takes a more precise approach and uses (5) for label propagation for AND-2 (we refer interested readers to [9] for more details about deriving GLIFT tracking logic).

$$O_t = A \cdot B_t + B \cdot A_t + A_t \cdot B_t \tag{5}$$

From (5), the output will be `LOW` (or `HIGH`) when both inputs are `LOW` (or `HIGH`). Now consider the cases when only one input is `LOW`. When the input is (`LOW`, `0`), the output will be dominated by this input and will be `LOW` (the other `HIGH` input does not flow to the output). When the input is (`LOW`, `1`), the output will be determined by the other `HIGH` input and thus will take a `HIGH` label.

Back to the example shown in (4), the constant (`LOW`, `0`) bits in the second operand will dominate the corresponding label bits of $Public$ as `LOW`. Only the constant (`LOW`, `1`) bit allows the least significant bit of $Secret$ to flow to the output. Thus, there is only 1 bit of information flow. GLIFT considers both the security label and the actual value in label propagation. In this way, it accounts for the influence of an input value on the output and thus more precisely measures the actual flows of information.

5.2 GLIFT for Hardware Trojan Detection

Our detection method identifies Trojans that violate information flow properties, e.g., the leakage of sensitive information or overwriting high integrity data. Figure 2 shows the design flow of our method.

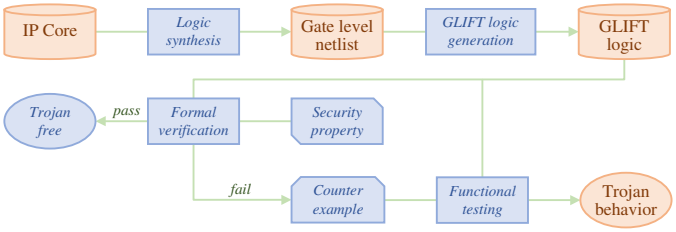

Fig. 2. The design flow of our method for hardware Trojan detection.

Given an IP core designed under test, we must first generate GLIFT logic. This process is completely automated. We first use a logic synthesis tool to compile the design to a gate level netlist. We then discretely instantiate GLIFT logic for each Boolean gate in the netlist by mapping these gates to a GLIFT logic library. This process is similar to mapping the synthesized design to a technology library and can be completed in linear time. Both the gate level netlist and GLIFT library can be described with standard HDL. Thus, GLIFT logic can be verified or tested using off-the-shelf EDA tools. This is a significant advantage of our method over others that require the designer to construct a formal model for the hardware design. And only after that, can you specify and prove properties on the design. GLIFT automatically provides that formal model making the security verification process significantly easier.

In the meantime, one needs to specify security properties that the IP core should adhere to. The security properties are translated into standard HDL assertion statements and verification constraints. We provide those assertions, constraints and the GLIFT logic to a standard hardware verification tool. If the design satisfies all the properties, it is free of Trojans that violate these security properties. Otherwise, formal verification will fail and provide a counter example that causes the security violation. With the counter example, we can perform functional testing on GLIFT logic, which enables us to determine the exact Trojan behavior and helps identify it in the entire design.

It is important to understand the different types of properties that can be checked by formal tools with and without GLIFT. Without GLIFT, formal tools can only check functional properties on netlists, e.g., if it is possible for some signals to take certain values. It is difficult to express security properties solely on the functional design. This is due to fact that the values do not carry information about how information flows. With GLIFT, data are associated with additional security labels, which enable reasoning about security of the design. GLIFT can precisely capture when information flow security properties related to confidentiality and integrity are violated, e.g., if sensitive data is multiplexed to a publicly observable output.

5.3 Deriving Security Theorems for Formal Proof

The security theorems are derived in two steps. The first step is classifying the signals in the hardware design into different security levels. As an example, we label secret data (such as the plaintext and key) as `HIGH` and publicly observable data (such as the encryption done signal) as `LOW` in confidentiality analysis; we label critical data (such as program counter) as `LOW` and untrusted data (such as input from UART port) as `HIGH` in integrity analysis.

The next step uses the labels to specify allowable (or forbidden) flows of information. These security properties are written to enforce that `HIGH` data should never flow to `LOW` data. Take cryptographic cores as an example, a first property to check for is that the key always flows to the ciphertext. To derive a security theorem for this property, we mark the key as `HIGH` while all the remaining inputs as `LOW` and check that the ciphertext is always `HIGH`. The following describes the security theorem for this property:

```
set key_t            HIGH
set DEFAULT_LABEL    LOW
assert cipher_t      HIGH
```

Consider another case where the key should never be altered. As a security theorem for this case, we label the key as `LOW` and all the remaining inputs as `HIGH` since this is a property related to integrity. We then check that the security label of the key register is always `LOW`. The following describes the security theorem for this case.

```
set key_t            LOW
set DEFAULT_LABEL    HIGH
assert key_reg_t     LOW
```

Similarly, one can derive other theorems for properties to be enforced. Since GLIFT logic can be described with standard HDL, the security theorems can be easily converted to assertion language statements and checked on GLIFT logic using standard hardware formal verification tools. This is a significant advantage of our method over existing work that requires description of the design in new semantics. Our method prevents the semantic differences of design and verification languages while minimizing the burden imposed on programmers.

6 Design Examples

We first use the *AES-T1700* and *RSA-T400* benchmarks from *trust-HUB.org* to demonstrate how our method can detect Trojans and reveal potentially malicious behaviors that may not be captured in pure functional testing and verification. We specify test and security constraints only for primary inputs and observe the security labels of primary outputs. We do not manipulate the internal registers within the benchmarks. We also use several *trust-HUB* benchmarks to show the performance and scalability of our method.

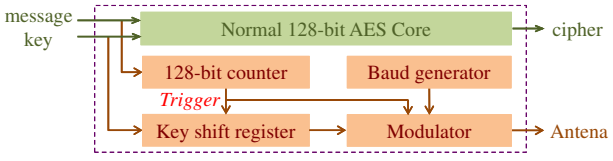

Fig. 3. The AES-T1700 benchmark contains a Trojan that leaks the key.

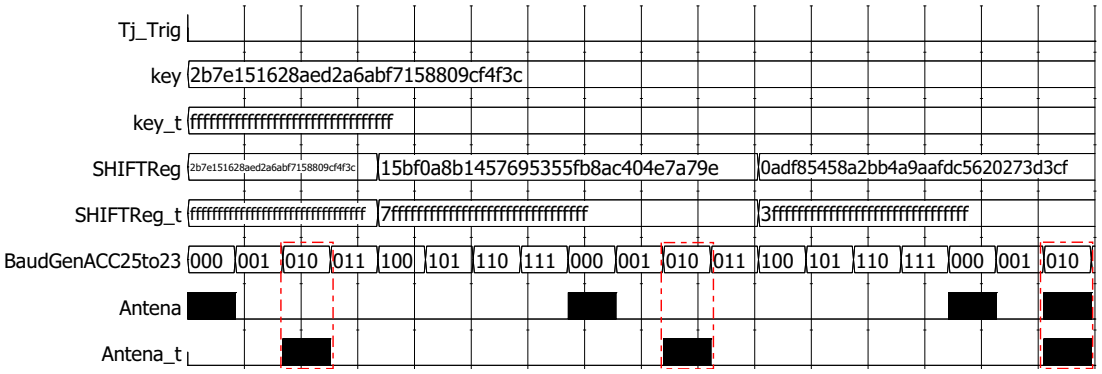

Fig. 4. We want to understand what is happening when key flows to the $Antena$ signal. This occurs when $Antena\_t$ is HIGH which occurs during the times noted by the red boxes.

6.1 The AES-T1700 Example

As shown in Fig. 3, the AES-T1700 benchmark contains a Trojan that leaks the key bits through a modulated RF channel. The Trojan is activated after $2^{129} - 1$ successive encryption operations. Once activated, the secret key is loaded into a shift register, whose least significant bit is modulated to leak through the RF channel. The probability of activating such a Trojan in functional testing is quite low.

To check the confidentiality property against key leakage, we mark the key as HIGH and all the remaining inputs as LOW. By asserting if an output can be HIGH, we can determine if the key flows to that output. In an initial analysis, we identify that both outputs of the benchmark, namely the ciphertext and the $Antena$ signal (note that this is the spelling of the signal in the file), can have HIGH labels. The subsequent analysis focuses on the $Antena$ output since it is normal for the key to flow to the ciphertext in a cryptographic function. We then use a SAT tool to prove if $Antena\_t$ ($Antena$'s label) is always LOW. The proof fails indicating that $Antena\_t$ could be HIGH. In other words, the $Antena$ output can possibly leak information about the key.

We use the *Mentor Graphics Questa Formal* tool to check if the internal registers in the model found by the SAT solver can meet the required conditions (note that SAT tools will stop at register boundary, i.e., it only performs combinatorial checks). We focus on the *SHIFTReg_t* register since *SHIFTReg* is the only register that can carry HIGH information in the model found by SAT solver. We formally prove that the *SHIFTReg_t* register is always type LOW using the *Questa* tool. The proof fails when the control point signal *Tj_Trig* is asserted.

For a better understanding, we simulate the GLIFT logic under the control point condition to capture how the key leaks to the $Antena$ output. The simulation result is shown in Fig. 4.

From Fig. 4, GLIFT indicates that the key leaks to the $Antena$ output when $BaudGenACC$[25:23] = 010 (Two additional signals $BaudGenACC$[15] = 1 and $BaudGenACC$[4] = 1 are not shown to simplify the figure) since $Antena\_t$ is 1 (indicating type HIGH). By observing the $Antena$ signal, we can see that in the first two red boxes, it is leaking a logical zero while in the third box it is transmitting a logical one. These are the lowest three bits of the key, which leaks to $Antena$ via a shift register, i.e., $SHIFTReg$. With enough simulation time, the entire 128-bit key will leak to the $Antena$ output.

Apart from these red boxed periods, there are additional transition activities in the $Antena$ signal when $BaudGenACC$[25:23] = 000. However, these are normal behaviors; they do not leak any information about the key since the taint label $Antena\_t$ is logical 0 (indicating type LOW). GLIFT precisely captures when and where key leakage happens while functional testing/verification cannot. Further, the Trojan design can be identified by tracking backwards from $Antena$ to the key using formal proofs upon the GLIFT logic.

6.2 The RSA-T400 Example

The BasicRSA-T400 benchmark contains a Trojan that replaces the key. Afterward, only the attacker can decrypt the ciphertext. This Trojan violates the integrity of the key, and thus we can write an information flow property to describe it formally. To check the integrity property against key replacement, we mark the key as LOW while all the remaining inputs are marked HIGH. By formally proving that the key register is always LOW, we can ensure that no such manipulation can be made. This is feasible but requires sufficient knowledge about the implementation details of the core, e.g., where the key registers are. We take a different approach in our test and mark discrete key bits as HIGH while the remaining inputs as LOW. By formally proving that the ciphertext is always HIGH, we can also guarantee that the key is never replaced. This is based on the observation that each single key bit should always flow to all digits of the ciphertext.

We then use the *Questa Formal* tool to prove that the taint label of ciphertext is always HIGH when it is valid. Formal proof results show that the taint label of the ciphertext can be LOW indicating that the key can possibly have no influence on the ciphertext, i.e., it has been replaced. The verification results also show that the ciphertext ready output of the RSA core can be HIGH. After a closer examination of the design, we figured out that the key leaked to the ready output through a timing channel. Although this is not a malicious scenario (Trojan), it does indicate the existence of a security flaw.

Table 1 summarizes the *trust-HUB* benchmarks that we tested. It shows the time used for GLIFT logic generation

TABLE 1
Designs from *trust-HUB* tested using our method. The GLIFT logic generation times are denoted in seconds. The proof times are in min:sec.

Benchmarks	Trojan behavior	Trigger	GLIFT gen. time	Proof time
AES-T100	Leaks the key through CDMA covert channel	Always on	2.22	06:48
AES-T1000	Leaks the key through CDMA covert channel	Single input	2.25	06:49
AES-T1100	Leaks the key through CDMA covert channel	Input sequence	2.25	06:46
AES-T1200	Leaks the key through CDMA covert channel	Counter	2.27	06:50
AES-T400	Leaks the key through modulated RF signal	Single input	2.45	06:44
AES-T1600	Leaks the key through modulated RF signal	Input sequence	2.50	06:37
AES-T1700	Leaks the key through modulated RF signal	Counter	2.51	06:51
RSA-T100	Leaks the key through ciphertext	Single input	0.08	05:19
RSA-T200	Replaces the key to disable encryption	Single input	0.08	05:36
RSA-T300	Leaks the key through ciphertext	Counter	0.09	16:31
RSA-T400	Replaces the key to leak the plaintext	Counter	0.08	14:01

and the proof time for Trojan detection. For the AES-T100, T1000, T1100 and T1200 benchmarks, our method has successfully bypassed the trigger conditions since the leakage points are XOR gates, which always allow the security labels to propagate independent of input values. For the AES-T400, T1600, T1700 and RSA-T100, T200 benchmarks, our method detects the Trojans and identifies the leakage points in less than 10 minutes.

From Table 1, our method can efficiently detect the Trojans in a number of *trust-HUB* benchmarks. It cannot capture all types of Trojans. Our method detects Trojans that can cause undesirable flow of information either through a maliciously modified data path or covert side channel. In addition, it only accounts for logical information flows and does not consider those that leak information through physical side channels. However, our approach represents a unique solution in the entire hardware Trojan detection method spectrum. It complements existing detection solutions by finding those Trojans that can cause violation of information flow security properties related to confidentiality and integrity.

7 Conclusion

We have demonstrated how information flow analysis can be used to detect hardware Trojans. Trojans are identified by formally proving information flow security properties related to confidentiality and integrity. By checking these security properties, we can uncover design flaws that can reveal hardware Trojans and other potential attack surfaces. Our techniques work directly on hardware described in standard language (Verilog, VHDL) and leverage off-the-shelf EDA tools for analysis.

The current method of specifying the security property is less than ideal. The security design flow would be significantly enhanced with a formal language that enables the security assessment team to specify important security properties, and maps those to information flow properties. Furthermore, many designs share common security properties. A library of such properties that can be easily leveraged across designs would be beneficial in enhancing the overall hardware security design flow. Finally, while Trojans represent a significant cause of concern for hardware security, unintentional design flaws can be equally harmful. Broadening the design techniques past the detection of Trojans to identify and mitigate non-malicious flaws is an important area of research.

8 Acknowledgments

This work was supported by the NSF under grant CNS-1527631.

References

[1] H. Salmani, M. Tehranipoor, and J. Plusquellic, "A novel technique for improving hardware trojan detection and reducing trojan activation time," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 20, no. 1, pp. 112–125, Jan 2012.

[2] J. Zhang, F. Yuan, L. Wei, Z. Sun, and Q. Xu, "Veritrust: Verification for hardware trust," in *Proc. of the 50th Design Automation Conference (DAC)*. New York, NY, USA: ACM, 2013, pp. 61:1–61:8.

[3] X. Wang, H. Salmani, M. Tehranipoor, and J. Plusquellic, "Hardware trojan detection and isolation using current integration and localized current analysis," in *Defect and Fault Tolerance of VLSI Systems (DFVS), 2008. IEEE International Symposium on*, Oct 2008, pp. 87–95.

[4] K. Hu, A. N. Nowroz, S. Reda, and F. Koushanfar, "High-sensitivity hardware trojan detection using multimodal characterization," in *Design, Automation Test in Europe Conference Exhibition (DATE), 2013*, March 2013, pp. 1271–1276.

[5] S. Narasimhan, D. Du, R. Chakraborty, S. Paul, F. Wolff, C. Papachristou, K. Roy, and S. Bhunia, "Hardware trojan detection by multiple-parameter side-channel analysis," *IEEE Trans. Comput.*, vol. 62, no. 11, pp. 2183–2195, Nov 2013.

[6] Y. Liu, K. Huang, and Y. Makris, "Hardware trojan detection through golden chip-free statistical side-channel fingerprinting," in *Proc. of the 51st Design Automation Conference (DAC)*. New York, NY, USA: ACM, 2014, pp. 155:1–155:6.

[7] M. Tehranipoor, H. Salmani, X. Zhang, X. Wang, R. Karri, J. Rajendran, and K. Rosenfeld, "Trustworthy hardware: Trojan detection and design-for-trust challenges," *Computer*, vol. 44, no. 7, pp. 66–74, July 2011.

[8] Y. Jin and Y. Makris, "Proof carrying-based information flow tracking for data secrecy protection and hardware trust," in *2012 IEEE 30th VLSI Test Symposium (VTS)*, April 2012, pp. 252–257.

[9] M. Tiwari, H. M. Wassel, B. Mazloom, S. Mysore, F. T. Chong, and T. Sherwood, "Complete information flow tracking from the gates up," in *Proc. of the 14th international conference on Architectural support for programming languages and operating systems*, ser. ASPLOS '09. New York, NY, USA: ACM, 2009, pp. 109–120.

[10] M. Tiwari, J. K. Oberg, X. Li, J. Valamehr, T. Levin, B. Hardekopf, R. Kastner, F. T. Chong, and T. Sherwood, "Crafting a usable microkernel, processor, and i/o system with strict and provable information flow security," in *Proceedings of the 38th Annual International Symposium on Computer Architecture*, ser. ISCA '11. New York, NY, USA: ACM, 2011, pp. 189–200.

[11] J. Oberg, S. Meiklejohn, T. Sherwood, and R. Kastner, "Leveraging gate-level properties to identify hardware timing channels," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 33, no. 9, pp. 1288–1301, Sept 2014.

[12] R. Karri, J. Rajendran, K. Rosenfeld, and M. Tehranipoor, "Trustworthy hardware: Identifying and classifying hardware trojans," *Computer*, vol. 43, no. 10, pp. 39–46, Oct 2010.