Detecting Hardware Trojans with Gate-Level Information-Flow Tracking

Wei Hu, University of California, San Diego
Baolei Mao, Northwestern Polytechnical University
Jason Oberg, Tortuga Logic
Ryan Kastner, University of California, San Diego

A method based on information-flow tracking uses gate-level logic to detect hardware Trojans that violate the confidentiality and integrity properties of third-party IP cores. Experiments on trust-HUB benchmarks show that the method reveals Trojan behavior and unintentional design vulnerabilities that functional testing cannot pinpoint.

With globalization of the hardware design and supply chain, hardware trustworthiness has become a major concern. Hardware design can involve multiple international teams with untrusted entities providing intellectual property (IP) cores. The resulting hardware often has unspecified functionality, which can be a conduit for information leaks or a backdoor that attackers can exploit.

These hidden functions can mask hardware Trojans, lightweight components carefully designed to activate only under rarely occurring conditions. Because they lie dormant until some point, Trojans are extremely difficult to detect during hardware design. External vendors might provide IP cores with built-in Trojans, and even internal IP cores could have Trojans—for example, inserted as a parting gift from a disgruntled employee. Although Trojans are intentionally created to be malicious, designers and tools can unintentionally introduce design vulnerabilities that could do equal harm when exploited. With modern hardware design becoming a

0018-9162/16/$33.00 © 2016 IEEE

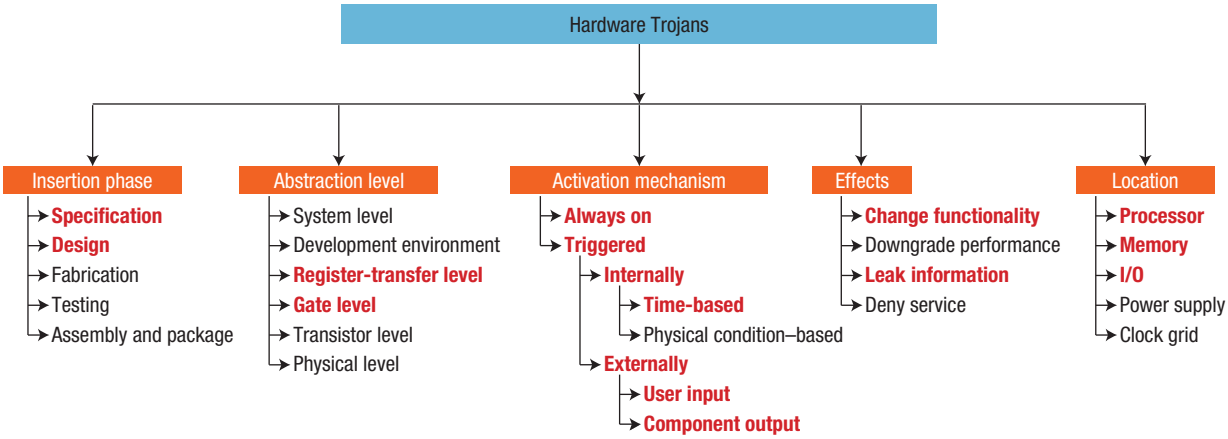

FIGURE 1. A taxonomy of hardware Trojans. Our method can detect Trojans at the register-transfer level and gate level inserted during specification or design as well as Trojans that have red boldface subcategories in the activation mechanism, effects, and location categories. (Source: R. Karri et al., "Trustworthy Hardware: Identifying and Classifying Hardware Trojans," *Computer*, vol. 43, no. 10, 2010, pp. 39–46.)

massive integration of in-house and external IP cores, it is easier for both Trojans and flaws to pass through design undetected.

As the sidebar "Hardware Trojan Detection Approaches" implies, detecting Trojans in IP cores is extremely difficult. Although much work has focused on detecting Trojans in fabricated hardware, the ideal strategy is to catch potential vulnerabilities that could signal a Trojan during the design phase, when they are much easier to eliminate or mitigate. As a step toward that solution, we developed a method that uses information-flow tracking (IFT) to detect Trojans in gate-level design. IFT has been widely deployed across the system stack—from programming languages and compilers to instruction-set architecture. It is an established formal method that can be used to prove important security properties, such as those related to confidentiality and integrity. We chose to use gate-level IFT (GLIFT) because, at this level, IFT precisely measures and controls all logical flows from Boolean gates.[1] GLIFT is also an established method that has been used to craft secure hardware architectures[2] and detect security violations from timing channels.[3] In our method, GLIFT formally verifies that an information flow adheres to security properties related to confidentiality and integrity. Counterexamples found during formal verification reveal harmful information flows that point to design flaws or malicious hardware Trojans that cause the system to leak sensitive information and violate data integrity. The designer can then take the appropriate action to correct the vulnerability that led to the violation. An understanding of malicious Trojan behaviors is useful in backtracking analysis to identify the Trojan design.

Our method works directly on hardware described in Verilog or the VHSIC Hardware Description Language (VHDL) and leverages off-the-shelf electronic design automation (EDA) tools for analysis. To assess its performance and scalability, we ran it on several trust-HUB benchmarks (www.trust-hub.org), which are designed for hardware Trojan classification and detection research. Results show that our method can detect Trojans in several trust-HUB benchmarks that cause the violation of information-flow security properties.

THREAT MODEL

The threat model on which our method is based rests on several assumptions:

- the third-party IP core might contain Trojans that are activated only under rare conditions;
- the Trojans are carefully designed and hard to activate through purely functional testing;
- when active, the Trojans leak sensitive information (such as the plaintext) to violate the integrity of critical data, for example, the secret key;
- when the Trojans are not active, the IP cores run normally and produce correct results; and
- the implementation details of the Trojan's trigger condition or payload are not known.

Our analysis requires access to the IP core's code at the register-transfer level (RTL) or to its gate-level netlist.

As Figure 1 shows,[4] the targeted Trojans are in the specification and design phases at the RTL or gate level. The Trojans can be either always on or triggered under specific conditions, such as through a single input, input sequence, or counter, and can cause a violation of the critical data's confidentiality or integrity properties. We assume that the attacker's primary

HARDWARE TROJAN DETECTION APPROACHES

Much work has been done on hardware Trojan detection, with existing methods being either invasive or noninvasive. Invasive methods insert test points in the design to increase observability, or they use reverse-engineering techniques to check for malicious design modifications in the physical layout. These methods are relatively expensive because they require highly specialized tools to access the chip's physical layout. In contrast, noninvasive methods do not need to modify the design but rather look for clues that reveal a Trojan's existence, such as faulty output, downgraded performance, and increased power consumption. Some methods try to capture these clues by functional testing; others reveal them through circuit-parameter characterization and the formal proof of properties related to information flow.

FUNCTIONAL TESTING

Testing or verification methods to detect Trojans identify suspicious signals in the circuit, typically those with an extremely low probability of switching to another logical state. Some methods use IC test methods to increase the Trojan's transition probability. For example, one research group used a procedure to insert a dummy scan flip-flop to help generate circuit transitions and reduce Trojan activation time.[1] Another group used functional testing to identify redundant circuits with low transition probability.[2]

Although these methods work for some Trojans, they might miss Trojans without a trigger signal. In general, methods that use testing to detect the existence of Trojans face many obstacles; IC testing poses difficult problems regardless of whether it considers logic that is intentionally difficult to activate.

CIRCUIT-PARAMETER CHARACTERIZATION

Several methods attempt to capture Trojan behaviors using side-channel signal analysis, the goal of which is to detect transient power and spurious delays arising from the Trojan's insertion in the design. For example, one research group proposed a current-integration methodology to reveal Trojan activity and used localized current analysis to identify the Trojan.[3] Others used circuit-parameter characterization to generate fingerprints or watermarks for the hardware design and compared them with those of a Trojan-free reference chip[4] or a small trusted region of the design. In the latter case, the trusted region was derived from running a trusted simulation model, measuring fabrication process parameters, and applying advanced statistical analysis.[5] Yet another group proposed a side-channel

goal is to learn sensitive information, so Trojans that cause a denial of service or downgrade performance are omitted. Leaks are assumed to be through logical attacks, not through power, electromagnetic, and other side channels.

MODELING CONFIDENTIALITY AND INTEGRITY PROPERTIES

The confidentiality property requires that secret information never leak to an unclassified domain, while the integrity property requires that untrusted data never be written to a trusted location. Hardware description languages (HDLs) are inadequate for enforcing such security properties because they specify only functionality. In contrast, information-flow analysis is able to model data movement.

Security requires knowing what should be protected, so a first step in modeling is to associate additional sensitivity information with data objects. In practice, these objects can have security labels at multiple levels according to sensitivity. For example, in a military information system, data can be labeled as unclassified, confidential, secret, or top secret.

The partial order between different security classifications can be defined using a security lattice. Let $L(\bullet)$ denote the function that returns the security label of a variable, which can be formalized as

$$A \rightsquigarrow B \Leftrightarrow L(A) \sqsubseteq L(B). \quad (1)$$

Equation 1 models confidentiality and integrity properties by specifying allowed information flows; in this case, information is allowed to flow from A to B if and only if A's security level is lower than or equal to B's. Under such a notion, both the confidentiality and

analysis method[6] that uses multiple parameters and leverages the relationship between dynamic current and maximum operating frequency to minimize the effect of process noise. Unfortunately, the increasing variation in the hardware manufacturing process and decreasing size of the Trojan payload work against these techniques.

FORMAL PROOF OF PROPERTIES

Some methods detect Trojans by formally proving security-related properties. A violation of a particular property indicates the existence of a Trojan. One method detects Trojan in cryptography hardware by formally proving security properties related to information flow.[7] Although it is a promising way to detect Trojans in third-party IP cores, it requires careful reasoning about where information can be declassified to reveal the Trojan payload's security label. This task can be challenging for hardware designers who lack security expertise.

In general, these methods require rewriting the hardware design in a formal language, which can increase design cost significantly, and most methods do not provide clues that reveal Trojan behavior, so they are not suitable for finding Trojans in the entire design. Our work addresses this problem by leveraging a precise gate-level information-flow model that can be described with a standard hardware description language and verified with off-the-shelf electronic design automation tools, which minimizes additional design cost.

References

1. H. Salmani, M. Tehranipoor, and J. Plusquellic, "A Novel Technique for Improving Hardware Trojan Detection and Reducing Trojan Activation Time," *IEEE Trans. VLSI Systems*, vol. 20, no. 1, 2012, pp. 112–125.
2. J. Zhang et al., "Veritrust: Verification for Hardware Trust," *Proc. ACM/IEEE 50th Design Automation Conf.* (DAC 13), 2013, article no. 61; doi: 10.1145/2463209.2488808.
3. X. Wang et al., "Hardware Trojan Detection and Isolation Using Current Integration and Localized Current Analysis," *Proc. IEEE Int'l Symp. Defect and Fault Tolerance of VLSI Systems* (DFVS 09), 2008, pp. 87–95.
4. K. Hu et al., "High-Sensitivity Hardware Trojan Detection Using Multimodal Characterization," *Proc. Design Automation Test in Europe Conf.* (DATE 13), 2013, pp. 1271–1276.
5. Y. Liu, K. Huang, and Y. Makris, "Hardware Trojan Detection through Golden Chip-Free Statistical Side-Channel Fingerprinting," *Proc. ACM/IEEE 51st Design Automation Conf.* (DAC 14), 2014, article no. 155, doi: 10.1145/2593069.259314.
6. S. Narasimhan et al., "Hardware Trojan Detection by Multiple-Parameter Side-Channel Analysis," *IEEE Trans. Computers*, vol. 62, no. 11, 2013, pp. 2183–2195.
7. Y. Jin and Y. Makris, "Proof Carrying-Based Information Flow Tracking for Data Secrecy Protection and Hardware Trust," *Proc. IEEE 30th VLSI Test Symp.* (VTS 12), 2012, pp. 252–257.

integrity properties can be modeled in a unified manner.

Our method uses a two-level security lattice LOW $\sqsubseteq$ HIGH. In a confidentiality analysis, sensitive data is labeled HIGH and unclassified data is LOW, but in an integrity analysis, critical data is labeled LOW and normal data is HIGH. For example, we label the secret key as HIGH in a confidentiality analysis but LOW in an integrity analysis.

METHODOLOGY

Our method has three main parts: GLIFT, detection of hardware Trojans, and the derivation of security theorems to formally prove properties.

GLIFT

GLIFT assigns a label (also known as a taint bit) to each bit of hardware design data. These assignments provide the basis for a model that designers can use to better understand how that data propagates through the design. Designers can then define security properties and use GLIFT to test or verify if the design adheres to them.

Suppose, for example, that the goal is to understand where information about the cryptographic key could flow. GLIFT assigns the label "confidential" to bits of the key, and the designer or test engineer can then write a property that precludes some part of the design from accessing those bits. The property might be, "Untrusted memory location *X* should never be able to ascertain any confidential information." This is the same as saying that the untrusted memory location can never be assigned a confidential label.

Because it associates each data bit with a security label, not a byte or word-level label, GLIFT can precisely account for information flow. In this

TABLE 1. Label propagation policy of conservative information-flow tracking (IFT) methods versus gate-level IFT (GLIFT) for AND-2.

Input labels	Conservative IFT			GLIFT		
	A	B	O	A	B	O
Both LOW	LOW	LOW	LOW	(LOW, –)*	(LOW, –)	(LOW, –)
A is LOW *B* is HIGH	LOW	HIGH	HIGH	(LOW, 0) (LOW, 1)	(HIGH, –) (HIGH, –)	(LOW, 0) (HIGH, –)
A is HIGH *B* is LOW	HIGH	LOW	HIGH	(HIGH, –) (HIGH, –)	(LOW, 0) (LOW, 1)	(LOW, 0) (HIGH, –)
Both HIGH	HIGH	HIGH	HIGH	(HIGH, –)	(HIGH, –)	(HIGH, –)

*–: value can be either logical 0 or 1.

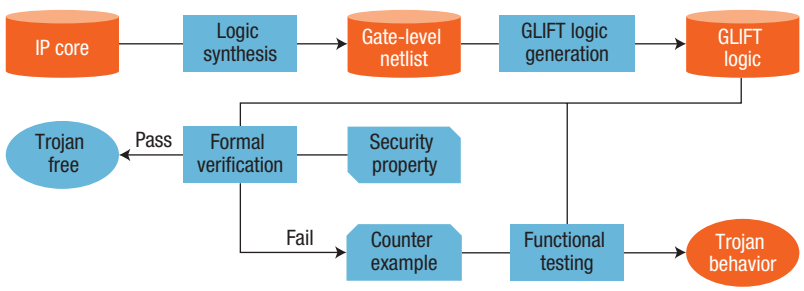

FIGURE 2. How our method detects hardware Trojans. The goal is to detect Trojans that violate information-flow security properties. A logic synthesis tool compiles the IP core's design to a gate-level netlist and then gate-level information-flow tracking (GLIFT) logic is automatically generated. Each gate is mapped to a GLIFT logic library, which can be completed in linear time. The GLIFT logic is formally verified against a security property that the designer has written. If it passes verification, there is no Trojan. If it does not, a counterexample is generated, which is used to functionally test the GLIFT logic to derive Trojan behavior.

case it knows that information flows from bit *A* to bit *B* if and only if *A*'s value influences *B*. Also, unlike other IFT methods, GLIFT accounts for the input data values when calculating the output label. Other IFT methods mark the output as HIGH if there is at least one HIGH input regardless of the input data values. With these characteristics, GLIFT can determine the output's security label more precisely than other IFT methods and thus more accurately measure actual information flows.

To better understand GLIFT, consider AND-2, a two-input AND gate whose Boolean function can be described as $O = A \bullet B$. Let A_t, B_t, and O_t denote the taints of *A*, *B*, and *O*, where $A, B \in \{0, 1\}$, and $A_t, B_t, O_t \in$ {LOW, HIGH} under some encoding scheme (for example, LOW = 0 and HIGH = 1) where

$$\text{LOW} \cdot \text{HIGH} = \text{LOW} \text{ and}$$
$$\text{LOW} + \text{HIGH} = \text{HIGH}. \quad (2)$$

Table 1 shows the label-propagation policy of conservative IFT methods and GLIFT for AND-2.

Conservative IFT methods typically set O_t to HIGH when either *A* or *B* is labeled HIGH. This policy is safe, in that it accounts for all possible flows of HIGH information, but it can cause many false positives (nonexistent flows) in information-flow measurement. To illustrate, let Secret be a 32-bit HIGH value. After performing

$$\text{Public} = \text{Secret} \cdot 0\text{x}01, \quad (3)$$

conservative methods will mark the entire Public as HIGH, indicating that 32 bits of information are flowing from Secret to Public.

However, GLIFT uses a more precise label-propagation method for AND-2. The output will be LOW (or HIGH) when both inputs are LOW (or HIGH). When there is only one LOW input and it 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.

In the Equation 3 example, 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.

These examples show that GLIFT considers both the security label and the actual value in its propagation, thus accounting for how an input value can influence output and more precisely measuring the actual information flows.

Hardware Trojan detection

Figure 2 shows the process of Trojan detection with our method.

Because both the gate-level netlist and GLIFT library can be described with a standard HDL, off-the-shelf EDA tools can be used to verify or test GLIFT logic. This feature contrasts sharply with other hardware Trojan detection methods, which require the designer to construct a formal hardware design model before specifying and proving properties. GLIFT

TABLE 2. Signal classification and labeling examples.

Confidentiality analysis			Integrity analysis		
Data type	**Example**	**Label**	**Data type**	**Example**	**Label**
Secret	Plaintext and key	HIGH	Critical	Program counter	LOW
Not secret	Clock, reset, and start of encryption signal	LOW	Noncritical	Input from open network or keyboard	HIGH

automatically provides that formal model, which makes security verification much easier.

The designer writes security properties, which are translated into standard HDL assertion statements and verification constraints and input along with the GLIFT logic to a standard hardware-verification tool. If the design satisfies all the properties, it is free of Trojans that violate those properties. Otherwise, formal verification will fail and provide a counterexample that causes the security violation. The counterexample enables functional testing on GLIFT logic, which determines the exact Trojan behavior. It is then possible to identify Trojan behavior across the design.

GLIFT is essential in broadening the properties that formal tools can check. Without GLIFT, formal tools can check only functional properties on netlists, such as the possibility that signal *X* can take value *Y*. It is difficult to express security properties solely for the functional design because values do not reveal how information flows. With GLIFT, data is associated with additional security labels, which enables reasoning about the design's security. GLIFT can precisely capture when information-flow security properties related to confidentiality and integrity are violated, such as whether sensitive data is multiplexed to a publicly observable output.

Deriving security theorems

Designers derive security theorems in two steps.

The first step is classifying the signals in the hardware design into different security levels. For example, they might use the classification and labeling in Table 2.

The next step is to use the labels to specify allowable (or forbidden) information flows. In this case, the designer would write a property to enforce the requirement that HIGH data should never flow to LOW data. The properties mark the input signals and specify which signals must be checked against their labels. As an example, the first property to check for cryptographic cores, is that the key always flows to the ciphertext. To derive a security theorem for this property, we mark the key as HIGH and all the remaining inputs as LOW and check that the ciphertext is always HIGH. Figure 3a describes the security theorem for this property.

Figure 3b describes a security theorem for a case in which the key should never be altered. The key is labeled LOW and all remaining inputs are HIGH because this property is related to integrity. To ascertain if the property holds, the designer checks that the key register's security label is always LOW.

Other theorems to enforce security properties can be similarly derived, easily converted to assertion language statements, and proved using standard hardware-verification tools. Again, this ease of use contrasts sharply with hardware Trojan detection methods that require each design to be described in new semantics. Our method has the dual advantage of eliminating these semantic differences and minimizing the burden on designers to write descriptions.

```
set key_t                HIGH
set DEFAULT_LABEL        LOW
assert cipher_t          HIGH
```
(a)

```
set key_t                LOW
set DEFAULT_LABEL        HIGH
assert key_reg_t         LOW
```
(b)

FIGURE 3. Sample security theorems for (a) a property that requires the key to always flow to the ciphertext and (b) a property that requires the key register to never be overwritten.

TROJAN DETECTION USING BENCHMARKS

Table 3 summarizes the trust-HUB benchmarks we tested, showing the time for GLIFT logic generation and Trojan detection. From specified test and security constraints, we observed the security labels of primary outputs but did not manipulate the benchmarks' internal registers. For the AES-T100, T1000, T1100, and T1200 benchmarks, our method successfully bypassed the trigger conditions because the leakage points were XOR gates, which always allow security labels to propagate regardless of input values. As the table shows, for the AES-T400, AES-T1600, AES-T1700, RSA-T100, and RSA-T200 benchmarks, our method detected Trojans and identified leakage points in less than 10 minutes.

Two examples, the AES-T1700 and RSA-T400 benchmarks, demonstrate how our method can detect Trojans and reveal potentially malicious behaviors that functional testing and verification might fail to capture.

AES-T1700 benchmark

As shown in Figure 4, the AES-T1700 benchmark contains a Trojan that leaks the key bits through a modulated RF channel. The Trojan activates after

TABLE 3. Designs from trust-HUB tested using our GLIFT method.

Benchmarks	Trojan behavior	Trigger	GLIFT logic-generation time (s)	Proof time (s)
AES-T100	Leaks the key through code division multiple access (CDMA) covert channel	Always on	2	408
AES-T1000	Leaks the key through CDMA covert channel	Single input	2	409
AES-T1100	Leaks the key through CDMA covert channel	Input sequence	2	406
AES-T1200	Leaks the key through CDMA covert channel	Counter	2	410
AES-T400	Leaks the key through modulated RF signal	Single input	2	404
AES-T1600	Leaks the key through modulated RF signal	Input sequence	3	397
AES-T1700	Leaks the key through modulated RF signal	Counter	3	411
RSA-T100	Leaks the key through ciphertext	Single input	<1	319
RSA-T200	Replaces the key to disable encryption	Single input	<1	336
RSA-T300	Leaks the key through ciphertext	Counter	<1	991
RSA-T400	Replaces the key to leak plaintext	Counter	<1	841

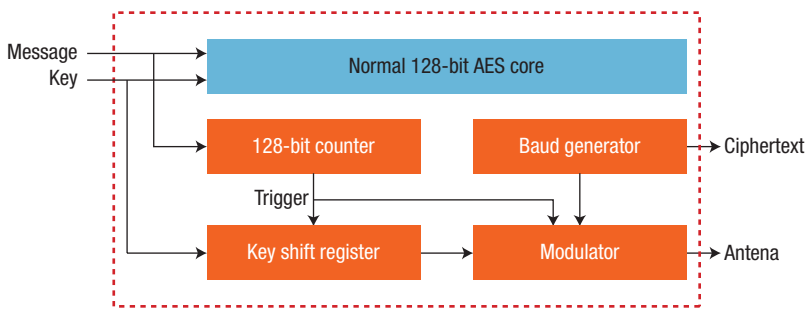

FIGURE 4. The AES-T1700 benchmark used to evaluate our method's performance. The benchmark contains a Trojan that leaks the key.

(2^{129} – 1) successive encryption operations. Once activated, it loads the secret key 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.

Property checking. To check the confidentiality property against key leakage, we marked the key as HIGH and all the remaining inputs as LOW. We then wrote an assertion statement that an output can be HIGH so that we could determine if the key flows to that output. In an initial analysis, we identified that both outputs, the ciphertext and Antena signal, can have HIGH labels. The subsequent analysis focused on the Antena output, as it is normal for the key to flow to the ciphertext in a cryptographic function. We then used a Boolean satisfiability (SAT) solver to prove that Antena_t (Antena's label) is always LOW. The proof failed, indicating that Antena_t could be HIGH and thus that Antena output could leak information about the key.

Next, we used Mentor Graphics' Questa Formal to check if the internal registers in the model found by the SAT solver could meet the required conditions. (SAT tools typically perform only combinatorial checks.) We focused on the SHIFTReg_t register because it was the only register that could carry HIGH information, according to the SAT solver's model. We used Questa Formal to formally prove that the SHIFTReg_t register was always LOW. The proof failed when the control point signal Tj_Trig was asserted.

Logic simulation. To better understand these findings, we simulated the GLIFT logic under the control-point condition to capture how the key leaks to the Antena output. Figure 5 shows the simulation results. GLIFT indicated that the key can leak to the Antena output. As the figure shows, when BaudGenACC[25:23] = 010, Antena_t is 1, which was equivalent to HIGH. There are also transitions in the Antena signal when BaudGenACC[25:23] = 000. However, these behaviors do not leak any information about the key because Antena_t is 0, indicating LOW.

The simulation results prove that GLIFT precisely captures when and where key leakage happened, which functional testing and verification could not do. A designer could use these results to identify the location

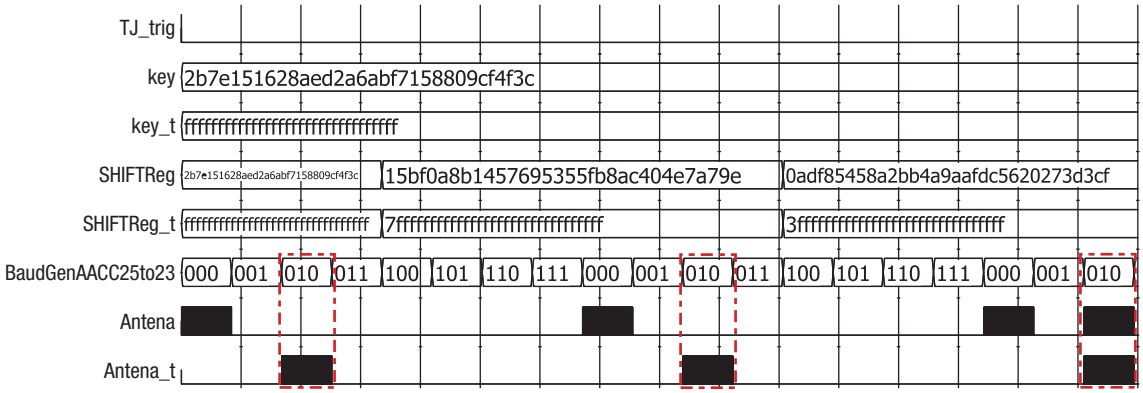

FIGURE 5. GLIFT logic simulation results. The simulation's goal was to reveal how the key leaks to the Antena output. The key flows to the Antena signal, when Antena_t is HIGH, denoted by the black rectangles within the boxes with dashed red lines, which denote the times when Antena_t is HIGH. To have no leakage, Antena_t must always be LOW.

of Trojans throughout the design by using formal proofs on the GLIFT logic to backtrack from Antena to the key.

RSA-T400 benchmark

The BasicRSA-T400 benchmark contains a Trojan that replaces the key, after which time only the attacker can decrypt the ciphertext. For this test we wrote an information-flow property to check the violation of the key's integrity. To check the integrity property against key replacement, we marked the key LOW and all the remaining inputs HIGH.

By formally proving that the key register is always LOW, we could ensure that no key replacement is possible. However, we would need knowledge about the core's implementation details, such as where the key registers are, which we did not have. Instead, we marked discrete key bits as HIGH and the remaining inputs as LOW. By formally proving that the ciphertext is always HIGH, we could also guarantee that the key could never be replaced because each single key bit should always flow to all digits of the ciphertext.

We again used Questa Formal to prove that the ciphertext security label is always HIGH when it is valid. Formal proof results showed that the label could be LOW, indicating that the key had been replaced. The verification results also showed that the RSA core's ciphertext-ready output can be HIGH. After examining the design more closely, we deduced that the key leaked to the ready output through a timing channel. Although this is not a malicious Trojan, the results revealed a security flaw that designers would have to address.

To our knowledge, our method is the first use of GLIFT for hardware Trojan detection and the first to provide a formal mechanism for detecting hardware Trojans that cause violations in information-flow security properties as well as a means of capturing Trojan behavior. Relative to other Trojan detection approaches, our method is much easier to use because there is no requirement to rewrite the design in a formal language.

The results in Table 3 show that our method efficiently detected Trojans that can cause undesirable information flow either through a maliciously modified datapath or covert side channel. It cannot capture all types of Trojans and accounts only for logical information flows, not those that leak information through physical side channels. However, our method holds a unique place in the spectrum of methods to detect hardware Trojans—namely, the identification of Trojans that can cause violation of information-flow security properties related to confidentiality and integrity.

We have already identified areas for future work. The security design process would benefit from a formal language that the security assessment team can use to specify important security properties and then map them to information-flow properties. Also, many designs have security properties in common. A library of shared properties could be easily leveraged across designs. Finally, although Trojans represent a significant cause of concern for hardware security, unintentional design flaws can be equally harmful. Broadening design techniques beyond Trojan detection to the identification and mitigation of nonmalicious design flaws is an important research area.

ACKNOWLEDGMENTS

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

REFERENCES

1. M. Tiwari et al., "Complete Information Flow Tracking from the Gates Up," *Proc. ACM 14th Int'l Conf. Architectural Support for Programming Languages and Operating Systems*, (ASPLOS 09), 2009, pp. 109–120.
2. M. Tiwari et al., "Crafting a Usable Microkernel, Processor, and I/O System with Strict and Provable Information Flow Security," *Proc. ACM 38th Int'l Symp. Computer*

ABOUT THE AUTHORS

WEI HU is a postdoctoral scholar in the Department of Computer Science and Engineering at the University of California, San Diego (UCSD). His research interests include hardware security, logic synthesis, and formal methods. Hu received a PhD in control science and engineering from Northwestern Polytechnical University (NPU). Contact him at weh040@ucsd.edu.

BAOLEI MAO is a doctoral student in the School of Automation at NPU. His research interests include hardware security and side-channel analysis. Mao received an MS in control theory and engineering from NPU. Contact him at maobaolei524@gmail.com.

JASON OBERG is a cofounder and chief executive officer of Tortuga Logic. His research interests include enhancing the industry approach to addressing security vulnerabilities during IC design. Oberg received a PhD in computer science from UCSD. Contact him at jason@tortugalogic.com.

RYAN KASTNER is a professor in the Department of Computer Science and Engineering at UCSD. His research interests include security, heterogeneous computing, and embedded systems. Kastner received a PhD in computer science from the University of California, Los Angeles. He is a member of IEEE. Contact him at kastner@ucsd.edu.

Architecture (ISCA 11), 2011, pp. 189–200.
3. J. Oberg et al., "Leveraging Gate-Level Properties to Identify Hardware Timing Channels," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, vol. 33, no. 9, 2014, pp. 1288–1301.
4. R. Karri et al., "Trustworthy Hardware: Identifying and Classifying Hardware Trojans," *Computer*, vol. 43, no. 10, 2010, pp. 39–46.

Selected CS articles and columns are also available for free at **http://ComputingNow.computer.org**.