Identifying and Measuring Security Critical Path for Uncovering Circuit Vulnerabilities

(Invited Paper)

Wei Hu
School of Automation
Northwestern Polytechnical University
Xi'an 710072, Shaanxi, China
Email: weihu@nwpu.edu.cn

Armaiti Ardeshiricham and Ryan Kastner
Department of Computer Science and Engineering
University of California San Diego
La Jolla, California 92093, USA
Email: {aardeshi, kastner}@ucsd.edu

***Abstract*—Hardware is an increasingly attractive attack surface since it controls low-level access to critical resources like cryptographic keys, personally identifiable information, and firmware. Unfortunately, it is difficult to assess the security vulnerabilities of a hardware design, which is a consequence of too few hardware security design tools and metrics. In this work, we describe important security qualities of hardware designs by formalizing hardware security metrics. We introduce the concept of security critical path and show how it can be used to evaluate properties related to confidentiality, integrity, and timing channels. We describe techniques for evaluating the security critical paths and assess their effectiveness in uncovering security flaws in a number of different hardware designs.**

I. Introduction and Motivation

Hardware is an attractive attack target as it controls low level access to critical resources that manage confidential information or important configuration data. There are many attacks that have targeted hardware vulnerabilities. For example, a recent attack exploited a subtle flaw in QualComm's TrustZone implementation to break the full disk encryption feature of Android devices [1]. Timing channels in shared resources like cryptographic core [2] and cache [3] leak the secret key and other confidential information. Backdoors have been identified in military level FPGA devices [4], opening up a door for attackers to retrieve the AES key for protecting configuration bitstream. There is suspicion of hardware Trojans residing in defense weapon [5], which allowed a remote attacker to switch off the system.

Hardware security vulnerabilities are notoriously hard to detect and eliminate. Subtle design flaws often hide in hard-to-cover corner cases [6]. Hardware Trojans are specifically developed to reside in difficult-to-find locations, e.g., don't care conditions that are out of the reach of functional verification tools [7]. Power and timing side channel attacks are extremely effective at uncovering confidential information [2], [8] and are difficult to mitigate.

Unfortunately, there is little automation for detecting hardware security vulnerabilities; companies typically fall-back upon manual inspection of the hardware design in order to identify the security vulnerabilities. This is largely the consequence of the lack of effective tools to evaluate security in the design flow. Hardware design tools cannot provide answers to important design questions such as if one design is more secure than another. Answering such questions requires security metrics, which are fundamental to developing automated techniques so that tools can evaluate security alongside performance, power consumption, resource utilization, and other common hardware design metrics.

There is a substantial lack of metrics for measuring hardware security. Our goal is to come up with a number of qualitative and quantitative metrics that can be evaluated during the design phase and allow quantitative measurement of security. Specifically, we transfer the idea of critical path from timing to security. We describe qualitative metrics for identifying circuit paths related to security and quantitative metrics for assessing the criticality of the security paths. We use quantitative measurements to evaluate the security paths in order to identify the security critical path. This gives designers helpful information to pinpoint potential security vulnerabilities and also allows comparison of relative security of different hardware designs.

In this work, we make a first attempt at formalizing the concept of *security critical path* in terms of confidentiality, integrity, and timing channel properties. We identify these paths using qualitative security metrics and quantify their security using quantitative measurements. More specifically, we make the following contributions:

- Formalizing the concept of security critical path and proposing criteria for characterizing these paths;
- Providing security metrics for identifying security critical paths and measuring their security;
- Presenting concrete design examples to demonstrate the effectiveness of the proposed solution.

The remainder of the paper is organized as follows. Section II describes our security evaluation methodology. In Section III, we formalize the concept of security critical path and propose criteria for characterizing such path. We provide security metrics for identifying and measuring security paths in Section IV. Section V shows examples on different hardware designs to demonstrate how the proposed solution can help uncover security vulnerabilities. We discuss potential research direction in Section VI and conclude in Section VII.

II. Hardware Security Evaluation Methodology

Figure 1 shows the design flow for identifying and quantifying security critical paths. Our method targets the early hardware design and testing phases as this is where we have the most flexibility to make changes to the design, which are often necessary to patch security flaws. Given register-transfer or gate-level netlist, our technique employs qualitative metrics to identifying security related paths. We then use qualitative metrics to evaluate the criticality of these security paths and rank the security paths based upon the quantitative measurement results. Assuming an effective metric, the most critical security path will have the highest probability of being associated with a security vulnerability.

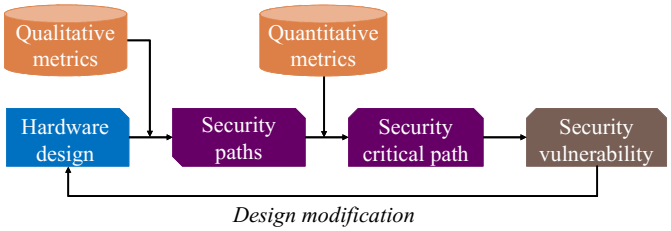

Fig. 1. The proposed security evaluation framework incorporates different qualitative and quantitative metrics to identify security-relevant paths and assess their criticality to reveal security vulnerabilities.

In the following two sections, we address two important elements in Fig. 1. We formalize the concepts related to security path and security critical path in Section III. We present qualitative and quantitative metrics for identifying and evaluating security paths in Section IV.

III. Security Critical Path

A hardware design netlist can be represented as a directed graph $G = (V, E)$, where the set of vertices $V = \{v^1, v^2, \cdots, v^n\}$ are the circuit nodes (i.e., logical elements) and the set of edges $E = \{e^1, e^2, \cdots, e^m\}$ defines the connections between the nodes. Here, a vertex can be a primary input/output, Boolean gate, or flip-flop when working at the gate level. At RTL, vertices represent adders, multipliers, multiplexors, registers and other coarser grained circuit elements. A *path* in digital hardware is a sequence of edges that connects a sequence of nodes. It starts from a source node (e.g., primary input), traverses through adjacent nodes under defined connection relations and ends at a destination node (e.g., a primary output or register).

Hardware designers already use path-based techniques to reason about circuit reliability and correctness. For example, path delay is an effective parameter for arguing about timing failure. Control and data flows paths are fundamental for modeling security related behaviors, and there are various techniques that can determine the existence of information flows via circuit paths [9]–[15]. In this work, we provide a formal framework to summarize and compare these existing techniques. And we show how to extend them to provide a measure of the path's "criticality".

In order to understand hardware security properties, we need to associate the circuit with security attributes and augment addition logic for security attribute propagation. A frequently used technique is hardware information flow tracking (IFT). Information flow models can be employed to systematically evaluate paths in digital circuits with respect to security. IFT tools extend the original circuitry with security attributes and logic to propagate these attributes. This means that all wires and registers are assigned with security labels, and additional tracking logic is added to propagate these labels. As we will show later, flexibility in crafting different label definition and propagation strategies results in a rich environment for security evaluation. A digital design that is enhanced with IFT features can be defined as follows:

Definition 1 - An IFT-Extended Circuit is a directed graph $G_{IFT} = (V_{IFT}, E_{taint})$ where the set of vertices $V_{IFT} = \{(v^1, v^1_{IFT}), (v^2, v^2_{IFT}), \cdots, (v^n, v^n_{IFT})\}$ and the set of edges $E_{taint} = \{(e^1, e^1_{taint}), (e^2, e^2_{taint}), \cdots, (e^m, e^m_{taint})\}$ are lists of tuples that capture security attributes alongside the Boolean functionality. Apart from the original functionality, all nodes and edges are associated with security labels and tracking logic for label propagation, respectively. The security labels (e^i_{taint}) and propagation logic (v^i_{IFT}) can be vectors, which allows the IFT model to tracks multiple forms of information flows through distinct channels. For example, the IFT model can track functional and timing flows in two different channels [15], [16]. We use the notation e^i_t to refer sensitivity label of e^i (i.e., being secret or untrusted), and e^i_τ to capture timing variations of e^i. Each security label (i.e., e^i_t or e^i_τ) could be a single bit or multiple bits to provide qualitative or quantitative analysis, respectively. We first define *security path* in terms of confidentiality, integrity, and timing channel based on a qualitative IFT model and then define the notion of *criticality* under quantitative metrics.

Definition 2 - A Security Path is any path in the circuit that starts from a node with sensitivity label of `HIGH` (i.e., being secret or untrusted) and ends in a node which is supposed to maintain a `LOW` (i.e., public or trusted) label. We denote a security path as $\varepsilon = \{e^i, \cdots, e^j\}$, where e^i_t is `HIGH` and e^j_t should be `LOW` to adhere to confidentiality or integrity policies. The set of all security paths are represented with ξ. We use the notation $\varepsilon^i.e^j$ to refer to edge e^j from path ε^i. The existence of *security path* indicates potential data leakage or unprivileged access via that path. Similarly, we can define a security path that points out leakage through a timing side channel, exploitable even in cases when the attacker cannot directly read the leaked data.

Definition 3 - A Timing Channel Security Path is any path in the circuit that starts from a sensitive node and ends in a node that should be timing invariant. We denote such a path as $\varepsilon_\tau = \{e^i, \cdots, e^j\}$, where e^i_t is `HIGH` and e^j_τ should be `LOW`. The set of all such paths are represented with ξ_τ. Simply said, timing channel security path is a type of security path corresponding to timing information leakage.

Once identified, the set of security paths pinpoint vulnerable places in the design. However, they fall short in indicating the

amount of information which may leak through these paths and the potential severity of the security flaw. Thus, we seek quantitative methods, which enable comparison between paths.

Definition 4 - A Security Critical Path is defined under a quantitative security metric $\mathscr{Q}$ and refers to a path that is the most vulnerable to security attacks among the set of security paths. Security path ε_i is the critical path *if and only if* the following holds:

$$\mathscr{Q}(\varepsilon^i.e_t^j) \geq \mathscr{Q}(\varepsilon^k.e_t^j)\ \forall \varepsilon^k \in \xi \tag{1}$$

Definition 5 - A Timing Channel Security Critical Path is the most vulnerable path with respect to timing side channel attack among all the timing channel security paths analyzed with a quantitative security metric $\mathscr{Q}$. Security path ε_τ^i is the timing channel security critical path *if and only if*

$$\mathscr{Q}(\varepsilon_\tau^i.e_\tau^j) \geq \mathscr{Q}(\varepsilon_\tau^k.e_\tau^j)\ \forall \varepsilon_\tau^k \in \xi_\tau \tag{2}$$

To better understand the notion of criticality, consider the AES-T1700 benchmark from *Trust-HUB.org*. This benchmark contains a hardware Trojan triggered by a 128-bit counter. When activated, the key is loaded into a shift register and modulated to leak through an unused pin, namely *Antena* (as spelled in the benchmark).

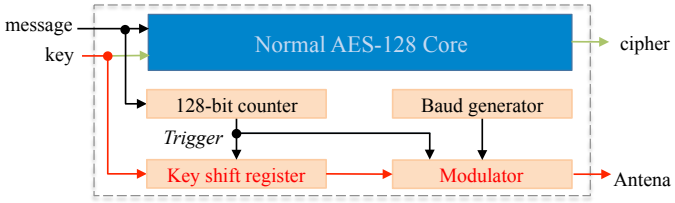

Fig. 2. Architecture of the AES-T1700 benchmark from Trust-HUB. The benchmark contains a hardware Trojan that modulates key bits to leak through a RF (radio frequency) channel.

We label the *key* as `HIGH` in order to understand its security (similar analysis can be performed for the *message*). The security label of *cipher* can be declassified to `LOW` considering that it is protected by the AES algorithm and also publicly observable. Here, all circuit paths originating from the *key* to *cipher* and *Antena* are security paths, i.e., the paths in green and red. However, only the path in red would be considered as security critical. This path feeds individual key bits to *Antena*. The *key* can be retrieved by decoding the RF signal transmitted from this pin. This path is more vulnerable to security attack, considering that the ciphertext is generated by a correct AES implementation and thus secure. In the next section we discuss how the red paths can be distinguished from the green ones employing quantitative security metrics.

IV. Security Metrics

A. Qualitative Metrics

1) Graph Connectivity: Connectivity analysis is a general technique for path search on graphs. To employ it as a metric for security path search, we need to extend the circuit graph with security attributes and constrain the search to start from a `HIGH` node and end at a `LOW` one.

Graph connectivity can be an efficient metric for calculating all security paths, considering that the complexity of the graph search problem is bounded by $O(n^2)$. However, it may report non-existent paths since it does not take into account the functionality of circuit nodes such as a path disallowed by a security policy (e.g., access to a protected memory space) or under certain configuration (e.g., scan a disable debug port). In design practice, this method can be used to profile all potential security paths, providing a good starting point for further analysis.

2) Miter Based Equivalence Checking: Miter based equivalence checking provides a precise solution for determining if there is a path between two circuit nodes. Given a Boolean function $F = f(x^1, x^2, \cdots, x^n)$, we can construct two functions $F_0 = f(x^1, x^2, \cdots, x^i = 0, \cdots, x^n)$ and $F_1 = f(x^1, x^2, \cdots, x^i = 1, \cdots, x^n)$ with x^i evaluated to zero and one respectively. By formally checking if F_0 and F_1 are equivalent, we can determine if there is a path from x^i to F. To check for a path starting from an internal node, we need to cut the input cone to that node and make its output a new primary input.

Miter based equivalence checking has been employed by existing secure path verification tools such as *Cadence Jasper* and *Mentor Graphics SecureCheck* [9]–[11]. Its computational complexity is typically $O(2^n)$. However, it only indicates if there is a path between two circuit nodes without reporting path details. It can be used in combination with graph connectivity analysis to precisely determine all security paths.

3) Taint and Information Flow: Taint and information flow analysis extends the original digital circuit with security attributes and logic for performing attribute propagation. Given a `HIGH` source node, this metric can determine which nodes `HIGH` information could possibly flow to through testing or verification. When combined with graph search, it can be used to calculate all potential security paths.

There are various information flow models that can be used as our metric. A conservative information flow model always propagate security attribute as it is. Specifically, the output of a node will be marked as `HIGH` whenever there is a `HIGH` input to that node. Similar to graph connectivity analysis, it does not consider the functionality of circuit nodes either. More precise information flow models such as gate and register transfer level information flow tracking [12], [13] take into account the functionality of nodes and input values when determining if `HIGH` information could propagate under the given input condition. These precise models can reduce the number of false paths indicated by a conservative one.

We can also associate different security attributes and labels with different information flow models. Binary security labels are effective in identifying security paths related to confidentiality and integrity properties [12]. Multi-level security labels allow a finer classification of security classes and provide more insights about information flow behaviors [14]. In order to capture a timing channel security path, we need to extend the

binary IFT model with a timing label and additional logic for tracking timing information flows [15].

Information flow analysis is a frequently used technique for verifying security properties. The computational complexity of this metric is typically $O(2^{2n})$. Practical information flow analysis methods usually allow a certain amount of false positives to reduce verification cost.

B. Quantitative Metrics

Qualitative metrics are effective for calculating the set of security paths. However, we also need quantitative metrics for evaluating the security of these paths and identifying the security critical path.

1) Path Length: Path length is a straightforward metric. Unlike timing, where longer paths tend to cause a timing failure, shorter paths are more likely to be security critical. Generally, as data propagate through more circuit nodes, we would learn less information about the original input. This can be confirmed by quantitative measurements using another metric in the following subsection.

Path length is a simple metric that can be computed in linear time. However, unlike path delay, which is the sum of wire and node delays along that path, there is no established connection between path length and security. A major reason is that our circuit model does not have a parameter for security similar to delay. We either need to reveal a connection exists or rely on more effective metrics.

2) Mutual Information: Mutual information is an information theoretic measure that describes the amount of information we can learn about one signal by observing another. Figure 3 shows the different amounts of information we can learn about input A at different circuit nodes. Our mutual information measurements are in *bits*.

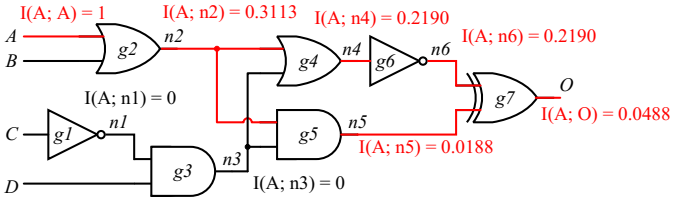

Fig. 3. Mutual information between input A and signals at different circuit nodes. It decreases as the input propagates through more gates.

Mutual information is more a measurement of mutual dependency of two variables. It correlates to the probability for one variable to take a certain value when a specific value is observed at another. Mutual information has been shown to be an effective measurement for quantifying information leakage [17]. However, it is usually estimated from a large number of discrete samples. The accuracy highly depends on effective and precise estimation of distributions, which can be hard for variables with a huge state space.

A possible solution is to estimate mutual information in a compositional manner. We can construct a library for Boolean gates, which defines the mutual information between each input and the output under different input distributions. This can be used as a metric of how much information would propagate on average by each gate. Given a security path, a multiplication of mutual information for all the gates along the path would be used as a quantitative metric for evaluating security. Such compositional mutual information can be easily estimated. It also reveals the trend that we can learn less information about the original input after it propagates through more gates. However, it does not take into account of the distribution of the inputs at internal nodes and thus can be less accurate than mutual information measurement.

3) Taint Probability: Taint probability is defined as the probability for a signal to have a `HIGH` security label. It measures the controllability of the `HIGH` input on that signal. Figure 4 shows the taint probability of signals along all the security paths. Here, output taint labels are calculated using the GLIFT method [12]. The output of a gate will be marked as tainted (i.e., `HIGH`) if and only if at least one tainted input has an influence at the output. The output of the *XOR* gate has a higher taint probability than its inputs. This is caused by a reconvergent fanout loop.

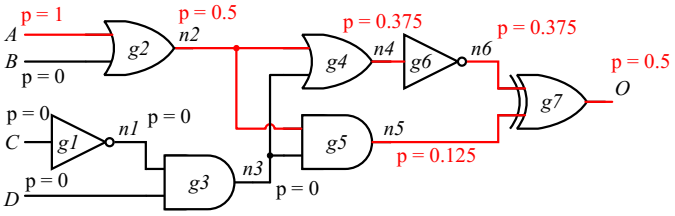

Fig. 4. Taint probability measurements between input A and signals at different circuit nodes.

As mentioned, the GLIFT model was employed for taint propagation in this example. Sometimes, we may want to allow some relaxation in the precision of label propagation policy. Take the two-input AND gate as an example, we may always allow tainted information in one of the inputs to propagate while the other input still adheres to the GLIFT label propagation policy, or in a conservative manner, always allow tainted information in both inputs to propagate regardless of the input condition. This enables various tradeoffs between the quality and complexity of this metric.

V. Design Examples

In this section, we demonstrate how security critical paths can help uncover hardware security vulnerabilities using realistic design examples.

A. Design Flaw

Consider an AES design from *dpacontest.org* as shown in Fig. 5. The AES core is functionally correct. However, it contains a design flaw that feeds the intermediate encryption results to the ciphertext, leaking a significant amount of information about the key and plaintext.

We first label the *key* as `HIGH` and use qualitative metrics to calculate security paths. Through graph connectivity or

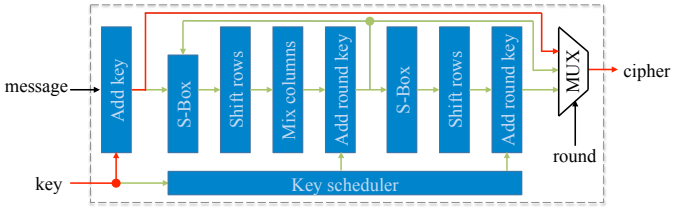

Fig. 5. Architecture of an AES core from dpacontest.org.

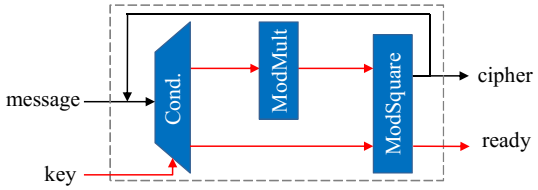

Fig. 6. Architecture of an RSA core from opencores.org.

information flow analysis, we can identify multiple security paths from the *key* to the *cipher*. The shortest path feeds $key\ xor\ message$ directly to the output. Other paths assign the intermediate encryption results after different rounds to *cipher*. We then mark the *message* as `HIGH` and perform similar security path search.

Using path length as a simple quantitative metric, we suspect that the shortest path (i.e., the one in red) is the security critical path. We further use mutual information to measure the amount of information we can learn about a key or plaintext byte after different AES rounds[1]. In our test, we set plaintext (or the key) to constant when performing mutual information measurement for the key (or plaintext) and run 100000 random tests. Table I shows the mutual information between the lowest key byte (i.e., *key[7:0]*) or plaintext byte (i.e., *message[7:0]*) and the lowest two encryption intermediate result bytes (i.e., *cipher[7:0]* and *cipher[15:8]*). We see similar mutual information results for other key and plaintext bytes in our test.

From Table I, the mutual information between *key[7:0]* (or *message[7:0]*) and *cipher[7:0]* is close to 8 bits after the first add key operation. This means that we can almost learn 100% information about this key byte, which is caused by the design flaw associated with the security critical path. After the first round of encryption, the mutual information measurement will quickly decrease to about 0.54 bits. This is the result of the confusion and diffusion introduced by the AES algorithm. The mutual information between *key[7:0]* and the second intermediate encryption result byte is low, indicating mutual independency. Mutual information measurements have confirmed that the path in red is security critical.

B. Timing Channel

Figure 6 shows an RSA core from *opencores.org*. The core is also functionally correct. However, the core contains a timing channel in that it takes different number of clock cycles to encrypt different messages.

We label individual key bits as *HIGH* in our analysis. Qualitative metrics such as hardware IFT (e.g., GLIFT and RTLIFT) indicates that there are security paths from the key bits (except for the least significant key bit) to both the *ciphertext* and *ready* signals. The security paths to *ciphertext* is desirable because the key should have an effect on encryption result. However, the key is not flowing to the ready signal in a functional manner. Instead, it determines when *ready* should be asserted. Using an IFT model extended with a timing label [15], we have detected a timing channel security path from the key to the *ready* signal.

We then assess the criticality of such a timing channel security path for six different 128-bit RSA architectures using a quantitative metric. This allows a comparison of their security in terms of timing leakage. We measure the mutual information between individual key bits and encryption time (i.e., when *ready* is asserted) under 6000 allowed RSA key pairs. Here, the *R-2-L* implementation performs repeated square-and-multiply from right to left. The L-2-R-always design inserts a dummy multiply to balance the conditional branch of the RSA algorithm. The Power-ladder algorithm carefully redesigns the algorithmic flow so that both modular multiply and square are performed regardless of the current key bit. The Montgomery design uses the Montgomery multiplier. The Base-blind algorithm introduces a random number to mask the messages. The Exponent blind algorithm introduces a random number to protect the private key.

From Fig. 7, the R-2-L design tends to leak more information about the key since it is unprotected. The Exponent-blind design better protects the key than the Base-blind design. The Montgomery design has the smallest leakage because the modular multiplication runs in constant time so that the runtime variation caused by the plaintext can be eliminated.

From the results, quantitative metric allows a comparison of relative security, which could provide an answer to important design questions such as if one design is more secure and if one mitigation technique is more effective than another. This is an important step for developing automated hardware security evaluation tools.

VI. Potential Research Directions

A. Extending the Concept of Security Path for More Security Properties

Currently, we have formalized the idea of security path and security critical path for confidentiality, integrity and timing channel security properties. However, security vulnerabilities such as power side channel and hardware Trojan fall out of this scope. A potential research vector is to extend the concept of security path for a wider range of security properties. A tie between power side channel as well as hardware Trojan and security is switching probability, which may provide a possible solution.

[1]We measure mutual information at the byte level so that we can collect enough samples to accurately estimate the distributions of the key, plaintext and intermediate encryption result segments.

TABLE I

MUTUAL INFORMATION BETWEEN THE LEAST SIGNIFICANT KEY (OR PLAINTEXT) BYTE AND THE LOWEST TWO ENCRYPTION INTERMEDIATE RESULT BYTES AFTER DIFFERENT ROUNDS OF ENCRYPTION (BITS).

Mutual information	AddKey	r1	r2	r3	r4	r5	r6	r7	r8	r9	r10
I(key[7:0]; cipher[7:0])	7.9979	0.5410	0.5466	0.5474	0.5441	0.5459	0.5494	0.5434	0.5428	0.5395	0.5436
I(key[7:0]; cipher[15:8])	0.5460	0.5463	0.5438	0.5420	0.5454	0.5488	0.5459	0.5420	0.5428	0.5473	0.5396
I(message[7:0]; cipher[7:0])	7.9979	0.5376	0.5456	0.5450	0.5431	0.5438	0.5435	0.5435	0.5437	0.5452	0.5461
I(message[7:0]; cipher[15:8])	0.5460	0.5466	0.5473	0.5466	0.5440	0.5466	0.5415	0.5438	0.5425	0.5423	0.5412

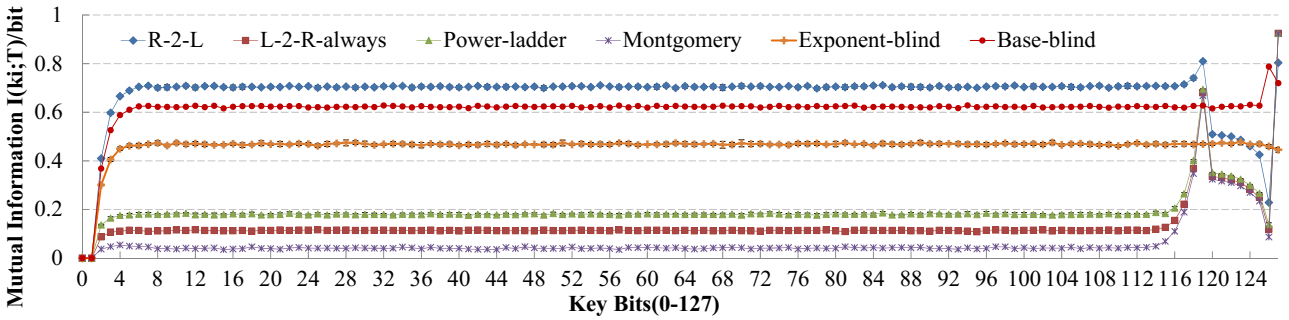

Fig. 7. Mutual information between key bits and the runtime measurements for different 128-bit RSA implementations.

B. Security Metrics

Security metrics are important factors for identifying and evaluating security paths. However, our current security metric library still has a limited capacity. Deriving more effective security metrics for both qualitative and quantitative assessment of security is an essential element for the development of automated hardware security tools. In addition, we need effective ways for calculating the security metrics, e.g., precise estimation of statistical and information theoretic measures with a minimum number of samples.

VII. CONCLUSION

In this work, we propose to formalize the ideas of security path and security critical path for understanding the security of hardware designs and uncovering circuit vulnerabilities. We provide qualitative metrics for calculating security paths and quantitative metrics for evaluating the criticality of these paths. This is an important first step towards an automated hardware security evaluation framework that can help hardware designers pinpoint security vulnerabilities.

ACKNOWLEDGMENT

This work was supported in part by the NSF under grant CNS-1527631 and by the Fundamental Research Funds for the Central Universities under grant 3102017OQD094.

REFERENCES

[1] Bits, Please, "Extracting qualcomm's keymaster keys - breaking android full disk encryption," in *http://bits-please.blogspot.com/2016/06/extracting-qualcomms-keymaster-keys.html*, June 2016.

[2] P. C. Kocher, "Timing attacks on implementations of diffie-hellman, rsa, dss, and other systems," in *International Cryptology Conference on Advances in Cryptology*, London, UK, 1996, pp. 104–113.

[3] D. J. Bernstein, "Cache-timing attacks on aes," *Vlsi Design IEEE Computer Society*, vol. 51, no. 2, pp. 218 – 221, 2005.

[4] S. Skorobogatov and C. Woods, *Breakthrough Silicon Scanning Discovers Backdoor in Military Chip*. Springer, 2012, pp. 23–40.

[5] S. Adee, "The hunt for the kill switch," *Spectrum, IEEE*, vol. 45, no. 5, pp. 34–39, May 2008.

[6] J. Zhang, F. Yuan, and Q. Xu, "Detrust: Defeating hardware trust verification with stealthy implicitly-triggered hardware trojans," in *ACM SIGSAC Conference on Computer and Communications Security*. New York, NY, USA: ACM, 2014, pp. 153–166.

[7] W. Hu, L. Zhang, A. Ardeshiricham, J. Blackstone, B. Hou, Y. Tai, and R. Kastner, "Why you should care about dont cares: Exploiting internal dont care conditions for hardware trojans," in *International Conference on Computer-aided Design*. New York, NY, USA: IEEE, 2017.

[8] E. Brier, C. Clavier, and F. Olivier, *Correlation Power Analysis with a Leakage Model*. Springer Berlin Heidelberg, 2004, pp. 16–29.

[9] J. Mazzawi and Z. Hanna, "Formal analysis of security data paths in rtl design," in *International Conference on Hardware and Software: Verification and Testing*. Springer-Verlag, 2013, pp. 7–7.

[10] G. Cabodi, P. Camurati, S. F. Finocchiaro, C. Loiacono, F. Savarese, and D. Vendraminetto, "Secure path verification," in *IEEE International Verification and Security Workshop (IVSW)*, July 2016, pp. 1–6.

[11] Mentor Graphics, "Questa secure check - exhaustive verification of secure paths to critical hardware storage." [Online]. Available: https://www.mentor.com/products/fv/questa-secure-check

[12] M. Tiwari, H. M. Wassel, B. Mazloom, S. Mysore, F. T. Chong, and T. Sherwood, "Complete information flow tracking from the gates up," in *the 14th International Conference on Architectural Support for Programming Languages and Operating Systems*, 2009, pp. 109–120.

[13] A. Ardeshiricham, W. Hu, J. Marxen, and R. Kastner, "Register transfer level information flow tracking for provably secure hardware design," in *Design, Automation & Test in Europe*, 2017, pp. 1695–1700.

[14] W. Hu, D. Mu, J. Oberg, B. Mao, M. Tiwari, T. Sherwood, and R. Kastner, "Gate-level information flow tracking for security lattices," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 20, no. 1, 2014.

[15] A. Ardeshiricham, W. Hu, and R. Kastner, "Clepsydra: Modeling timing flows in hardware designs," in *IEEE/ACM International Conference on Computer-Aided Design*, 2017.

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

[17] B. Mao, W. Hu, A. Althoff, J. Matai, J. Oberg, D. Mu, T. Sherwood, and R. Kastner, "Quantifying timing-based information flow in cryptographic hardware," in *the International Conference on Computer Aided Design*, 2015, pp. 552–559.