On Secure Data Flow in Reconfigurable Scan Networks

Pascal Raiola*, Benjamin Thiemann*, Jan Burchard‡, Ahmed Atteya§, Natalia Lylina§, Hans-Joachim Wunderlich§, Bernd Becker*, Matthias Sauer*

*University of Freiburg, Germany
‡Mentor, a Siemens Business, Hamburg, Germany
§University of Stuttgart, Germany

***Abstract*—Reconfigurable Scan Networks (RSNs) allow flexible access to embedded instruments for post-silicon test, validation and debug or diagnosis. The increased observability and controllability of registers inside the circuit can be exploited by an attacker to leak or corrupt critical information.**

Precluding such security threats is of high importance but difficult due to complex data flow dependencies inside the reconfigurable scan network as well as across the underlying circuit logic.

This work proposes a method that fine-granularly computes dependencies over circuit logic and the RSN. These dependencies are utilized to detect security violations for a given insecure RSN, which is then transformed into a secure RSN.

Experimental results demonstrate the applicability of the method to large academical and industrial designs. Additionally, we report on the required effort to mitigate found security violations which also motivates the necessity to consider the circuit logic in addition to pure scan paths.

***Keywords*—Reconfigurable Scan Network, Hardware Security, Data Dependency, IEEE Std 1687**

I. Introduction

Today's complex circuits employ a great variety of on-chip functional and non-functional instrumentation to facilitate amongst others on-chip diagnosis, post-silicon validation, bring-up and production test, therefore allowing test/debug access to the circuit [1]. In order to cope with the complexity of connecting these instruments, reconfigurable scan networks *(RSNs)* as standardized by e.g. IEEE Std 1149.1-2013 and IEEE Std 1687-2014 are increasingly deployed in industrial designs. Such RSNs allow for the configuration of the active scan path and hence provide flexible and scalable access to embedded instruments.

However, an attack might not only be performed on the underlying circuit logic, but could in addition also use the flexibility and powerful observability and controllability properties of an RSN to potentially compromise or read out sensitive data [2]. Such an attack must be prevented to enable secure in-field operation.

In this work, we present a method that analyzes and resolves security violations throughout both the RSN and the underlying logic circuit.

The potential contrariety between testability and security has already been discussed for conventional design-for-test infrastructures such as standard scan chains [3]–[5]. The data flow in conventional scan infrastructure is static and thus of much lower complexity than in RSNs.

Approaches to defend against attacks on reconfigurable scan infrastructure have been presented in various works. The authors of [6], [7] utilize an authorization technique to skip certain scan segments, unless a secret key is provided; in [8] an authorization instrument is used for access management. Secure test wrappers are introduced e.g. by [9], where a technique is proposed, which functions without any hard-coded secrets in the design.

Various techniques to obfuscate the scan data have been introduced: E.g. the scan chain is partitioned into sub-chains and the access to the sub-chains is pseudo-randomized [10] or obfuscation is achieved with the use of state-dependent flip-flops [11]. The authors of [12] present a compaction approach where the full test response is compacted to one bit via on-chip test comparison.

An attacker must be prevented from controlling obfuscated data flow, as e.g. overwriting sensitive data with obfuscated data might cause a system to show unspecified and thus potentially insecure behavior. Therefore the technique presented in this paper prevents *any* data traversal over an insecure data path.

In [13] a filter is introduced locally at the interface of the RSN (TAP), that only allows a precomputed set of secure scan-in access sequences. [14] introduces a filter, that prevents forbidden accesses, by monitoring the security requirements online.

Security violations can occur for a pair of scan flip-flops or even scan registers which cannot be separated by scan path configuration, forcing a filter to make every such pair inaccessible for debug and diagnosis. In contrast the proposed method guarantees to include all scan flip-flops in the final secure reconfigurable scan network.

The use of e-fuses or a wafer saw to fully deactivate the scan path has been proposed e.g. in [15], making any in-field use of the scan infrastructure impossible and is therefore not considered in this work.

As already mentioned before, we present a method that analyzes data flow throughout both the RSN and the underlying logic circuit, in order to detect security violations and to structurally transform a given (insecure) RSN into a *secure* RSN based on a user-given security specification [16], [17]. Paths that may leak or corrupt sensitive data, because they pass through instruments, which do not guarantee an adequate trustworthiness, are identified and prohibited.

The proposed technique utilizes the method for calculating complex dependencies introduced in [18] and lifts techniques from [17] to the challenge of complex underlying circuit logic.

The applicability of the presented method is demonstrated by an experimental evaluation using academical and industrial RSN benchmarks. As we will demonstrate, the computational runtime is feasible for all considered benchmarks. Additionally, we report on the required effort to mitigate found security violations which also motivates the necessity to consider the circuit logic in addition to pure scan paths.

To the best of our knowledge, this work is the first to face the challenge of computing the functional data flow through both the reconfigurable scan infrastructure and the whole underlying circuit as well as to provide a solution to mitigate security threats introduced by complex data paths leading through the circuit logic and the scan infrastructure.

The remainder of the paper is organized as follows: Section II summarizes the background on the underlying model. The method to analyze complex data dependencies and to prevent given security threats by transforming the RSN is explained in greater detail in Section III. Section IV provides the experimental results. Section V concludes the paper.

II. Underlying Model

A. *Reconfigurable Scan Network (Running Example)*

The focus of this work lies on Reconfigurable Scan Networks (RSNs). An example RSN is illustrated in Figure 1 (blue box). The RSN consists of two scan multiplexers and 14 scan flip-flops (`SF1`

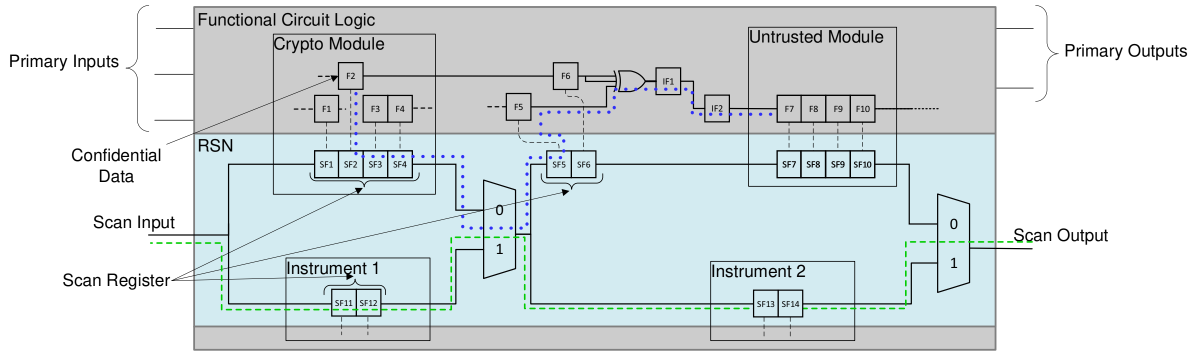


Fig. 1: Running example: circuit (gray background) with reconfigurable scan network (blue background), where flip-flops, modules and a fraction of the circuit logic is illustrated. Shifting confidential data (inside the flip-flop F2) from the crypto module into the untrusted module must be prohibited. A *pure* scan path (dashed green line) and a *hybrid* scan path (dotted blue line) are illustrated.

to SF14) that are combined into 5 scan registers. The illustrated part of the functional circuit logic encompasses 10 flip-flops which are directly connected to the RSN (F1 to F10) and two flip-flops which are not directly connected to the RSN, so-called *internal flip-flops* (IF1 and IF2).

In an RSN different active scan paths can be configured. Figure 1 pictures the active scan path if both scan multiplexers are set to 1 (green dashed line).

RSNs are equipped with three global signals *capture*, *shift* and *update*. These signals controlled by the TAP controller enable either the *capture-*, *shift-* or *update-phase*, respectively:

- Capture-phase: Data from the underlying circuit is transferred into scan registers.
- Shift-phase: Data is shifted along the scan infrastructure, i.e. from the scan-in port toward the scan-out port.
- Update-phase: Data inside the scan registers is transferred into optional so-called shadow registers or into the underlying circuit.

For clarity, these signals are omitted in Figures 1, 4 and 5.

B. Security Specification

We describe access permissions and restrictions to instruments connected to scan segments in an RSN using the security specification from [17]. For the *security specification* each scan segment is annotated with a trust category, formalizing the trustworthiness of the scan segment (or its surrounding core), and a set of accepted trust categories, characterizing the respective data sensitivity.

The security specification is violated if an active scan path traverses over two segments, where the data stored in one segment is too confidential to be on the same scan path with the other segment. We consider an RSN as *(data flow) secure*, if there is no such violation. A detailed description of the security specification can be found in [17].

Especially for complex RSNs, it is in general not straightforward to efficiently detect such security violations as an in-depth analysis of the RSN structure and its security specification is required.

C. Hybrid Scan Paths

Figure 1 shows a scan path starting from the scan-in port, over scan elements and ending at the scan-out port (green dashed line). It can be seen that this way data is shifted solely through the scan infrastructure and not through the underlying circuit – we call such a path a *pure scan path*. There also might be paths, that use the underlying circuit in addition to the scan infrastructure to transfer information. We call such a scan path, that indirectly connects scan elements over the underlying circuit a *hybrid scan path*.

In the running example (Figure 1) there are two scan paths on which the confidential data from the crypto module can be shifted into the untrusted module:

- Pure (not illustrated): The confidential content of F2 is captured into SF2, then shifted *purely* over the scan infrastructure first into SF3, SF4, SF5 and SF6 and then into SF7. Lastly it is updated into F7.
- Hybrid (blue dotted line): Similar to the pure scan path, at first the confidential content of F2 is captured into SF2 and then shifted into SF5; then, the data is updated into F5. An attacker might now use the circuit's functionality to transfer the data over IF1 and IF2 into F7.

These paths can be utilized by an attacker to retrieve confidential information, e.g. via a side-channel attack on the untrusted module. Therefore both paths pose a security threat and must be prohibited.

D. Attack Scenario

In the following we will present a security threat that can be detected and resolved with the proposed method. It should be noted that [17] also elaborated on a similar attack scenario, but only paths *purely* over the scan infrastructure were investigated. In this work pure and additionally *hybrid paths*, i.e. paths over both the scan infrastructure and the underlying circuit, are investigated.

Scan infrastructure connects instruments deeply inside the circuit with each other. Those instruments might be from different vendors and sources, where one is untrusted potentially due to lower security standards: A sensor might be very vulnerable to side-channel attacks, as it hardly contains secret information.

Thus, shifting confidential data over a scan path involving such an untrusted module must be prohibited to avoid an insecure operation of the chip.

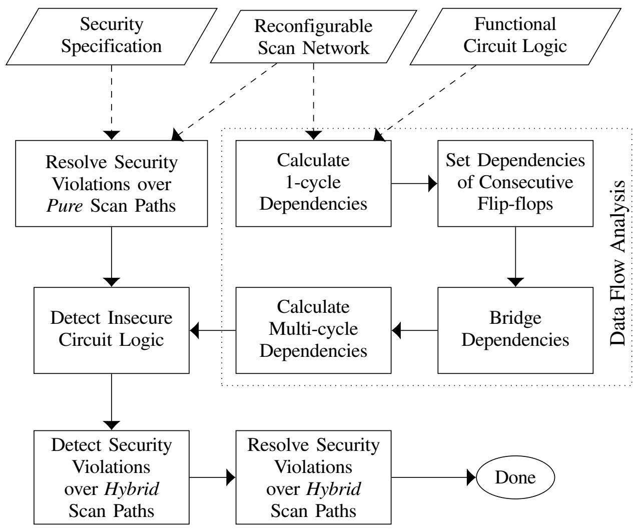
In summary, the following conditions describe the above described security threat:

- One scan segment contains *confidential data*.
- There is a pure or hybrid scan path through this segment and an *untrusted* module.

III. Proposed Method

An overview of the proposed method is outlined in Figure 2. First, the RSN is passed to the method of [17], where it is annotated with the user-given security specification and all security violations over pure scan paths are detected and resolved.

A *data flow analysis* (over multiple cycles) is performed to capture the circuit's capability to transfer data from one instrument to another. Based on these information, the data flow is checked against the user-given security specification. In case any security



Security Specification
Reconfigurable Scan Network
Functional Circuit Logic
Resolve Security Violations over *Pure* Scan Paths
Calculate 1-cycle Dependencies
Set Dependencies of Consecutive Flip-flops
Data Flow Analysis
Detect Insecure Circuit Logic
Calculate Multi-cycle Dependencies
Bridge Dependencies
Detect Security Violations over *Hybrid* Scan Paths
Resolve Security Violations over *Hybrid* Scan Paths
Done

Fig. 2: Overview of the proposed method.

violations are identified, the structure of the RSN is changed such that the violations are resolved.

For the data flow analysis (cf. Chapter III-A) over the circuit logic, first a SAT-based method [18] computes the circuit-internal dependencies over one cycle. Then two subroutines which are essential for the method's feasibility are executed: setting the dependencies of consecutive flip-flops inside a scan register and bridging internal dependencies, which greatly reduces the amount of calculations and stored data.

As a security analysis must investigate on the dependency of distant segments, we compute multi-cycle dependencies iteratively [18] on the reduced data set.

The calculated multi-cycle dependencies are then used to detect insecure circuit logic, i.e. security violations independently of the scan infrastructure (cf. Chapter III-B). If the circuit logic was not found to be insecure, the security analysis on hybrid scan paths is performed, where security violating data paths through both the RSN and the circuit logic are detected (cf. Chapter III-C) and then removed (cf. Chapter III-D).

A. Data Flow Analysis

We use the notation of structural and functional dependency as in [18]: If there is a connection between two flip-flops, on which data can be propagated in one cycle, we say that one flip-flop *1-cycle functionally depends* on the other. In contrast, if there is a connection between two flip-flops, but data cannot be propagated from one flip-flop to the other, the dependency of one flip-flop on the other is *only structural*. If there is a path between two flip-flops, where every two consecutive elements *1-cycle functionally depend* on each other, the flip-flop at the end of the path is called *path-dependent* on the flip-flop at the beginning of the path. Naturally, 1-cycle functionally dependent flip-flops are also path-dependent (over a path of length 1).

In the example of Figure 1, IF2 is 1-cycle functionally dependent on IF1, IF1 is 1-cycle functionally dependent on F5 and IF1 is 1-cycle only structurally dependent on F6 due to the reconvergence. It is also seen that IF2 is path-dependent on F5 and IF2 is multi-cycle only structural dependent on F6.

After the calculation of the 1-cycle dependencies, we execute two subroutines, which greatly reduce the computational effort for the calculation of multi-cycle dependencies:

1) (Pre-)Setting dependencies of consecutive flip-flops: For each pair of flip-flops inside a scan register the latter flip-flop is set to be path-dependent on the former due to the data-propagation functionality of the RSN.

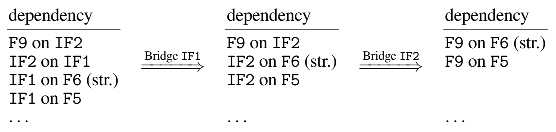


Fig. 3: Two iterative steps to bridge dependencies over internal flip-flops of the running example (Figure 1). *Only structural* dependencies are marked with *(str.)*.

The number of hereby preset dependencies is quadratic in the number of scan flip-flops per scan register. Thus, a substantial amount of dependencies does not have to be elaborately calculated. Additionally, the proposed presetting allows for a faster multi-cycle dependency calculation of dependencies over large scan registers.

2) Bridging dependencies: In this work we propose to *bridge* the dependencies over flip-flops, that are not directly connected to the scan infrastructure; so-called *internal flip-flops* (e.g. flip-flops IF1 and IF2 in Figure 1). We thus avoid denoting the pair-wise dependency of every (internal) flip-flop pair, so that they can be excluded from the subsequent calculation of multi-cycle dependencies. On the average (for the experiments provided in the following section) we reduce the number of denoted flip-flops by 41.72% and the number of denoted dependencies by 65.37%.

The method iteratively bridges over every internal flip-flop. For example, to bridge the dependencies over the internal flip-flop IF1, the method considers the combination of:

- each flip-flop, on which IF1 was denoted as structurally or path-dependent:
  F5 (path-dependent) and F6 (only structural)
- each flip-flop, which depends on IF1:
  IF2 (path-dependent)

The multi-cycle dependency of the above listed flip-flops is updated as follows (cf. the first step in Figure 3):

- If one flip-flop is path-dependent on IF1 and IF1 is path-dependent on the other flip-flop, the dependency between those flip-flops is set to path-dependent.
  Thus, IF2 is path-dependent on F5.
- If a flip-flop is denoted as only structural dependent on IF1 or IF1 is denoted as only structural dependent on a flip-flop, there is a structural dependency between those flip-flops. Therefore, if they are not already known to be path-dependent, the dependency is set to *only structural dependency*.
  Thus, IF2 is only structural dependent on F6.

It should be noted, that the dependencies of the also internal flip-flop IF2 are only updated, if IF2 has not been bridged over yet.

At last, the iterative method of [18] is applied to calculate the multi-cycle dependencies in the circuit logic. The runtime of this technique is cubic in the number of flip-flops. Thus, the due to bridging reduced amount of analyzed flip-flops, greatly reduces the number of executed calculations. Bridging is therefore necessary for the feasibility of the proposed method.

It should be noted that at this point of the method, the dependencies are calculated omitting the RSN. This is due to the fact, that to later resolve security violations, the RSN connections get changed multiple times. Thus, the dependencies change and would have to be re-calculated after every change to the RSN. Instead, the dependencies are calculated once at the beginning of the method without RSN-internal connections and therefore can be reused for the entire security analysis.

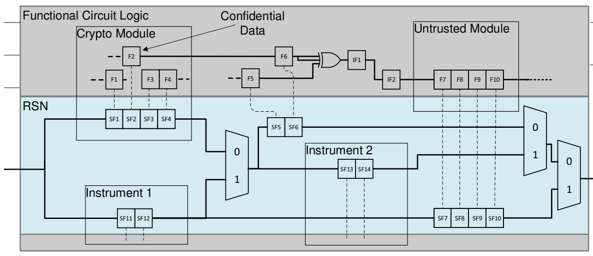


Fig. 4: Running example (cf. Figure 1) after the violating *pure* scan path has been resolved.

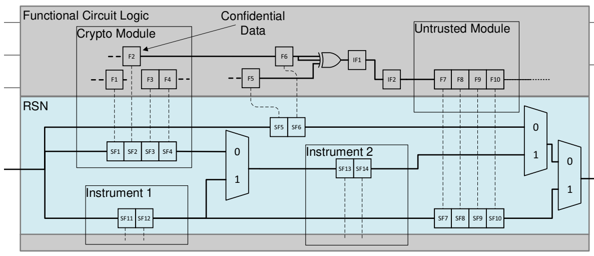


Fig. 5: Secure version of the running example (cf. Figure 1) after the application of the proposed method. The path over F6 crosses the reconvergence at the XOR-gate and thus cannot propagate (confidential) data to the untrusted module.

B. Insecure Circuit Logic

If the circuit logic was designed without examining security aspects, a security violation might occur over a path solely inside the circuit logic, not involving the scan infrastructure.

Such a security violation can only be fixed by analyzing and altering the circuit functionality, with respect to the security requirements. In this work, we investigate security violations, that occur due to the existence of scan infrastructure and can hence be resolved by modifying the scan infrastructure, especially without altering the circuit logic.

Therefore, for each pair of path-dependent flip-flops, we test if this circuit-internal dependency causes a security violation. The violation would occur even without the scan infrastructure and thus had to be resolved with a new design of the circuit.

C. Detecting Security Violations over Hybrid Scan Paths

In [17] a method to efficiently detect security violations on *pure* scan paths was shown. This method can be summarized in the following steps:

Propagation of Security Attributes: A forward traversal through the circuit was executed, propagating security attributes from the scan-in ports, over the scan registers toward the scan-out ports.

Finding a Security Violation: The propagated security attributes were used to efficiently test for security violations.

Resolving the Security Violation: If a security violation was found, multiple candidates to resolve that violation were generated and evaluated. The candidate with the lowest cost was applied.

This process was repeated until all security violations were resolved.

In this work, we at first apply the method from [17]. Afterwards, all security violations over *pure* scan paths are resolved, but there might be still security violations over *hybrid* scan paths, which need to be addressed. Figure 4 shows the running example of Figure 1, after the method was applied. It is seen that SF6 and SF7 have been disconnected and newly connected to other scan elements. In Chapter II-C, both a pure and a hybrid scan path were shown that could allow an attacker to shift confidential data into the untrusted module. As the pure scan path has led over the now disconnected SF6 and SF7, the method resolved the security violation, which occurred over this pure scan path.

However, the second security violation over the hybrid path of Chapter II-C remains unresolved. This violation is targeted by the novel technique presented in this work. While in the method of [17] all flip-flops of one scan register could be handled simultaneously, this is no longer possible if the underlying circuit is considered: Scan flip-flops of the same scan register might be linked to different parts of the underlying circuit and may therefore have non-trivial dependency differences. For example in Figure 4 there is a connection from F2 to F6, then into the scan register encompassing SF5 and SF6. From this scan register, there is a second path, leading into F5, then IF1, IF2 and then into the untrusted module. A method on register level would falsely concatenate those paths and wrongly detect a security violation, as on register level it appears to be possible to propagate the confidential data of F2 over these two paths into the untrusted module. Contrary, if the method employs flip-flop granularity, it is seen, that the two paths cannot be concatenated into one insecure path, as there is no connection from SF6 to SF5 (only vice versa). Therefore a flip-flop-granular method correctly detects no security violation.

To employ flip–flop granularity, we store for each scan register, both the propagated security attributes of the first scan flip-flop and the first flip-flop where the propagated security attributes change.

D. Resolving Security Violations

After the successful detection of security violations, the RSN must be secured. We therefore separate insecure paths and reconnect the disconnected segments to other scan segments, with which they do not cause a security violation.

Due to the hybrid scan paths the proposed algorithm must deal with the following challenges:

Maintaining a Cycle-free Scan Infrastructure: To resolve a security violation, a connection between scan segments is cut. Scan segments are prevented from dangling, by connecting every separated segment S directly to other segments, which were already connected to S over multiple cycles. To maintain a cycle-free scan infrastructure, the reconnection must ensure to not create cycles. Therefore only segments, that are multi-cycle predecessors/successors over *pure* scan paths are connected to such a segment S. If no such predecessor (successor) exists, S will be connected to the scan-in port (scan-out) port.

Omnidirectionally Propagated Security Attributes: If only *pure* scan paths are investigated, security attributes are propagated from the scan-in port over every scan segment exactly once toward the scan-out port. If also hybrid scan paths are invoked, scan segments might be connected differently, e.g. while one scan segment $\mathtt{S_x}$ might be a predecessor of $\mathtt{S_y}$ with respect to the scan infrastructure, there might be a path in the underlying circuit, such that over this path $\mathtt{S_y}$ is a predecessor of $\mathtt{S_x}$. Therefore propagation of security attributes might be cyclic and is in general more elaborate on hybrid scan paths than on pure scan paths. While on hybrid scan paths the propagation in general touches some segments multiple times, reaching a fixed-point is still guaranteed, as only a (small) finite number of security attributes are propagated.

Rootless Cyclic Propagation of Security Attributes: After scan segments are separated, the propagated security attributes need to be updated. With the above mentioned propagation of security attributes, for every scan register information on the (transitive) predecessors is stored. The underlying details on the propagation of security attributes are based on [17]. However, the following should be mentioned: While the root-cause for a propagated

	Benchmark	#Scan Registers	#Scan Flip-Flops	#Scan Mux's	#Reg. with security violation	#Applied Changes			Runtime (in s)			
										Violation Detect. & Correct.		
						pure	hybrid	total	Dependency Calculation	pure	hybrid	Total Runtime
Bastion	BasicSCB	21	176	10	1.56	1.4	0.6	2.0	0.13	0.00	0.00	0.13
	Mingle	22	270	13	2.21	1.8	0.8	2.5	0.18	0.00	0.00	0.19
	TreeFlat	24	101	24	3.65	3.0	1.7	4.7	0.05	0.01	0.01	0.06
	TreeFlatEx	122	5 194	59	8.45	5.8	6.3	12.1	26.48	0.07	0.09	26.65
	TreeBalanced	90	5 581	46	7.22	4.7	4.3	9.0	43.12	0.04	0.05	43.21
	TreeUnbalanced	63	41 887	28	6.27	3.9	3.7	7.6	16 686.78	0.02	0.08	16 686.87
	q12710	50	26 185	27	5.20	3.8	3.3	7.1	5 703.16	0.02	0.04	5 703.22
	t512505	287	77 005	159	12.44	9.2	15.7	24.9	28 702.78	0.32	1.14	28 704.23
	p22810	524	30 098	270	21.75	17.2	24.6	41.9	1 082.98	1.02	1.91	1 085.91
	a586710	64	41 667	32	5.89	4.3	4.2	8.4	14 724.12	0.01	0.08	14 724.21
	p34392	197	23 196	96	11.26	8.2	13.3	21.4	1 072.99	0.07	0.21	1 073.27
	p93791	1 185	98 611	596	40.51	35.4	44.1	79.5	14 592.50	1.83	5.32	14 599.64
	FlexScan	8 485	8 485	4 243	207.22	203.7	247.7	451.4	32.73	827.54	1 012.72	1 872.99
Industrial	MBIST_1_5_5	113	548	15	6.64	2.3	10.8	13.2	0.21	0.01	0.03	0.25
	MBIST_1_5_20	338	1 523	15	9.00	3.3	36.2	39.5	1.13	0.04	0.38	1.55
	MBIST_1_20_20	1 313	6 068	45	7.60	2.4	38.2	40.6	13.90	0.15	1.25	15.29
	MBIST_2_5_5	224	1 091	28	6.18	3.6	8.1	11.7	0.46	0.04	0.08	0.58
	MBIST_2_5_20	674	3 041	28	8.88	4.7	38.9	43.6	3.28	0.17	1.05	4.50
	MBIST_2_20_20	2 624	12 131	88	2.45	1.6	1.0	2.6	67.86	0.44	0.52	68.82
	MBIST_5_5_5	557	2 720	67	9.64	6.6	15.1	21.7	1.51	0.15	0.35	2.02
	MBIST_5_20_20	6 557	30 320	217	4.56	2.8	10.1	12.9	465.85	2.70	6.40	474.95
	MBIST_20_20_20	26 222	121 265	862	19.62	15.1	89.8	104.8	9 359.48	0.87	73.19	9 433.54

TABLE I: Experimental results for the proposed method (cf. Figure 2)

security attribute might now be separated from following segments, the following segments potentially form a cycle. The segments must then be updated, such that the security attribute is not propagated inside this cycle anymore, which can be done by a root-cause analysis or a recalculation of all security attributes. To avoid a runtime-consuming root-cause analysis, in this work we propagate the security attributes anew for hybrid scan paths.

After all security violations over hybrid scan paths are resolved, the method returns a (data flow) secure RSN. The secure design of the running example is illustrated in Figure 5. To resolve the violation over the hybrid scan path, the first scan multiplexer was disconnected from `SF5`; also `SF5` was connected to the scan-in port. No functional path leads from the crypto module to the untrusted module; only a structural path exists, but leads over the reconvergence at `F6`, and therefore cannot propagate (confidential) data. Thus, the RSN is (data flow) secure.

IV. Experimental Evaluation

A. Experimental Setup

The proposed method has been evaluated using a subset of the benchmarks introduced by *BASTION* in [19] which have 21 to 8,485 scan registers and 101 to 98,611 scan flip-flops. The benchmark subset encompasses all 12 acyclic benchmarks with explicitly stated modules or instruments for which an ICL source file exists as well as the benchmark with the largest number of scan registers (*FlexScan*). To integrate *FlexScan* it was assumed, that each scan register belongs to a different module. Additionally, we evaluated our method on a set of 9 industrial style benchmarks. These benchmarks model a scalable memory built-in self-test (MBIST) network and have 113 to 26,222 scan registers and 548 to 121,265 scan flip-flops. The network MBIST_n_m_o consists of a chip with n cores, each of which contains m MBIST controllers. Each controller is responsible for o individual memories. The network is built up hierarchically to allow for fast access times to each MBIST controller and to facilitate the parallel operation of the controllers. On the chip level each core can be included or excluded from the scan path. Similarly, each MBIST controller can also be included or excluded from the scan path through the core.

Each of the total 22 benchmarks is only available without the underlying circuit. We therefore randomly generated 10 circuits per benchmark and evaluated the method on those 10 circuits. Table I presents the average results over all generated circuits.

The algorithm introduced in the previous chapters has been implemented on top of the tool *eda1687* [20], and uses the techniques from [17] and [18] as presented in Chapter III. All experiments are conducted on a single core of an Intel Xeon CPU running at 3.3 GHz with 64 GB of main memory.

The security specification for real world applications is specified based on the sensitivity and confidentiality of the underlying instruments. In order to allow a thorough evaluation of the characteristics of the presented method on a wide set of benchmarks, we randomly generated the security specifications with 16 different security requirements for each benchmark and list the averaged results over all security specifications, where a security violation occurred, but the circuit logic itself is not insecure (cf. Chapter III-B).

B. Experimental Results

The results for all evaluated benchmark circuits are presented in Table I. The first four columns enlist structural information: the name of the respective benchmark, the total number of scan registers, the total number of scan flip-flops, and the total number of scan multiplexers. The name encoding of the industrial benchmarks is described in Chapter IV-A. It is seen, that the benchmarks vary greatly in size and complexity, starting from the smallest benchmark *BasicSCB* with only 176 scan flip-flops and 10 scan multiplexers to one of the largest benchmarks *MBIST_20_20_20*, encompassing over 121,000 scan flip-flops and over 850 scan multiplexers. The respective number of scan multiplexers in general approximates the complexity of a benchmark circuit, with the exception of *FlexScan*, where all 4,243 scan multiplexers are connected in serial. The next column lists the number of scan registers, where at least one flip-flop causes a security violation with a predecessor before the RSN is passed to the method. After application of the method all security violations are resolved.

As described in the previous section, the proposed method resolves the security violations by changing the scan infrastructure (cf. Figure 1 and Figure 4). It first resolves all security violations over pure scan paths and then all security violations over hybrid scan paths (cf. Figure 2). Columns 6, 7 and 8 list the number of applied changes to resolve all security violations for pure and hybrid scan paths and the total number, respectively.

The importance of looking into hybrid scan paths can clearly be seen. By resolving only security violations over pure scan paths (as was done in [17]), the benchmark circuits are in general still vulnerable to the in Chapter II-D presented attack scenario. Even more so, the number of changes to resolve security violations over pure scan paths is on average less than half (43%) the number of total changes that are needed to secure the RSN. Thus, it can be seen that by resolving the violations over pure scan paths, in general less than half the effort of securing the RSN is processed. Only with the combined method presented in this work, it can be guaranteed that the network is free from such security violations and that secret data cannot be shifted through an untrusted instrument.

The last 4 columns enlist the runtime of the proposed method and three subroutines in seconds, starting with the runtime for the calculation of all dependencies, i.e. for the data flow analysis described in Chapter III-A. The next two columns show the runtime needed for the detection and correction of security violations; for most benchmarks these two steps are fully executed in under 10 seconds, only for the two benchmarks with the highest amount of scan registers (*FlexScan* and *MBIST_20_20_20*), the combined runtime is larger.

The total runtime encompasses the runtime of the three previous columns plus the runtime of the detection of insecure circuit logic (cf. Chapter III-B).

Small benchmark circuits run in under one minute, while the largest total runtime for one benchmark circuit is 28,704s (around 8h) for *t512505*. In real world applications, it is sufficient to apply the proposed algorithm once per circuit design. Table I shows that even for large benchmark circuits the proposed method computes "overnight", whether or not the circuit is vulnerable to the proposed security threats and how the vulnerability can be prevented with an alternative design of the reconfigurable scan network. The alternative design includes all scan registers of the original insecure reconfigurable scan network.

C. Approximating Path-Dependency with Structural Dependency

Table I (Column 'Dependency Calculation') shows that the proposed approach of computing multi-cycle dependencies highly contributes to the overall runtime. The runtime for dependency computation can be greatly reduced, if instead of path-dependency, only structural dependency is computed, as the algorithm then only has to structurally traverse through the circuit, not considering reconvergencies and other data flow canceling effects.

With this *over*-approximation, every security violation is still detected, but false positives might occur, i.e. an only structural data path is falsely computed as insecure. The in general higher amount of "detected" security violations leads to a higher demand of changes to the scan infrastructure.

Application of the proposed technique, where path-dependency was over-approximated by structural dependency, resulted in on average 61% additional changes to the scan infrastructure, e.g. re-routing wires and placing new multiplexers, which should for most security-relevant real-world applications be by far more expensive than 8 hours of one-time run-time.

Additionally, for 6.21% of the investigated benchmarks, due to the over-approximation with structural dependency, the circuit logic of the benchmark was falsely classified as insecure (cf. Chapter III-B). This leads to an entirely different problem class, which is by far harder to solve and requires expensive solutions, as the logic of the underlying circuit (unnecessarily) would have to be changed.

V. Conclusion

While scan infrastructure is highly beneficial for circuit testing, if not thoughtfully implemented, it can be exploited by an attacker. We presented an attack scenario, where an attacker uses the data flow of the scan infrastructure and the circuit logic to e.g. shift confidential data into a module with low security standards and then read it out via a side-channel attack. In this work a method was presented to efficiently

- calculate all multi-cycle dependencies of scan flip-flops over the circuit logic,
- detect insecure circuit logic and
- detect and resolve security violations not only over pure but also over hybrid scan paths, i.e. scan paths leading over both scan infrastructure and circuit logic.

The method's feasibility is demonstrated on a wide set of academical and industrial benchmarks. After application of the proposed method, an attack via the presented attack scenario is fully prevented.

Acknowledgments

This work was financed by Baden-Württemberg Stiftung (IKT-Sicherheit, SHIVA).

References

[1] J. Rearick and A. Volz, "A Case Study of Using IEEE P1687 (IJTAG) for High-Speed Serial I/O Characterization and Testing," in *Proc. IEEE Int'l Test Conference (ITC)*, 2006, paper 10.2.

[2] J. D. Rolt, G. D. Natale, M.-L. Flottes, and B. Rouzeyre, "A Novel Differential Scan Attack on Advanced DFT Structures," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 18, no. 4, pp. 58:1–58:22, Oct. 2013.

[3] D. Hely, M. L. Flottes, F. Bancel, B. Rouzeyre, N. Berard, and M. Renovell, "Scan Design and Secure Chip [Secure IC Testing]," in *Proc. IEEE On-Line Testing Symposium (IOLTS)*, 2004, pp. 219–224.

[4] M. Tehranipoor and C. Wang, Eds., *Introduction to Hardware Security and Trust*. Springer, 2012.

[5] J. Da Rolt, A. Das, G. Di Natale, M.-L. Flottes, B. Rouzeyre, and I. Verbauwhede, "Test Versus Security: Past and Present," *IEEE Trans. on Emerging Topics in Computing*, vol. 2, no. 1, pp. 50–62, March 2014.

[6] J. Dworak, A. Crouch, J. Potter, A. Zygmontowicz, and M. Thornton, "Don't Forget to Lock your SIB: Hiding Instruments using P1687," in *Proc. IEEE International Test Conference (ITC)*, 2013, paper 6.2.

[7] A. Zygmontowicz, J. Dworak, A. Crouch, and J. Potter, "Making It Harder to Unlock an LSIB: Honeytraps and Misdirection in a P1687 Network," in *Proc. Conference on Design, Automation & Test in Europe (DATE)*. EDAA, 2014.

[8] R. Baranowski, M. A. Kochte, and H.-J. Wunderlich, "Fine-Grained Access Management in Reconfigurable Scan Networks," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 34, no. 6, 2015.

[9] K. Rosenfeld and R. Karri, "Security-Aware SoC Test Access Mechanisms," in *Proc. IEEE VLSI Test Symposium (VTS)*, 2011, pp. 100–104.

[10] J. Lee, M. Tehranipoor, C. Patel, and J. Plusquellic, "Securing Designs against Scan-Based Side-Channel Attacks," *IEEE Trans. on Dependable and Secure Computing*, vol. 4, no. 4, pp. 325–336, Oct.-Dec. 2007.

[11] R. Nara, H. Atobe, Y. Shi, N. Togawa, M. Yanagisawa, and T. Ohtsuki, "State-Dependent Changeable Scan Architecture against Scan-Based Side Channel Attacks," in *IEEE International Symposium on Circuits and Systems (ISCAS)*, 2010, pp. 1867–1870.

[12] J. DaRolt, G. D. Natale, M.-L. Flottes, and B. Rouzeyre, "On-chip test comparison for protecting confidential data in secure ICs," in *IEEE European Test Symposium (ETS)*, 2012, pp. 1–1.

[13] R. Baranowski, M. A. Kochte, and H.-J. Wunderlich, "Securing Access to Reconfigurable Scan Networks," in *Proc. IEEE Asian Test Symposium*, 2013.

[14] A. Atteya, M. A. Kochte, M. Sauer, P. Raiola, B. Becker, and H.-J. Wunderlich, "Online prevention of security violations in reconfigurable scan networks," in *Proc. IEEE European Test Symposium (ETS)*, 2018.

[15] E. Ebrard, B. Allard, P. Candelier, and P. Waltz, "Review of Fuse and Antifuse Solutions for Advanced Standard CMOS Technologies," *Microelectronics Journal*, vol. 40, no. 12, pp. 1755–1765, 2009.

[16] M. A. Kochte, M. Sauer, L. Rodríguez Gómez, P. Raiola, B. Becker, and H.-J. Wunderlich, "Specification and Verification of Security in Reconfigurable Scan Networks," in *IEEE European Test Symposium (ETS)*, 2017.

[17] P. Raiola, M. A. Kochte, A. Atteya, L. Rodríguez Gómez, H.-J. Wunderlich, B. Becker, and M. Sauer, "Detecting and Resolving Security Violations in Reconfigurable Scan Networks," in *Proc. IEEE International Symposium on On-Line Testing and Robust System Design (IOLTS)*, 2018.

[18] M. Soeken, P. Raiola, B. Sterin, B. Becker, G. De Micheli, and M. Sauer, *Proc. 12th International Haifa Verification Conference (HVC)*. Springer, 2016, ch. SAT-Based Combinational and Sequential Dependency Computation.

[19] A. Tšertov, A. Jutman, S. Devadze, M. S. Reorda, E. Larsson, F. G. Zadegan, R. Cantoro, M. Montazeri, and R. Krenz-Baath, "A suite of IEEE 1687 benchmark networks," in *IEEE International Test Conference (ITC)*, 2016.

[20] R. Baranowski, M. A. Kochte, and H.-J. Wunderlich, "Reconfigurable Scan Networks: Modeling, Verification, and Optimal Pattern Generation," *ACM Trans. on Design Automation of Electronic Systems*, vol. 20, no. 2, 2015.