FastPath: A Hybrid Approach for Efficient Hardware Security Verification

Lucas Deutschmann* — *RPTU Kaiserslautern-Landau* — Kaiserslautern, Germany — lucas.deutschmann@rptu.de

Andres Meza* — *UC San Diego* — San Diego, USA — anmeza@ucsd.edu

Dominik Stoffel — *RPTU Kaiserslautern-Landau* — Kaiserslautern, Germany — dominik.stoffel@rptu.de

Wolfgang Kunz — *RPTU Kaiserslautern-Landau* — Kaiserslautern, Germany — wolfgang.kunz@rptu.de

Ryan Kastner — *UC San Diego* — San Diego, USA — kastner@ucsd.edu

***Abstract*—Many verification methods have been proposed to detect microarchitectural information leakage in response to the surge of security breaches in hardware designs. These sophisticated efforts have gone a long way toward preventing attackers from breaking the system's confidentiality. However, each approach has its own set of weaknesses: it may not be scalable enough, exhaustive enough, flexible enough to meet changing requirements or fit well into existing verification flows.**

We propose FastPath, a hybrid verification methodology that combines the efficiency of simulation with the exhaustive nature of formal verification. FastPath employs a structural analysis framework to automate the method further. Our experimental results compare FastPath to a state-of-the-art formal approach, showing a significant reduction in manual effort while achieving the same level of exhaustive confidence. We also discovered and contributed a fix for a previously unknown leak of internal operands in cv32e40s, a RISC-V processor intended for security applications.

***Index Terms*—Hardware Security, Information Flow Tracking, Simulation, Formal Verification, Data-Oblivious Computing.**

I. Introduction

A seemingly endless flood of timing side channel attacks [1], [2], [3], [4], [5] severely disrupted the view of hardware as a root of trust. This necessitates rethinking architectures and fortifying them to meet ever-changing security requirements. In response, software and hardware communities are actively proposing novel mitigation techniques [6], [7], [8], [9], [10], [11] to address these requirements.

Most countermeasures against microarchitectural timing side channels demand that specific basic hardware operations are *data-oblivious*. Data-obliviousness requires these hardware primitives to process data without any measurable data-dependent side effects, e.g., the operation's timing must be independent of its inputs. A prominent software paradigm, called *Constant-Time Programming* [12], [8], [13], assumes that simple instructions, such as an addition, do not leak data via timing side channels. More complex instructions, such as a division or floating point operations, are assumed to be data-dependent and replaced with data-oblivious primitives [14]. Making matters worse, whether an instruction is data-oblivious or not depends on the underlying microarchitecture. Intel [15], ARM [16] and RISC-V [17] have therefore added ways to refine the instruction set architecture (ISA) with information about the data-obliviousness of individual instructions.

However, recent insights [18], [5] have shown that, despite these precautions, sophisticated optimizations in modern processors can invalidate these assumptions. Chen et al. [5] exploit an optimization feature of recent Apple devices, called data memory-dependent prefetcher, to attack constant-time cryptographic routines by tricking the prefetcher into misinterpreting secret data as addresses. This attack demonstrates that it is insufficient to simply assume a certain behavior of the hardware layer. Instead, it is necessary to provide *exhaustive* security guarantees upon which software countermeasures can be built.

As part of the effort to restore trust in hardware, we propose FastPath, a hybrid security verification approach. The method exhaustively verifies *non-interference* [19], [20] properties, e.g., the property that confidential information does not leak into publicly visible states of the system. In this paper, we specifically verify that hardware operates in a data-oblivious manner. However, FastPath is not limited to this threat model.

The key observation behind FastPath is that recent formal methods [21], [22] achieve high scalability through a *semantic partitioning* of the computational problem. Rather than partitioning the design *structurally* into paths to be checked formally, these methods logically decompose the global proof problem. While the new techniques scale even to complex out-of-order processors, deriving this logical partitioning can require a considerable amount of manual refinement.

With FastPath, we identify and leverage a unique synergy between these formal methods and simulation-based security verification: simulating a design with Information Flow Tracking (IFT) [23] efficiently derives the bulk of the problem partitioning required by the formal methods, eliminating most manual effort. Formal verification, on the other hand, complements the weakness of simulation in finding subtle leaks in corner-case scenarios. To round out our hybrid approach, FastPath employs static structural analysis to detect when simulation and formal are not necessary and to generate property templates when they are needed. This novel way of combining these methods creates a highly automated and scalable, yet exhaustive verification approach.

In summary, this paper makes the following contributions:

- We propose FastPath, a hybrid verification approach for hardware security that leverages the complementary nature of simulation and formal methods (Sec. IV). Instead of combining the two approaches in a straightforward way by applying a formal analysis to certain paths not covered during simulation, our method uses a global property formulation and a seamless communication of intermediate results between the different methods.
- We integrate a structural analysis and IFT framework called HyperFlow Graph (HFG) [24] (Sec. III-A) into FastPath. The structural information can be used to exclude trivial cases where no syntactic information flow paths exist and to generate templates for the formal properties.
- We apply FastPath to a variety of accelerator and processor designs (Sec. V) and compare the manual effort with a formal-only approach [22]. Our case studies show a 36% to 100% reduction in manual refinement. In an experiment with the cv32e40s [25] processor, FastPath was able to detect a previously unknown leak of internal operands.

*Both authors contributed equally to this research.

This work has been supported in part by funding from the Agentur für Innovation in der Cybersicherheit GmbH (Cyberagentur), in part by BMBF ZuSE (Scale4Edge), 16ME0122K-16ME0140+16ME0465, and in part by Intel Corp. (Scalable Assurance).

II. Threat Model

FastPath verifies *non-interference* properties [19], [20]. *Non-interference* divides the system into *high*-security and *low*-security domains and can be used to model a variety of different threat models. Confidentiality, for example, requires that information never leak from *high* to *low*. For the purposes of this paper, we adopt the threat model and definitions from [22]. In particular, we consider a hardware system that processes confidential input data. The attacker exploits violations of data-obliviousness in the hardware and observes any data-dependent side-effect, i.e., any deviation in the control behavior caused by the values of the operands being processed. This is usually reflected in a difference in timing, resulting in different values at the control related output of the system. Note that our method is not limited to this threat model and can be extended to other attack scenarios.

Our threat model is described more precisely as follows: We model a hardware system as a standard Mealy-type Finite-State Machine (FSM) $M = (I, O, S, S_0, \delta, \lambda)$, where we denote sets of input symbols I, output symbols O, states S, and initial states $S_0 \subseteq S$. The state transition function is given by $\delta : S \times I \mapsto S$ and the output function is given by $\lambda : S \times I \mapsto O$. In addition, we denote the set of Register-Transfer Level (RTL) input signals by X, output signals by Y, and state signals by Z. We partition the set of input signals X (output signals Y) into disjoint sets of control inputs X_C (control outputs Y_C) and data inputs X_D (data outputs Y_D). In this threat model, as described in previous works [26], [22], [27], [28], the system processes the *high*-security data inputs X_D, while the attacker is able to observe the *low*-security control outputs Y_C. If a data-dependent control behavior is observed, such as a different timing of a handshake signal (e.g., `valid`), an attacker can infer information about the data being processed. This breaks confidentiality and can, for example, severely weaken encryption techniques. This security threat is eliminated if the design is proven to be *data-oblivious*.

Definition 1 (*Data-Obliviousness*)**.** A hardware module is called *data-oblivious* if, under a set of system-level constraints, the sequence of values at its control outputs Y_C is uniquely determined by the sequence of values at its control inputs X_C. In other words, its control inputs X_D have no influence on its control outputs Y_C.

Data-obliviousness of hardware designs is the verification target pursued throughout this paper.

III. Background

A. HyperFlow Graph

A HyperFlow Graph (HFG) [24], [29] is a graph-based intermediate representation which models how information flows in a hardware design. Derived automatically from a static analysis of RTL code, an HFG $G(N, E)$ contains a set of nodes N and a set of directed, labeled edges E. An HFG node $n_x \in N$ represents a unique hierarchical design signal `sig_x`. An HFG edge $e(ui, n_s, n_d, C) \in E$ represents a flow scenario in which information flows, explicitly or implicitly, from a source signal `sig_s` to a destination signal `sig_d`. Each edge can be precisely mapped to an originating RTL code construct. If the originating code construct has no guarding conditions ($C = \emptyset$), then the flow is always "active". Otherwise, the flow is only active if all guarding conditions $c \in C$ are simultaneously satisfied. Note that multiple edges are allowed between two nodes so each edge also has a unique identifier ui.

HFGs enable a variety of queries that produce security-relevant information. In this work, we primarily utilize the HFG path query $q(n_s, n_d)$ which returns a set of HFG paths P where each path $p \in P$ comprises a finite sequence of HFG edges $p = (e_1, \ldots, e_k)$ that could *potentially* enable information flow from the source signal `sig_s` to the destination signal `sig_d`. Emphasis is placed on the *potential* nature of the returned HFG paths due to the possibility for false positives. By design, the static analysis employed in the HFG construction process is intended to enable speed and scalability, but this comes at the price of producing an over-approximation of the information flow in the system. This means that the HFG path query can produce false positives but it will never produce false negatives. In other words, if there are no HFG paths ($P = \emptyset$) connecting the source signal `sig_s` to the destination signal `sig_d`, then it is guaranteed that the source signal `sig_s` cannot influence or leak information to the destination signal `sig_d`. However, if there are any HFG paths ($P \neq \emptyset$), then each path $p \in P$ has the potential to be an unrealizable sequence of design states (a false positive) and no guarantees on signal influence can be made without further analysis. As discussed in Sec. IV, our hybrid verification strategy employs simulation-based IFT and formal verification to address this issue.

B. Information Flow Tracking

Hardware Information Flow Tracking (IFT) [23] is a powerful verification strategy that allows designers to reason about information movement through a hardware design. Hardware IFT properties can be used to verify trace and hyperproperties related to confidentiality, integrity, and availability. These properties can specify where, when, and how information should or should not flow from source signals to destination signals, with relatively few operators and expressions. IFT properties can be written as assertions using the no-flow operator (`=/=>`) as follows: {`src_sigs`} `=/=>` {`dst_sigs`}. Hardware IFT tools determine whether the hardware adheres to the properties.

In simulation-based IFT, a given design is instrumented with additional logic (i.e., an IFT circuit) that tracks information flow by calculating and updating security labels (e.g., HIGH and LOW). An IFT-enhanced simulation produces a simulation trace with functional values and security labels for all signals. All design signals are initially labeled LOW, except for the source signals from which information is being tracked, which are labeled HIGH. Throughout the simulation, the IFT circuit updates signal label values considering the functional values from the simulation testbench and security labels. Any signal that starts with a LOW label and eventually reaches a HIGH label has received information from a source signal, i.e., the source signal is affecting it.

The ability to determine signal influence from merely tainting a source signal as HIGH is extremely valuable when dealing with large, complex designs. However, IFT-enhanced simulations are completely dependent on the underlying testbench to produce a property-relevant information flow. As a result, IFT properties that could be falsified may not be falsified due to an insufficient testbench. FastPath overcomes issues related to exhaustiveness via its formal verification step (Sec. IV).

C. Unique Program Execution Checking

Unique Program Execution Checking (UPEC) [21] is a formal verification methodology to exhaustively detect the propagation of confidential (or malicious) information at the RTL. The approach was originally created to detect transient execution side channel (TES) attacks, but has since been extended to cover a variety of different threat models. According to our threat model (see Sec. II), we focus on UPEC for data-independent timing (UPEC-DIT) [22].

UPEC is based on a technique called Interval Property Checking (IPC) [30] and uses a 2-safety computational model, i.e., the design under verification (DUV) is instantiated twice. At the start of the proof, each signal is initialized with the same value as its counterpart in the other instance. The only discrepancy between both instances is the information to be tracked, usually a proprietary secret that should not be revealed to the attacker. Any malicious information flow is detected by following the propagation of this difference through the system. In addition, IPC allows the equal initial state to be *symbolic*, and thus implicitly models every possible history of the system. Therefore, UPEC can "fast-forward" to the point where the propagation occurs, allowing scalability to complex systems.

UPEC-DIT is a variant of UPEC that can exhaustively verify the data-obliviousness of hardware systems according to Def. 1. In its computational model, the control inputs X_C are constrained to be equal between the two instances, while the data inputs X_D remain unconstrained. Also, both instances start from the same symbolic state. UPEC-DIT then verifies that for any arbitrarily long sequence of inputs, the control outputs Y_C of the two instances never diverge, certifying that the control behavior is independent of the processed data. The approach achieves scalability by an iterative partitioning of the internal state-holding signals based on counterexamples. Every signal declared as data is allowed to assume different values in the following iteration of the algorithm, thus implicitly modeling any previous propagation path. This process is repeated until a malicious propagation is found, or until a fixed point is reached where no new propagation occurs.

The exhaustive nature of UPEC allows to provide formal security guarantees, and thus achieve a much higher level of confidence compared to simulation-based approaches. However, the iterative manual inspection of counterexamples can be tedious and requires a certain level of design and verification expertise. In FastPath, we offload a majority of the manual work to fully-automated structural analysis and IFT-based simulation. We then leverage the intermediate simulation results to perform a UPEC-based proof, achieving the same level of exhaustiveness as the original approach combined with an improved scalability.

IV. Methodology

Fig. 1 gives an overview of FastPath. FastPath has three main steps: structural analysis based on the HFG [24], efficient bug hunting with IFT-enhanced simulation [23], and exhaustive formal verification based on UPEC-DIT [22]. We start with the RTL description of the DUV and a security specification that defines what data is considered confidential. This paper focuses on data-dependent side effects (as described in Sec. II), but the proposed methodology can also be extended to other threat models. We assume an attacker is able to measure the timing of a system by observing control-related outputs, e.g., by monitoring the signals of a bus.

Our goal is to exhaustively verify (with as little manual inspection as possible) that sensitive data inputs (X_D) do not affect the control behavior of the system (Y_C). In addition, by inspecting violations of data-obliviousness, FastPath systematically derives a set of software constraints under which the hardware is guaranteed to operate in a data-oblivious manner. The verified hardware, in combination with these derived restrictions for the software, forms a root of trust for the higher levels of the system stack.

A. Structural Analysis

The first step is to create an HFG of the design. Because this process considers only structural information, it is easily scalable to complex systems. The HFG serves two primary purposes. First, it is a starting point for the other two approaches, providing structural information that can be used to generate property templates or speed up proofs with optimizations such as cone-of-influence reduction. Second, we can leverage it to skip the proof procedure in trivial cases. There can be no information flow if there is no structural connection between sensitive data inputs X_D and publicly visible control outputs Y_C. This may be the case if the design is largely data-oriented and does not implement complex control behavior (cf. Sec. V). In particular, FastPath terminates early if

$$\forall n_x \in X_D, \forall n_y \in Y_C : q(n_x, n_y) = \emptyset$$

where $q(n_x, n_y)$ is an HFG path query (cf. Sec. III-A).

In many cases, however, structural connections exist, but the design still operates data-independently in the given application scenario. For example, data-dependent instructions are explicitly avoided in constant-time programming. Even though they are never used, the functionality of these instructions is still implemented, creating a structural path through which information could flow. Another example of such an application scenario is the limited functionality of an unprivileged user. Structural analysis alone is not sophisticated enough to account for such usage restrictions. Therefore, we need to be able to verify a system's data-obliviousness under a given set of software constraints.

B. IFT-Enhanced Simulation

In the next step, an IFT-enhanced simulation of the design is performed. This can be done using any existing testbench or random simulation approach. We seek to verify the following IFT property:

$$X_D \; =/=> \; Y_C$$

At the start of the simulation, all sensitive data inputs X_D are tainted (labels are set to HIGH), and the labels of the remaining signals are set to LOW. If any control output $y_c \in Y_C$ has its labels transition from LOW to HIGH, this constitutes a counterexample and indicates that it is influenced by the sensitive data.

In the case of a counterexample, the verification engineer must investigate the root cause. The counterexample may represent a scenario that contradicts the given application scenario. In this case, the setup must be adjusted, usually by introducing constraints on the simulation or restricting the flow policy, and the simulation is restarted. However, if an actual security problem is found, the leakage must be fixed in the RTL design, and the methodology starts over.

The main advantage of running an IFT-enhanced simulation is the ease of setup and the efficiency of debugging. Given a stimulus source and an interface partitioning (cf. Sec. II), no manual effort is required to run the simulation. In addition, each counterexample shows a complete propagation path from data input to control output, starting from reset. This avoids false alarms and makes root cause analysis easier than in a formal approach where the counterexamples may start from an arbitrary run-time state. However, if a signal does not become tainted during simulation, this does not mean that no information can flow into that signal. It only indicates that there is no information flow for the scenarios exercised by the testbench. Subtle corner-case scenarios can easily be missed, which often happens in the presence of unknown vulnerabilities. Therefore, FastPath uses formal verification to ensure that data cannot propagate further than what was found by simulation.

Definition 2 (*Untainted State Signals*)**.** For a given design with an input partitioning $X = X_D \cup X_C$ and a given testbench, we define

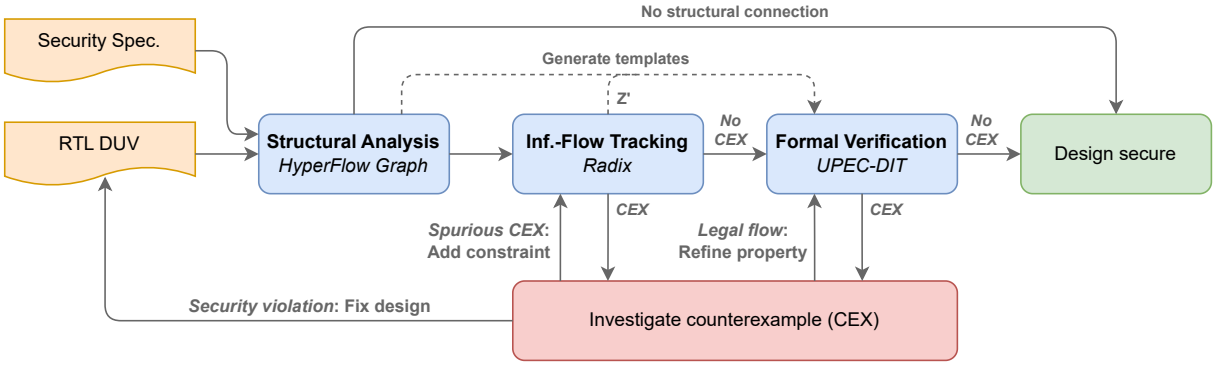

Fig. 1. The FastPath Verification Flow determines if the RTL design adheres to the security specification. FastPath uses a three-step process to efficiently compute counterexamples and update the specification with new constraints and property refinements. The output is a security violation that must be fixed in the design, or a guarantee that the design is secure.

the subset of all state-holding signals Z that have not been influenced by X_D during simulation as the set of untainted state signals $Z' \subseteq Z$.

Def. 2 describes the set of signals Z' whose labels remained LOW during simulation. The goal of the subsequent formal step is to verify that Z' constitutes a semantic partitioning of the computational problem. In particular, FastPath formally verifies by inductive reasoning that no sequence of inputs to the design exists for which a state variable $z' \in Z'$ becomes tainted. Hence, no taint can reach any output $y_c \in Y_C$, which means that X_D never influences the attacker-observable control outputs Y_C.

Finally, we emphasize that FastPath does not require sophisticated testbenches to achieve significant efficiency improvements. Because the formal step catches any remaining leaks, the verification engineer does not have to worry about missing corner cases during simulation. In addition, refinements to the IFT flow policy can be applied without worrying about over-constraining the policy.

C. Unique Program Execution Checking (UPEC)

In the final step, we apply an inductive UPEC-DIT [22] property to achieve exhaustive coverage. In the original UPEC-DIT approach, the verification engineer is required to manually inspect *every* propagation originating from the data inputs X_D. This process can be tedious, especially for large data-driven designs, as most propagation alerts are legal, but still require the verification engineer to manually remove the newly reached signal from consideration. In this work, however, we already possess knowledge about possible propagation paths. We use it to skip most of the manual steps of the UPEC-DIT procedure by jumping directly to the, potentially final, inductive step.

```
1 UPEC-DIT(Z', Y_C):
2 assume:
3   at t:              two_safety_eq(Z')
4   during [t, t+1]: software_constraints()
5 prove:
6   at t+1:            two_safety_eq(Z')
7   during [t, t+1]: two_safety_eq(Y_C)
```

Listing 1. Generalized UPEC-DIT interval property

At the end of the IFT-simulation, the set of all untainted state signals Z' is returned to the HFG analysis. Using the structural information, the HFG stage then generates both the 2-safety computational model (see Sec. III-C) and the verification framework. The key formal property is shown in Lst. 1. It employs the following macros:

- *two_safety_eq*(A) specifies that all signals of the given set A are equal between the two instances of the computational model. We use this macro in both the assumption and commitment parts of the property to verify that, when considering all previous propagation paths, no further information flow is possible.
- *software_constraints*() specifies the given usage constraints under which the system operates data-obliviously. These are inherited from the simulation, but may need to be extended if new violations are found using the formal property (see below). However, this did not happen in our experiments.

In case the property check fails, the verification engineer must examine the returned counterexample and identify the root cause. If the counterexample shows a security vulnerability, the design must be fixed and FastPath starts over. If the counterexample shows a scenario which violates data-obliviousness, but can be prevented with a software constraint, we update *software_constraints*(). After adding such a new software constraint, we backtrack to the simulation step (Sec. IV-B), since a new constraint may increase the set of untainted state signals Z'. If a spurious counterexample is found (i.e., an unreachable scenario caused by the symbolic initial state), we refine the property with an invariant (see [22]). Finally, if the counterexample shows a legal data propagation (that was missed by IFT-simulation), we refine the property by removing the corresponding signal from Z'. After refining the property, we simply repeat the formal property check.

If the property is verified successfully, it means that the resulting Z' is a partitioning of the problem and will never be affected by the data inputs X_D under the derived constraints. This results in a fixed point where no control output Y_C is ever influenced by confidential data X_D, and our system is thus data-oblivious according to Def. 1.

V. Experiments

We conducted experiments on a diverse set of designs, including various accelerators and processors. Table I gives an overview of our experiments. All experiments are available in our GitHub repository [31]. We used Cycuity's Radix-S tool [32] for IFT simulation and Siemens EDA's OneSpin 360 tool [33] for formal verification.

For each experiment, we show:

- The result of the analysis. In particular, we report whether the design operates data-obliviously either always (*True*), only

Design	Data-Oblivious	Method	State Size		Data Prop. Found		Manual Inspections		
			Signals	Bits	IFT	+ UPEC	Original [22]	FastPath	Reduction (%)
SHA512	True	HFG	37	2162	–	–	33	0	100.0
AES (opencores)	True	HFG	24	554	–	–	19	0	100.0
AES (secworks)	True	HFG	26	2470	–	–	11	0	100.0
CVA6-DIV	Constrained	UPEC	15	217	10	10	12	3	75.0
FWRISCV-MDS	Constrained	UPEC	13	331	5	8	9	4	55.5
ZipCPU-DIV	False	IFT	14	142	14	–	9	1	88.8
cv32e40s	Constrained	UPEC	471	5624	16	17	30	19	36.6
BOOM	Constrained	UPEC	5340	41948	163	177	185	24	87.0

TABLE I
CASE STUDIES

under certain, reasonable software constraints (*Constrained*), or not at all (*False*). FastPath systematically finds such constraints under which the hardware operates data-obliviously. In our experiments, we also performed a more fine-grained analysis that considers a propagation path for each combination $(x_D, y_C) \in X_D \times Y_C$ of a data input x_D and a control output y_C. For some of the designs, we present these results in more detail in the text below, highlighting interesting findings.

- The step in the FastPath methodology at which the experiment becomes complete, whether by simple structural proof using the HFG (Sec. IV-A), by finding a security issue using IFT simulation (Sec. IV-B), or by exhaustive formal verification using UPEC-DIT (Sec. IV-C).
- The number of state-holding word-level signals and the total number of bits consumed by these signals.
- The number of signals influenced by the data inputs. In particular, we present how many propagations were found by IFT alone and how many were found after employing formal verification, i.e., UPEC-DIT. If the method terminates earlier, e.g., due to a structural proof by the HFG, this step is not required.
- Finally, we measure the amount of manual signal inspection required. We count the number of divergent signals, i.e., signals with different values in the two instances of the 2-safety UPEC model, that are found in counterexamples during the execution of FastPath and require manual inspection. Counterexamples that cause the IFT flow policy to be updated or a constraint or an invariant [21] to be created are also counted. We compare this value to the manual effort of the formal-only approach presented in [22].

A. Cryptographic Accelerators

We analyzed three different crypto-accelerators, one implementing the Secure Hash Algorithm (SHA) [34] and the other two implementing the Advanced Encryption Standard (AES) [35], [36]. We investigated whether any plaintext or key input could affect the control behavior of the system.

Our proposed methodology was able to prove the data obliviousness of all designs through a simple structural analysis using the HFG. The HFG showed that there is no path, implicit or explicit, starting from any of the data inputs X_D to any of the control outputs Y_C. While the absence of such a path is not trivially guaranteed, it can be expected because the round-based nature of the underlying algorithms makes a data dependency very unlikely.

The ability to perform structural analysis trivializes these proofs by eliminating the need for simulation or formal verification. As a result, no manual effort is required beyond the initial annotation of the interface. In contrast, the original UPEC-DIT method [22], for the SHA accelerator, requires a manual inspection of 33 signals influenced by X_D before reaching a fixed point.

B. Division Modules

We verified three division modules taken from processor designs. A structural path between X_D and Y_C exists for all designs, which means that structural analysis alone is not sufficient to verify data-obliviousness.

For the division unit of the ZipCPU [37] project, IFT simulation detected several data dependencies, such as an early termination for a divisor of zero. Since there is no reasonable software constraint under which the design operates data-obliviously, there is no need to proceed to the formal analysis.

The *Featherweight RISC-V* [38] processor employs a single module for multi-cycle multiplication, division and shifting. Using a random pattern testbench, IFT simulation was able to confirm the findings of [22] that the timing of shift operations is dependent on the shift amount. We introduced a constraint to exclude shifting, rerun the simulation and generated formal properties based on the untainted signals Z'. In the formal step, UPEC was able to detect three additional data propagations that were missed due to the simplicity of the testbench. After refining the formal property, FastPath verified that the design can operate data-obliviously if the software satisfies the constraint derived by our method (no shifting).

We also verified a hardened version of the CVA6 division unit taken from [39]. It implements security labels that indicate whether operands are public or confidential. Division timing is dynamically optimized to depend only on public information, but to perform better than worst-case latency. We taint the operands according to their labels, i.e., whenever they are confidential. In this experiment, the IFT simulation reported a false counterexample due to an overly conservative flow policy. In FastPath, however, we do not have to worry about restricting the IFT policy, since the formal step catches any missing propagation paths. In addition, two invariants were required to verify the security mechanism of this design.

C. In-Order Processor

We applied FastPath to cv32e40s [25], a RISC-V processor intended for security applications. It implements a variety of security mechanisms, including a data-independent timing (*data_ind_timing*) mode that ensures that normally variable-time instructions execute with a fixed latency. Some special cases, such as misaligned memory accesses, are excluded. Our results confirm the expectation that CSR accesses, jumps and branches can still influence the control behavior of the system even when in *data_ind_timting* mode. Load and store instructions that access more than a single byte can cause misalignment and thus affect control. In contrast, as intended, the latency

of division instructions becomes constant when the *data_ind_timing* mode is active.

However, FastPath was able to identify another, previously unknown, leakage affecting the operands of *all* instruction types. The method detected that the operands residing in the pipeline buffer of the ID-EX stage are always visible on the primary interface to the data memory, regardless of whether a memory access is taking place. Even if the corresponding validity signal is deasserted, the core still interprets the operands as address and data and passes them on to the outputs. Any IP, faulty or malicious, that does not comply with the protocol, can easily obtain information about internal computations, rendering security mechanisms such as *data_ind_timing* obsolete. We reported the vulnerability and worked with the processor development team to contribute a fix.

We used a fairly rudimentary testbench in our experiment, yet the IFT simulation was able to find almost all data propagations. The only missed state signal was inside the multiplier and is only tainted during MULH instructions. The formal step of FastPath showed a counterexample for this signal. In this experiment, most manual inspections involved deriving constraints for non-oblivious instructions, or writing invariants.

D. Out-of-Order Processor

Finally, to demonstrate its efficiency and scalability, we apply FastPath to the Berkeley Out-of-Order Machine (BOOM) [40] and compare the manual effort with the results of [22]. We confirm their findings regarding the data-obliviousness of certain instructions.

BOOM implements a sophisticated floating point (FP) pipeline, resulting in a large number of signals in the data path. Using a simple testbench, FastPath identifies the vast majority of these signals, as well as non-oblivious instructions, via IFT simulation. This reduces the manual effort of the formal step to corner cases, such as special cases for FP computations, and greatly reduces the overall manual effort for verifying the most complex design in our benchmark suite.

E. Discussion - Runtime and Manual Effort

IFT-simulation and formal property checking produce counterexamples that provide violations of the specified and desired security behavior as described in the security properties. These counterexamples require a manual analysis, during which the verification engineer must determine whether the counterexample describes a scenario that violates the intended design behavior. In other words, the verification engineer must determine if the security properties correctly describe the threat model. Property specification is challenging, and analyzing the counterexamples often provides cases where the properties must be refined. This analysis may take a considerable amount of time, depending on the verification engineer's design knowledge, the design's complexity, and the complexity of the counterexample at hand. For this reason, we use the number of signals that must be inspected manually as a metric for the verification effort. The manual inspection time is hard to quantify but can take weeks to months [21] for the designs considered in this work. As shown in Table I, compared to previous work, FastPath is able to significantly reduce the time spent manually examining counterexamples and thus the overall verification effort.

We observe a reduction of manual effort between 36% and 100%. The smallest reduction is obtained for the cv32e40s processor. This can be explained by the fact that this processor contains a relatively small data path but a complex control structure interweaved with several security features. This results in a comparably large number of counterexamples that lead to invariants or restrictions needed for security-aware software development. Remarkably, for BOOM, by far the largest and most labor-intensive design examined in this work, an 86% reduction of manual effort was achieved. In contrast to cv32e40s, BOOM employs a much larger and more sophisticated data path. In spite of its complex control for instruction scheduling, our hybrid approach is particularly effective because it clearly separates control from data path. Overall, our experiments demonstrate the immense promise of FastPath for increasing productivity in complex design projects.

Tool runtime is an important aspect when assessing the scalability of a verification approach. For FastPath, however, these runtimes were negligible. Running a single IFT-simulation, including the extraction of all tainted signals from the simulation trace, took 1–2 minutes. In the formal step, the initial design elaboration in the formal model checker took around 5 minutes for the most complex experiment. Afterward, a single check of the formal property (Lst. 1) finished in less than 10 seconds by merit of the symbolic initial state. Hence, tool runtime contributes only a small fraction to FastPath's verification time.

VI. Related Work

FastPath combines simulation and formal verification in the context of hardware security. Previous work exploring synergies between the two techniques, such as [41], has mostly attempted to improve simulation efficiency and coverage with formal methods. Consequently, such approaches inherit the limitation of simulation techniques and cannot provide formal guarantees. But there are also efforts in the other direction, i.e., the use of random simulation to improve formal approaches, such as in [42]. However, none of these methods address hardware security issues.

Significant efforts [43], [26], [22], [44], [27], [28], [45] have been made to detect leakage of confidential operands through timing or other microarchitectural side effects. Clepsydra [43] instruments an RTL design with IFT logic to discover timing flows through either simulation or formal verification. UPEC-DIT [22] relies on standard SystemVerilog Assertions (SVA) properties to detect data-dependent side-effects in the microarchitecture. While the symbolic initial state of the approach allows for a very high scalability, its main drawback lies in the considerable manual refinement effort (cf. Sec. V). More recent formal methods take a different approach [27], [28], [45], but have not been shown to scale to complex systems. We believe that also these methods could benefit from a hybrid approach along the lines proposed in this paper.

VII. Conclusion

FastPath is a hybrid verification methodology that leverages the complementary strengths of structural analysis, IFT-simulation, and formal verification to produce exhaustive security guarantees for data-oblivious architectures. FastPath uses a global formulation that automatically transfers intermediate results between the three verification techniques. The key benefit of FastPath is a drastic reduction of manual effort, achieved by combining the efficiency of simulation with the exhaustiveness of formal methods. In our case studies on functional units, FastPath was able to verify (or disprove) data-obliviousness with little to no manual interaction. Furthermore, especially for complex systems with intensive data paths like BOOM, FastPath offers unique scalability both in terms of runtime and manual effort. Thus, FastPath provides a tremendous reduction in security verification effort since the manual inspection dominates the overall verification time. Our future work aims at exploring the synergies of the proposed method for different threat models.

References

[1] Y. Yarom, D. Genkin, and N. Heninger, "CacheBleed: a timing attack on OpenSSL constant-time RSA," *Journal of Cryptographic Engineering*, vol. 7, pp. 99–112, 2017.

[2] P. Kocher, J. Horn, A. Fogh, D. Genkin, D. Gruss, W. Haas, M. Hamburg, M. Lipp, S. Mangard, T. Prescher, M. Schwarz, and Y. Yarom, "Spectre attacks: Exploiting speculative execution," in *40th IEEE Symposium on Security and Privacy (S&P'19)*, 2019.

[3] M. Lipp, M. Schwarz, D. Gruss, T. Prescher, W. Haas, A. Fogh, J. Horn, S. Mangard, P. Kocher, D. Genkin, Y. Yarom, and M. Hamburg, "Meltdown: Reading kernel memory from user space," in *27th USENIX Security Symposium (USENIX Security 18)*, 2018.

[4] J. R. S. Vicarte, M. Flanders, R. Paccagnella, G. Garrett-Grossman, A. Morrison, C. W. Fletcher, and D. Kohlbrenner, "Augury: Using data memory-dependent prefetchers to leak data at rest," in *2022 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2022, pp. 1491–1505.

[5] B. Chen, Y. Wang, P. Shome, C. W. Fletcher, D. Kohlbrenner, R. Paccagnella, and D. Genkin, "GoFetch: Breaking constant-time cryptographic implementations using data memory-dependent prefetchers," in *USENIX Security Symposium*, 2024, pp. 1–21.

[6] J. Yu, M. Yan, A. Khyzha, A. Morrison, J. Torrellas, and C. W. Fletcher, "Speculative taint tracking (STT): A comprehensive protection for speculatively accessed data," in *52nd Annual IEEE/ACM International Symposium on Microarchitecture*, 2019, pp. 954–968.

[7] J. Yu, L. Hsiung, M. El Hajj, and C. W. Fletcher, "Data oblivious ISA extensions for side channel-resistant and high performance computing." in *26th Network and Distributed System Security Symposium*, 2019.

[8] S. Cauligi, C. Disselkoen, K. v. Gleissenthall, D. Tullsen, D. Stefan, T. Rezk, and G. Barthe, "Constant-time foundations for the new spectre era," in *41st ACM SIGPLAN Conference on Programming Language Design and Implementation*, New York, NY, USA, 2020, p. 913–926.

[9] M. Schwarz, M. Lipp, C. Canella, R. Schilling, F. Kargl, and D. Gruß, "ConTExT: A generic approach for mitigating spectre," in *Network and Distributed System Security Symposium 2020*, Feb. 2020.

[10] K. Loughlin, I. Neal, J. Ma, E. Tsai, O. Weisse, S. Narayanasamy, and B. Kasikci, "DOLMA: Securing speculation with the principle of transient Non-Observability," in *30th USENIX Security Symposium (USENIX Security 21)*. USENIX Association, 2021, pp. 1397–1414.

[11] T. Jauch, A. Wezel, M. R. Fadiheh, P. Schmitz, S. Ray, J. M. Fung, C. W. Fletcher, D. Stoffel, and W. Kunz, "Secure-by-construction design methodology for CPUs: Implementing secure speculation on the RTL," in *2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD)*, 2023, pp. 1–9.

[12] B. Coppens, I. Verbauwhede, K. De Bosschere, and B. De Sutter, "Practical mitigations for timing-based side-channel attacks on modern x86 processors," in *30th IEEE Symposium on Security and Privacy*. IEEE, 2009, pp. 45–60.

[13] "A beginner's guide to constant-time cryptography," https://www.chosenplaintext.ca/articles/beginners-guide-constant-time-cryptography.html, accessed: 2024-09-27.

[14] M. Andrysco, A. Nötzli, F. Brown, R. Jhala, and D. Stefan, "Towards verified, constant-time floating point operations," in *2018 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '18, New York, NY, USA, 2018, p. 1369–1382.

[15] "Data operand independent timing ISA guidance," https://www.intel.com/content/www/us/en/developer/articles/technical/software-security-guidance/best-practices/data-operand-independent-timing-isa-guidance.html, acc.: 2024-08-14.

[16] "Arm Armv8-A architecture registers," https://developer.arm.com/documentation/ddi0595/2021-06/AArch64-Registers/DIT--Data-Independent-Timing, accessed: 2024-08-14.

[17] "RISC-V cryptography extension," https://github.com/riscv/riscv-crypto, accessed: 2024-09-25.

[18] J. R. S. Vicarte, P. Shome, N. Nayak, C. Trippel, A. Morrison, D. Kohlbrenner, and C. W. Fletcher, "Opening Pandora's box: A systematic study of new ways microarchitecture can leak private data," in *ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*, 2021.

[19] J. A. Goguen and J. Meseguer, "Security policies and security models," in *IEEE Symposium on Security and Privacy (SP)*, 1982, pp. 11–20.

[20] M. R. Clarkson and F. B. Schneider, "Hyperproperties," *Journal of Computer Security*, vol. 18, no. 6, pp. 1157–1210, 2010.

[21] M. R. Fadiheh, A. Wezel, J. Müller, J. Bormann, S. Ray, J. M. Fung, S. Mitra, D. Stoffel, and W. Kunz, "An exhaustive approach to detecting transient execution side channels in RTL designs of processors," *IEEE Transactions on Computers*, vol. 72, no. 1, pp. 222–235, 2023.

[22] L. Deutschmann, J. Müller, M. R. Fadiheh, D. Stoffel, and W. Kunz, "A scalable formal verification methodology for data-oblivious hardware," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 43, no. 9, pp. 2551–2564, 2024.

[23] W. Hu, A. Ardeshiricham, and R. Kastner, "Hardware information flow tracking," *ACM Comp. Surveys (CSUR)*, vol. 54, no. 4, pp. 1–39, 2021.

[24] A. Meza and R. Kastner, "Information flow coverage metrics for hardware security verification," 2023. [Online]. Available: https://arxiv.org/abs/2304.08263

[25] "OpenHW Group CORE-V CV32E40S RISC-V IP," https://github.com/openhwgroup/cv32e40s, updated: 2024-10-31, accessed: 2024-11-11.

[26] K. v. Gleissenthall, R. G. Kıcı, D. Stefan, and R. Jhala, "Solver-aided constant-time hardware verification," in *2021 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '21, New York, NY, USA, 2021, p. 429–444.

[27] S. Dinesh, M. Parthasarathy, and C. W. Fletcher, "ConjunCT: Learning inductive invariants to prove unbounded instruction safety against microarchitectural timing attacks," in *2024 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2024, pp. 3735–3753.

[28] K. Ceesay-Seitz, F. Solt, and K. Razavi, "μCFI: Formal verification of microarchitectural control-flow integrity," in *2024 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '24, New York, NY, USA, 2024.

[29] K. Ryan, M. Gregoire, and C. Sturton, "Seif: Augmented symbolic execution for information flow in hardware designs," in *12th Int. Workshop on Hardware and Architectural Support for Security and Privacy*, ser. HASP '23. New York, NY, USA: ACM, 2023, p. 1–9.

[30] J. Urdahl, D. Stoffel, and W. Kunz, "Path predicate abstraction for sound system-level models of RT-level circuit designs," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 33, no. 2, pp. 291–304, 2014.

[31] "FastPath repository," https://github.com/anmeza/dac_2025_fastpath.

[32] "Cycuity Radix," https://cycuity.com/.

[33] "Siemens EDA OneSpin 360," https://eda.sw.siemens.com/.

[34] "SHA cores," https://opencores.org/projects/sha_core, updated: 2018-03-06, accessed: 2024-09-26.

[35] "AES IP core," https://opencores.org/projects/aes_core, updated: 2018-06-10, accessed: 2024-09-26.

[36] "secworks aes," https://github.com/secworks/aes, updated: 2023-02-08, accessed: 2024-09-26.

[37] "The Zip CPU," https://github.com/ZipCPU/zipcpu, updated: 2024-08-26, accessed: 2024-10-25.

[38] "FWRISC," https://github.com/Featherweight-IP/fwrisc, updated: 2022-01-17, accessed: 2024-10-25.

[39] L. Deutschmann, Y. Kazhalawi, J. Seckinger, A. L. Duque Antón, J. Müller, M. R. Fadiheh, D. Stoffel, and W. Kunz, "Data-oblivious and performant: On designing security-conscious hardware," in *2024 IEEE 25th Latin American Test Symposium (LATS)*, 2024, pp. 1–6.

[40] J. Zhao, B. Korpan, A. Gonzalez, and K. Asanovic, "SonicBOOM: The 3rd generation berkeley out-of-order machine," in *Fourth Workshop on Computer Architecture Research with RISC-V*, May 2020.

[41] P.-H. Ho, T. Shiple, K. Harer, J. Kukula, R. Damiano, V. Bertacco, J. Taylor, and J. Long, "Smart simulation using collaborative formal and simulation engines," in *IEEE/ACM International Conference on Computer Aided Design (ICCAD)*, 2000, pp. 120–126.

[42] F. Krohm, A. Kuehlmann, and A. Mets, "The use of random simulation in formal verification," in *International Conference on Computer Design*. IEEE, 1996, pp. 371–376.

[43] A. Ardeshiricham, W. Hu, and R. Kastner, "Clepsydra: Modeling timing flows in hardware designs," in *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 2017, pp. 147–154.

[44] Z. Wang, G. Mohr, K. von Gleissenthall, J. Reineke, and M. Guarnieri, "Specification and verification of side-channel security for open-source processors via leakage contracts," in *2023 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '23, New York, NY, USA, 2023, p. 2128–2142.

[45] Y. Hsiao, N. Nikoleris, A. Khyzha, D. P. Mulligan, G. Petri, C. W. Fletcher, and C. Trippel, "RTL2MμPATH: Multi-μPATH synthesis with applications to hardware security verification," in *57th IEEE/ACM International Symposium on Microarchitecture (MICRO '24)*, 2024.