ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Technical Report No. 526

Timing-Sensitive Information Flow Analysis for Synchronous Systems

Boris Köpf and David Basin

Information Security
ETH Zurich, Switzerland
{bkoepf,basin}@inf.ethz.ch

Abstract. Timing side channels are a serious threat to the security of cryptographic algorithms. This paper presents a novel method for the timing-sensitive analysis of information flow in synchronous hardware circuits. The method is based on a parameterized notion of confidentiality for finite transition systems that allows one to model information leakage in a fine-grained way. We present an efficient decision procedure for system security and apply it to discover timing leaks in nontrivial hardware implementations of cryptographic algorithms.

1 Introduction

Timing side channels are a serious threat to the security of cryptographic algorithms [4, 12, 17]. By analyzing the running times of algorithms such as RSA decryption, an attacker may be able to deduce information about the secret key used, possibly even recovering the key in its entirety. Several countermeasures against this threat have been proposed, including blinding and randomization techniques. While these techniques successfully defeat certain known attacks, it is difficult to argue their completeness, in the sense that they defeat all attacks that exploit timing information.

One systematic and complete countermeasure for preventing timing attacks is to ensure that the algorithms' running times are independent of the secrets processed. Agat [1] pursues this approach by giving a security type system for a simple imperative programming language. If a program can be assigned a security type, then its running times are independent of the secrets it computes with, and hence do not reveal this confidential information. This is shown by proving the soundness of the type system with respect to a semantic notion of secure information flow. Although this result (as well as other approaches that use programming language-based models [28, 21, 3]) provides an attractive analysis method, the timing model used is based on a high-level language and is therefore too simplistic. Indeed, if the timing behavior of the underlying hardware is not accurately modeled, it is unclear what is gained from such a formal analysis. Unfortunately, providing precise timing models for today's processors seems out of reach. Giving upper bounds for the real-time behavior of multi-purpose processors is already a daunting task [19] and is still not sufficient for proving the absence of timing leaks.

In this paper, we approach the problem on a different level of abstraction. We focus on special-purpose hardware implementations of cryptographic algorithms, which are particularly important in resource-critical application domains [23]. We develop a method for the information flow analysis of synchronous (clocked) hardware circuits. To this end, we define R_I/R_O-security, a parameterized and timing-sensitive notion of security for Mealy machines, in which each transition corresponds to one clock tick. R_I/R_O-security can be instantiated to standard noninterference definitions, but it can also be used to express that only partial information on each confidential input is revealed through the system's observable behavior. In system runs of arbitrary length, this partial information can accumulate. We show that the guarantees of R_I/R_O-security can be combined with assumptions on the environment to derive an upper bound on the number of distinguishable output behaviors, a measure for what an attacker may learn about the processed secrets. We develop efficient algorithms for deciding whether a finite-state system is R_I/R_O-secure. For deterministic systems, we reduce this to a reachability problem for a special kind of product automaton. In the nondeterministic case, we reduce this to a generalization of the Partition Refinement Problem. We also provide a compositionality result as a first step to scaling-up the analysis method to more complex designs.

Finally, we report on initial experimental results using our method. We have encoded our decision procedures in an off-the-shelf model checker and used it to discover subtle timing side channels in a textbook hardware implementation of a finite-field exponentiation algorithm. The synchronous hardware description language GEZEL [24] provides the link between our analysis method and concrete hardware implementations. Namely, GEZEL allows one to specify synchronous circuits in terms of automata, and it comes with a tool for translating the designs into a subset of the industrial-strength hardware description language VHDL. The translation is *cycle-true*, which means that it preserves the timing behavior within the granularity of clock ticks. Moreover, the output is *synthesizeable*, i.e. it can be mapped to a physical implementation. Hence, the security guarantees obtained using our analysis method translate into guarantees for real-world hardware implementations.

Our main contributions are twofold. First, we extend well-studied notions of information flow security to a model that is appropriate for the analysis of timing side channels in hardware implementations. Second, we develop efficient algorithms for deciding whether a system is secure and we show that they can be practically applied to nontrivial circuits.

The remainder of this paper is structured as follows. In Section 2, we introduce our automaton model and define security. In Section 3, we develop reduction techniques and efficient algorithms for deciding whether an automaton has secure information flow. We report on experimental results in Section 4, before we present related work and draw conclusions in Sections 5 and 6.

2 The Scenario

2.1 Machine Model

We use Mealy machines as a model for hardware circuits that are synchronized by a global clock signal. We assume that one transition corresponds to one clock cycle and that, during each clock cycle, input signals are read and output signals generated. Furthermore, we assume input enabledness, that is, the machine can always react to every possible input. While hardware is typically designed to be deterministic, nondeterminism is useful too, for example, for modeling requirements, and hence we will keep our presentation general wherever possible. As there is no standard notion of a nondeterministic Mealy machine, we use the term *automaton with output.*

Definition 1. *An* automaton with output *is a 5-tuple* $M = (S, \Sigma, \Gamma, \delta, s_0)$*, where* S *is a finite set of states,* Σ *is a finite input alphabet,* Γ *is a finite output alphabet,* $\delta \subseteq S \times \Sigma \times \Gamma \times S$ *is a transition relation, and* $s_0 \in S$ *the initial state. We call* M deterministic *if for every* $(s, a) \in S \times \Sigma$ *there is at most one* $(b, s') \in \Gamma \times S$ *with* $(s, a, b, s') \in \delta$*.*

In a transition (s, a, b, s'), a denotes the input and b denotes the output. We will sometimes use the shorthand $\delta(s, a, b)$ to denote the set $\{s' \mid (s, a, b, s') \in \delta\}$. As noted above, we require the transition relation to be *total*, i.e. for all $s \in S$ and $a \in \Sigma$, there is at least one $b \in \Gamma$ and one $s' \in S$ with $(s, a, b, s') \in \delta$. We do not consider ϵ-transitions as they contradict the assumption that input and output are provided during each clock cycle. In the setting of hardware circuits, Σ and Γ will be of the form $\{0, 1\}^n$, for some $n \in \boldsymbol{N}$, and represent the values of all ingoing and outgoing signals.

2.2 Defining Security

We specify security with respect to an observer of the system. An observer is modeled in terms of its capabilities for distinguishing different system behaviors. If all system runs are indistinguishable, even when the system computes with different secret data, then the system is intuitively secure. Conversely, information may leak if the system shows distinguishable behavior while processing different secrets.

Distinguishing atomic inputs/outputs. The fact that two outputs $a, b \in \Gamma$ are indistinguishable is captured by an equivalence relation $R_O \subseteq \Gamma \times \Gamma$. We say that a and b are *observationally equivalent*, or simply R_O-*equivalent*, if and only if $a \; R_O \; b$. In other words, if the system outputs $x \in \Gamma$, an observer can only deduce the R_O-equivalence class $[x]$. Similarly, we use the equivalence relation $R_I \subseteq \Sigma \times \Sigma$ to model to what extent an observer can distinguish the input of the system.

In the following, Id_X denotes the identity on a set X and All_X denotes $X \times X$. For relations $R \subseteq \Gamma_1 \times \Gamma_1$ and $Q \subseteq \Gamma_2 \times \Gamma_2$, we overload notation and define $R \times Q \subseteq (\Gamma_1 \times \Gamma_2)^2$ as $(r_1, q_1)\ (R \times Q)\ (r_2, q_2)$ if and only if $r_1\ R\ r_2$ and $q_1\ Q\ q_2$.

Example 1. The relation $R_O = \mathrm{All}_\Gamma$ formalizes an observer who cannot distinguish between any two system outputs. In contrast, the relation $R_O = \mathrm{Id}_\Gamma$ models an observer who can determine the (singleton) Id_Γ-equivalence class of the output, or equivalently, who can see the entire output. We can also model more fine-grained capabilities. Consider, for example, $\Gamma = \Gamma_1 \times \Gamma_2$, with $\Gamma_1 = \{0,1\}^n$, and the predicate $\Psi_{\Gamma_1} = \{(a, b) \in \Gamma_1 \times \Gamma_1 \mid \|a\| = \|b\|\}$, where $\|x\|$ denotes the Hamming weight of x, i.e. the number of bits set to 1. The relation $\Psi_{\Gamma_1} \times \mathrm{Id}_{\Gamma_2}$ models that an observer can see the entire Γ_2-component of the output, but can only deduce the Hamming weight (determine the Ψ_{Γ_1}-equivalence class) of the Γ_1-component. ◇

Expressing security. Two states of a system are *observationally equivalent* if every output from one state can be matched by an R_O-equivalent output from the other state whenever the corresponding inputs are R_I-equivalent. We call the observational equivalence of states *R_I/R_O-equivalence*, which is a *partial equivalence relation* (PER), i.e. symmetric and transitive, but not necessarily reflexive. If the initial state of a system is *not* observationally equivalent to itself, then running the system on R_I-equivalent input sequences may lead to observable differences in the system behavior. This constitutes a refinement of an observer's knowledge about the input (modeled by R_I), and thus is an information leak. If, on the other hand, the initial state is observationally equivalent to itself, then we say that the system is *R_I/R_O-secure*. The idea that security can be modeled as a system being observationally equivalent to itself is formalized in the PER model of secure information flow [22].

The next section gives a formal account of these ideas.

2.3 A Parameterized Notion of Observational Equivalence

For the systems under consideration, we model observational equivalence of states by using a parameterized notion of strong bisimulation. This will capture timing behavior, as every transition corresponds to a tick of the global clock, and strong bisimulation equivalence allows one to distinguish process behaviors that differ in the number of transitions leading to some output.

Definition 2 (R_I/R_O-Equivalence). *Let $M = (S, \Sigma, \Gamma, \delta, s_0)$ be an automaton with output, and let $R_I \subseteq \Sigma^2$ and $R_O \subseteq \Gamma^2$ be equivalence relations. We define $\simeq^{R_I}_{R_O}$ as the union of all symmetric and transitive* *relations $\mathcal{R}$ on S with the property that for all $s_1, s_2 \in S$:*

$$s_1 \mathrel{\mathcal{R}} s_2 \Rightarrow \forall a_1, a_2 \in \Sigma.(a_1 \; R_I \; a_2 \Rightarrow \forall (s_1, a_1, o_1, s_1') \in \delta. \\ \exists (s_2, a_2, o_2, s_2') \in \delta. \\ s_1' \mathrel{\mathcal{R}} s_2' \wedge o_1 \; R_O \; o_2). \tag{1}$$

*Two states $s_1, s_2 \in S$ are R_I/R_O-*equivalent *iff $s_1 \simeq_{R_O}^{R_I} s_2$.*

Definition 3 (R_I/R_O-Security). *Let $M = (S, \Sigma, \Gamma, \delta, s_0)$ be an automaton with output, and let $R_I \subseteq \Sigma^2$ and $R_O \subseteq \Gamma^2$ be equivalence relations. Then M is R_I/R_O-*secure *iff $s_0 \simeq_{R_O}^{R_I} s_0$.*

It is easy to see that $\simeq_{R_O}^{R_I}$ is a partial equivalence relation on S and that $\simeq_{R_O}^{R_I}$ itself satisfies Property (1) of Definition 2.

We will now give several instances of R_I/R_O-security. We first show how it encompasses a number of security notions from language-based information-flow (for an overview of this area, see [20]). Afterwards, we instantiate R_I/R_O-security to specify partial information flow, which will prove to be useful in our experiments.

In the following examples, we assume two security domains, *high* and *low*, and we restrict the flow of information from the high domain to the low domain. A common assumption in programming language-based approaches is that each variable is classified as either high or low and that an observer may only see the values of the low variables. In our setting, input and output signals take the role of variables and have high and low components. This intuition is reflected by assuming that $\Sigma = \Sigma_H \times \Sigma_L$, where Σ_H and Σ_L represent the values of all high and low input signals, respectively. Similarly, we assume that $\Gamma = \Gamma_H \times \Gamma_L$. The policy that no information flows from the high into the low domain can then be formalized in the framework of R_I/R_O-equivalence by choosing $R_I = \mathrm{All}_{\Sigma_H} \times \mathrm{Id}_{\Sigma_L}$ and $R_O = \mathrm{All}_{\Gamma_H} \times \mathrm{Id}_{\Gamma_L}$. When Σ is understood, we write Id_L as an abbreviation for Id_{Σ_L}. We abbreviate analogously for Γ, All_L and the high domain.

Example 2. In the deterministic case, $\simeq_{\mathrm{All}_H \times \mathrm{Id}_L}^{\mathrm{All}_H \times \mathrm{Id}_L}$ represents a notion of observational equivalence closely related to Agat's *Γ-bisimulation* [1]. In our model, every transition takes one time unit, while in Agat's approach the duration of each transition is given by a label representing the primitive operations of the underlying machine. $\diamond$

Example 3. In the nondeterministic case, $\simeq_{\mathrm{All}_H \times \mathrm{Id}_L}^{\mathrm{All}_H \times \mathrm{Id}_L}$ represents a possibilistic notion of security similar to Volpano and Smith's *concurrent noninterference* [25], which has been used to model the security of multithreaded programs in the presence of a purely nondeterministic scheduler. Note that our definition is more restrictive with respect to timing, as it is based on strong bisimulation equivalence as opposed to the weak bisimulation-based concurrent noninterference. $\diamond$

In addition to capturing variants of previously studied notions of security, R_I/R_O-equivalence allows one to express more fine-grained forms of information flow.

Example 4. Consider the binary predicate $\Psi_\Sigma = \{(a,b) \in \Sigma \times \Sigma \mid \|a\| = \|b\|\}$, where $\Sigma = \{0,1\}^n$ and $\|x\|$ denotes the Hamming weight of x. Suppose we have $s_0 \simeq^{\Psi}_{\mathrm{Id}_\Gamma} s_0$, where s_0 is the initial state of a deterministic system. Then the system shows the same behavior for each pair of (and hence, by transitivity of $\simeq^{\Psi}_{\mathrm{Id}_\Gamma}$, for all) input traces $a_1 \cdots a_m$ and $b_1 \cdots b_m$, where $\|a_i\| = \|b_i\|$ for every $i \in \{1, \ldots, m\}$. $\diamond$

The converse of Example 4 is more subtle. An observable difference between two output traces of a deterministic system implies that the input traces' Hamming weight differs at some point in time. While the leakage of a single input's Hamming weight might be acceptable, the leakage of the Hamming weight of all symbols in the input trace can be used to encode arbitrary information.

Example 5. Consider an automaton M with a single state s_0, alphabets $\Sigma_H = \Sigma_L = \Gamma = \{0,1\}$, and transitions $\{(s_0, (h,l), h, s_0) \mid h,l \in \{0,1\}\}$. M maps every high input h to the identical output. Still, M is $\Psi_H \times \mathrm{Id}_L/\mathrm{Id}_\Gamma$-secure. $\diamond$

To assess how much can be learned by observing the behavior of a R_I/R_O-secure system, we need to consider the environment that provides it with high input.

2.4 Environment Behavior

In this section, we consider the interaction of a deterministic automaton $M = (S, \Sigma_H \times \Sigma_L, \Gamma, \delta, s_0)$ with an environment that provides it with high input. We will show how to combine security guarantees for M with restrictions on the environment to give bounds on the number of distinguishable behaviors. This is a useful measure for assessing the information leakage from the high to the low domain since, for a deterministic system and an arbitrary low input, variations in the output are necessarily due to variations in the high input. Thus, a greater number of distinguishable output traces means that more information about the high input is leaked.

We specify the *high environment* as a subset $E \subseteq \Sigma_H^*$ of high input traces of the form $\bigcup_{i=0}^{\infty} \circ_{j=1}^{i} A_j$, where $A_j \subseteq \Sigma_H$ represents the set of possible inputs from the high environment at time instant j and $\circ$ denotes word concatentation. Dually to the requirement that M is input enabled, we require the high environment to provide an input at every clock cycle. For an arbitrary trace $w \in \Sigma_L^*$, a high environment E, and $R_O \subseteq \Gamma \times \Gamma$, we denote the set of *distinguishable behaviors* by $\mathcal{B}_{M,R_O}(E,w)$. Concretely, for $(s,a,b,s') \in \delta$, we define $\lambda_{R_O}(s,a) = [b]$, where $[b]$ denotes the R_O-equivalence class of b. We canonically extend λ_{R_O} to a mapping from input traces to R_O^*-equivalence classes of output traces. Here, the relation R_O^*

is defined as $\bigcup_{i=0}^{\infty} R_O^k$ where $a_1 \cdots a_k \; R_O^k \; b_1 \cdots b_k$ iff $a_i \; R_O \; b_i$, for all $i \in \{1, \ldots, k\}$. Now we can formally define $\mathcal{B}_{M,R_O}(E, w) = \{\lambda_{R_O}(s_0, \langle v, w \rangle) \mid v \in E, \; |w| = |v|\}$, where $|\cdot|$ is the length function and where $\langle v, w \rangle$ denotes the trace in $(\Sigma_H \times \Sigma_L)^*$ obtained by pairing corresponding elements of v and w.

Recall that if a $Q \times \mathrm{Id}_L/R_O$-secure deterministic system is provided with input from a high environment E in which all traces of the same length are Q^*-equivalent, then it will produce only R_O^*-equivalent output. That is, $|\mathcal{B}_{M,R_O}(E, w)| = 1$ for every $w \in \Sigma_L^*$. If we weaken the requirement that the input is always Q-equivalent, the number of distinguishable behaviors may increase. The next theorem gives an upper bound for this number.

Theorem 1. *Let $M = (S, \Sigma_H \times \Sigma_L, \Gamma, \delta, s_0)$ be a deterministic automaton with output, let $Q \subseteq \Sigma_H \times \Sigma_H$ and $R_O \subseteq \Gamma \times \Gamma$ be equivalence relations, and let $E = \bigcup_{i=0}^{\infty} \circ_{j=1}^{i} A_j \subseteq \Sigma_H^*$ be a high environment. If M is $(Q \times Id_L)/R_O$-secure, then for all $w \in \Sigma_L^*$ we have*

$$|\mathcal{B}_{M,R_O}(E, w)| \;\leq\; \Pi_{j=1}^{\infty} |A_j/_Q| \; .$$

Proof. It suffices to prove $|\{\lambda_{R_O}(s_0, \langle v, w \rangle) \mid v \in \circ_{j=1}^{k} A_j\}| \;\leq\; \Pi_{j=1}^{k} |A_j/_Q|$ for all $w \in \Sigma_L^k$, as taking the limit $k \to \infty$ then leads to the desired result. We define the mapping $\lambda'_w : \circ_{j=1}^{k} A_j/_{Q^k} \to \{\lambda_{R_O}(s_0, \langle v, w \rangle) \mid v \in \circ_{j=1}^{k} A_j\}$ by $\lambda'_w([u]) = \lambda_{R_O}(s_0, \langle u, w \rangle)$. λ'_w is well-defined since $\lambda_{R_O}(s_0, \langle v, w \rangle) = \lambda_{R_O}(s_0, \langle v', w \rangle)$ for all v, v' with $v \; Q^k \; v'$. Note that λ'_w is surjective and that the range of a function is of cardinality less or equal than its domain, hence $|\{\lambda_{R_O}(s_0, \langle v, w \rangle) \mid v \in \circ_{j=1}^{k} A_j\}| \leq |\circ_{j=1}^{k} A_j/_{Q^k}|$ holds. We conclude with $|\circ_{j=1}^{k} A_j/_{Q^k}| = \Pi_{j=1}^{k} |A_j/_Q|$. $\square$

Note that the mapping λ'_w from the proof of Theorem 1 expresses the correpondence between equivalence classes of high input and output behaviors. The fact that its domain is independent of w shows that an attacker cannot learn more than the Q^*-equivalence class of a fixed high input, even if he runs the system with all possible low inputs.

Example 6. If a system is $\mathrm{All}_H \times \mathrm{Id}_L/\mathrm{All}_H \times \mathrm{Id}_L$-secure (see Example 3) and is provided with input from a high environment $E = \bigcup_{i=0}^{\infty} \circ_{j=1}^{i} A_j \subseteq \Sigma_H^*$, then the number of distinguishable output behaviors is $\Pi_{i=1}^{\infty} |A_j/_{\mathrm{All}_H}| = 1$. That is, the system can be securely operated in an arbitrary high environment. $\diamond$

Example 7. Consider again the $\Psi_H \times \mathrm{Id}_L/\mathrm{Id}_\Gamma$-secure automaton M from Example 5. The number of possible system behaviors is unbounded, as $|\{0, 1\}/_{\Psi_H}| = 2$ and $\Pi_{j=1}^{\infty} |\{0, 1\}/_{\Psi_H}|$ diverges. This estimation is tight in the sense that an arbitrary amount of information can be leaked. $\diamond$

Example 8. Consider an $\Psi_H \times Id_L/\mathrm{All}_H \times Id_L$-secure circuit in which the high component is initialized during the first clock tick and subsequent high input is 0. The environment here is given by $E = \bigcup_{i=0}^{\infty} \circ_{j=1}^{i} A_j$ where

$A_1 = \Sigma_H$ and $A_j = \{0\}$ for $j > 1$. Then, for each low input w, the system shows at most $|\Sigma_H/_{\Psi_H}|$ distinguishable behaviors, each of which corresponds to one Ψ_H-equivalence class. This correspondence is given by the mapping λ'_w from the proof of Theorem 1. Thus at most the secret input's Hamming weight is leaked during execution. $\diamond$

3 Deciding R_I/R_O-Equivalence

In this section, we reduce the question of deciding R_I/R_O-equivalence to tractable, well-understood problems and we analyze the complexity of the resulting algorithms. We start with the case when the automata are deterministic, which turns out to be very efficiently solvable.

3.1 Deterministic Case

We first reduce the problem of deciding the R_I/R_O-equivalence of states to a reachability problem for a special type of product automaton. This may seem surprising as, in general, information flow properties are properties of sets of traces rather than properties of individual traces [15]. The key idea behind our construction is that every trace of the product automaton corresponds to a pair of traces of the original system. Taking the transitivity of R_I/R_O-equivalence into account, it suffices to analyze each individual trace of the product automaton in order to establish R_I/R_O-security for the original system.

Definition 4. *Let $M_i = (S_i, \Sigma, \Gamma, \delta_i, s_{0,i})$, with $i \in \{1,2\}$, be deterministic automata with output, and let R_I and R_O be equivalence relations on Σ and Γ, respectively. Then $M_1 \times_{R_O}^{R_I} M_2$ is the automaton $(S_1 \times S_2, R_I, \{0,1\}, \delta', (s_{0,1}, s_{0,2}))$, where*

$$\delta' = \{((s_1,s_2),(a,b),\chi,(t_1,t_2)) \mid a\ R_I\ b \wedge (\chi = \mathit{if}\ c\ R_0\ d\ \mathit{then}\ 1\ \mathit{else}\ 0) \wedge (s_1,a,c,t_1) \in \delta_1 \ \wedge\ (s_2,b,d,t_2) \in \delta_2\}\,.$$

A *falsifying state* is a state with an outgoing transition labeled with 0. We now show that deciding observational equivalence of states is equivalent to determining whether a falsifying state can be reached in $M \times_{R_O}^{R_I} M$.

Theorem 2. *Let $M = (S, \Sigma, \Gamma, \delta, s_0)$ be a deterministic automaton with output, $R_I \subseteq \Sigma \times \Sigma$ and $R_O \subseteq \Gamma \times \Gamma$ equivalence relations, and let $s_1, s_2 \in S$. Then*

$$s_1 \simeq_{R_O}^{R_I} s_2 \Leftrightarrow \textit{no falsifying state is reachable from } (s_1,s_2) \textit{ in } M \times_{R_O}^{R_I} M\,.$$

Proof. ($\Rightarrow$) We show that no input $w \in (R_I)^*$ can trigger a transition labeled with 0. We proceed by induction on the length of w. The assertion is clear for $w = \epsilon$. Suppose now that $w = (a,b)w'$. As $s_1 \simeq_{R_O}^{R_I} s_2$ and δ is total and deterministic, there are unique transitions (s_1,a,c,t_1) and $(s_2,b,d,t_2) \in \delta$, with $t_1 \simeq_{R_O}^{R_I} t_2$ and $(c,d) \in R_O$. Hence $M \times_{R_O}^{R_I} M$ outputs 1 on this transition and we apply the induction hypothesis to (t_1,t_2) and w'.

($\Leftarrow$) We show that $\mathcal{Q} = \{(t_1, t_2) \mid (t_1, t_2) \text{ can be reached from } (s_1, s_2)\}$ fulfills (1) of Definition 2. Pick $(t_1, t_2) \in \mathcal{Q}$ and $(a, b) \in R_I$. Since δ is total and deterministic, there are unique transitions (t_1, a, c, t_1') and $(t_2, b, d, t_2') \in \delta$. Clearly, (t_1', t_2') can also be reached from (s_1, s_2) in $M \times_{R_O}^{R_I} M$ and, as no transition labeled with 0 can be triggered by assumption, $(c, d) \in R_O$ holds. Hence $\mathcal{Q}$ is contained in the union of all relations with (1) of Definition 2. $\square$

This theorem justifies a simple decision procedure where we decide R_I/R_O-equivalence by searching the product automaton from Definition 4. We use breadth-first search, as it will find a shortest path to a falsifying state.

Corollary 1. *Let $M = (S, \Sigma, \Gamma, \delta, s_0)$ be a deterministic automaton with output, let $s_1, s_2 \in S$, and let $R_I \subseteq \Sigma$ and $R_O \subseteq \Gamma$ be equivalence relations. Then $s_1 \simeq_{R_O}^{R_I} s_2$ can be decided in time $\mathcal{O}(|S|^2|R_I|)$, given the product automaton $M \times_{R_O}^{R_I} M$.*

Proof. Breadth-first search can be implemented in time $\mathcal{O}(|V|+|E|)$ on a graph $G = (V, E)$. $M \times_{R_O}^{R_I} M$ has $|S|^2$ states and $|S|^2|R_I|$ transitions. This yields an $\mathcal{O}(|S|^2|R_I|)$ upper bound for the time complexity of deciding R_I/R_O-equivalence. $\square$

3.2 Nondeterministic Case

A straightforward extension of the above reduction does not appear possible in the nondeterministic case. Instead, we use the Partition Refinement Problem [18] as a starting point for deciding R_I/R_O-equivalence.

A *partition* of a set S is a set $\pi = \{A_1, \ldots, A_n\}$ of pairwise disjoint *blocks* with the property that $\bigcup_{i=1}^n A_i = S$. A *refinement* of a partition π is a partition π' such that every block of π' is contained in some block of π. A partition π can also be formalized in terms of an equivalence relation R_π, where the elements of π correspond to the equivalence classes of R_π. The Partition Refinement Problem is, given a partition π and a property P, to find the coarsest refinement π' of π such that π' fulfills P. This is equivalent to finding the greatest equivalence relation $R_{\pi'}$, with $R_{\pi'} \subseteq R_\pi$, such that $R_{\pi'}$ satisfies P.

Since R_I/R_O-equivalence is a partial equivalence relation, PER for short, we need to generalize the Partition Refinement Problem. The *Partial Partition Refinement Problem* is, given a partial equivalence relation R and a property P, to find the coarsest refinement R' of R, such that R' satisfies P. We next show that the problem of deciding R_I/R_O-equivalence can be cast as an instance of this problem. Then, following the ideas in [11], we compute this coarsest refinement as the maximal fixed point of a monotone mapping Φ.

The *domain* of a partial equivalence relation R is the set $\mathrm{dom}\,(R) = \{x \in S \mid x \; R \; x\}$ on which R is reflexive, and hence an equivalence relation. A *partial partition* of a set S is a pair $\langle\{A_1, \ldots, A_n\}, C\rangle$, where the A_i are pairwise disjoint blocks with $\bigcup_{i=1}^n A_i \cup C = S$ and $\bigcup_{i=1}^n A_i \cap C = \emptyset$. There is a one-to-one correspondence between PERs R of a set S and partial partitions $\langle\{A_1,\dots,A_n\},C\rangle$, where the A_i correspond to the equivalence classes of R, and $C = S \setminus \mathrm{dom}\,(R)$. As notation, we denote this correspondence as $\langle\{A_1,\dots,A_n\},C\rangle \mathrel{\hat{=}} \langle R,C\rangle$. Let $\pi_1 = \langle\{A_1,\dots,A_n\},C_1\rangle$ and $\pi_2 = \langle\{B_1,\dots,B_m\},C_2\rangle$ be partial partitions of S. We define $\pi_1 \leq \pi_2$ to hold whenever $C_1 \supseteq C_2$ and if every block of π_1 is contained in some block of π_2. The relation $\leq$ is a partial order on the set of all partial partitions of a set S. In fact, it is also a lattice when we define the meet $\sqcap$ as $\langle\{A_1,\dots,A_n\},C_1\rangle \sqcap \langle\{B_1,\dots,B_m\},C_2\rangle = \langle\{E_{i,j}\},C_1 \cup C_2\rangle$, with $E_{i,j} = A_i \cap B_j$ for $i \in \{1,\dots,n\}$ and $j \in \{1,\dots,m\}$.

In the remainder of this subsection, let $M = (S,\Sigma,\Gamma,\delta,s_0)$ be a nondeterministic automaton with output, and let $R_I \subseteq \Sigma\times\Sigma$ and $R_O \subseteq \Gamma\times\Gamma$ be equivalence relations.

Definition 5 (R_I/R_O-partition). *A R_I/R_O-*partition *of S is a partial partition $\langle\{A_1,\dots,A_n\},C\rangle$ of S, with*

$$\begin{array}{l} \forall i,j \in \{1,\dots,n\}.\ \forall s_1,s_2 \in A_i.\ \forall (a_1,a_2) \in R_I.\ \forall x \in \Gamma/_{R_O}. \\ \quad \overline{\delta}(s_1,a_1,x) \cap A_j \neq \emptyset \ \Leftrightarrow\ \overline{\delta}(s_2,a_2,x) \cap A_j \neq \emptyset\ \wedge \\ \quad \overline{\delta}(s_1,a_1,x) \cap C \ =\ \overline{\delta}(s_2,a_2,x) \cap C \ =\ \emptyset\,, \end{array} \tag{2}$$

where $\overline{\delta}(s,a,x)$ denotes the set $\bigcup_{c\in x}\delta(s,a,c)$. A R_I/R_O-partition π of S is maximal *if $\pi \geq \pi'$ holds for every R_I/R_O-partition π' of S.*

We adapt (2) of Definition 5 to a mapping on partial partitions whose fixed points are precisely the R_I/R_O-partitions of S. To this end, let $\pi = \langle\{A_1,\dots,A_n\},C_1\rangle \mathrel{\hat{=}} \langle R_1,C_1\rangle$ be a partial partition of S. We define $\Phi(\pi) := \langle R_2, S\setminus \mathrm{dom}\,(R_2)\rangle$, where $s_1\ R_2\ s_2$ if and only if

$$\begin{array}{l} s_1\ R_1\ s_2 \ \wedge\ \forall j \in \{1,\dots,n\}.\ \forall (a_1,a_2) \in R_I.\ \forall x \in \Gamma/_{R_O}. \\ \qquad\qquad \overline{\delta}(s_1,a_1,x) \cap A_j \neq \emptyset \ \Leftrightarrow\ \overline{\delta}(s_2,a_2,x) \cap A_j \neq \emptyset\ \wedge \\ \qquad\qquad \overline{\delta}(s_1,a_1,x) \cap C_1 \ =\ \overline{\delta}(s_2,a_2,x) \cap C_1 \ =\ \emptyset\,. \end{array}$$

Lemma 1. *Let $\langle R,C\rangle$ be a partial partition of the set of states S. Then the following are equivalent:*

1. *$\langle R,C\rangle$ is a fixed point of Φ.*
2. *$\langle R,C\rangle$ is a R_I/R_O-partition of S.*
3. *R satisfies (1) of Definition 2.*

Proof. (*1. $\Rightarrow$ 2.*) The assertion follows by setting $R_1 = R_2 = R$ in the definition of Φ, and observing that two states s_1 and s_2 relate in R whenever they are contained in the same set A_i of the corresponding partial partition.

(*2. $\Rightarrow$ 3.*) Let $s_1\ R\ s_2$, $a_1\ R_I\ a_2$, and $(s_1,a_1,c_1,s_1') \in \delta$. As $\overline{\delta}(s_1,a_1,[c_1]) \cap C = \emptyset$, we have $s_1' \in \overline{\delta}(s_1,a_1,[c_1]) \cap A$ for some equivalence class A of R. By hypothesis, we also have $\overline{\delta}(s_2,a_2,[c_1]) \cap A \neq \emptyset$, and hence there is a transition $(s_2,a_2,c_2,s_2') \in \delta$ with $c_1\ R_O\ c_2$ and $s_1'\ R\ s_2'$.

(*3.* $\Rightarrow$ *1.*) Let $\langle R, C\rangle \mathrel{\hat{=}} \langle\{A_1,\ldots,A_n\}, C\rangle$ and $\Phi(\langle R, C\rangle) = \langle R', C'\rangle$. It suffices to show that $R' = R$. The implication $R' \subseteq R$ follows directly from the definition of Φ. To show that $R \subseteq R'$, choose $s_1\ R\ s_2$ and $a_1\ R_I\ a_2$. If $\overline{\delta}(s_1, a_1, x) \cap A_j \neq \emptyset$, then there is a $(s_1, a_1, c_1, s_1') \in \delta$ with $c_1 \in x$ and $s_1' \in A_j$. As R satisfies (1) of Definition 2, there is also a $(s_2, a_2, c_2, s_2') \in \delta$, with $c_2 \in x$ and $s_2' \in A_j$. Hence $\overline{\delta}(s_2, a_2, x) \cap A_j \neq \emptyset$, and $R \subseteq R'$ follows. □

From Lemma 1, it follows that the relation $\simeq_{R_O}^{R_I}$ is a maximal fixed point of the function Φ. In particular, $\simeq_{R_O}^{R_I}$ itself satisfies (1) of Definition 2 and is thus contained in every maximal fixed point of Φ. Conversely, as every fixed point of Φ satisfies Property (1), the maximal fixed point is contained in $\simeq_{R_O}^{R_I}$, the union of all such relations.

The following theorem gives a constructive way to derive maximal R_I/R_O-partitions.

Theorem 3. *There exists a unique maximal R_I/R_O-partition π^* of S, namely, $\pi^* = \Phi^n(\langle\{S\}, \emptyset\rangle)$, for some $n \in \boldsymbol{N}$.*

Proof. We first show that Φ is monotone with respect to $\leq$ and then apply the Knaster-Tarski fixed-point theorem to obtain a unique maximal fixed point. For the monotonicity consider the partial partitions $\pi_1 = \langle\{A_1,\ldots,A_n\}, C_1\rangle \mathrel{\hat{=}} \langle Q_1, C_1\rangle$ and $\pi_2 = \langle\{B_1,\ldots,B_m\}, C_2\rangle \mathrel{\hat{=}} \langle Q_2, C_2\rangle$, where $\pi_1 \leq \pi_2$. Furthermore, let $\Phi(\pi_1) = \langle Q_1', C_1'\rangle$ and $\Phi(\pi_2) = \langle Q_2', C_2'\rangle$. We need to show that $s_1\ Q_1'\ s_2$ implies $s_1\ Q_2'\ s_2$. Assume $s_1\ Q_1'\ s_2$. By the definition of Φ, this implies $s_1\ Q_1\ s_2$, which implies $s_1\ Q_2\ s_2$. Furthermore, for all $(a_1, a_2) \in R_I$, and for all $x \in \Gamma/_{R_O}$, we have $\overline{\delta}(s_1, a_1, x) \cap C_1 = \overline{\delta}(s_2, a_2, x) \cap C_1 = \emptyset$. As $C_1 \supseteq C_2$, we also have $\overline{\delta}(s_1, a_1, x) \cap C_2 = \overline{\delta}(s_2, a_2, x) \cap C_2 = \emptyset$. Finally, let $(a_1, a_2) \in R_I$ and $x \in \Gamma/_{R_O}$, and suppose $s_1' \in \overline{\delta}(s_1, a_1, x) \cap B_i$. s_1' is also contained in some $A_j \subseteq B_i$, as otherwise this would contradict the assumption $\overline{\delta}(s_1, a_1, x) \cap C_1 = \emptyset$. Then, as $s_1\ Q_1'\ s_2$, we also have $\overline{\delta}(s_2, a_2, x) \cap A_j \neq \emptyset$. Hence we conclude $\overline{\delta}(s_2, a_2, x) \cap B_i \neq \emptyset$. The proof that $\overline{\delta}(s_2, a_2, x) \cap B_i \neq \emptyset$ implies $\overline{\delta}(s_1, a_1, x) \cap B_i \neq \emptyset$ follows along the same lines and concludes the proof of the monotonicity of Φ.

As S is finite, the lattice of partial partitions of S is also finite and hence complete. The Knaster-Tarski fixed-point theorem guarantees the existence of a unique maximal fixed point π^*. We have $\Phi(\pi) \leq \pi$ for every partial partition π of S, and hence iteratively applying Φ to $\pi_\top = \langle\{S\}, \emptyset\rangle$ leads to the fixed point $\pi^* = \Phi^n(\pi_\top)$ after a finite number of steps n. By Lemma 1, this fixed point is also a maximal R_I/R_O-partition. □

Theorem 3 provides the basis of an efficient algorithm for deciding the R_I/R_O-equivalence of states.

Corollary 2. *For two states $s_1, s_2 \in S$ we can decide $s_1 \simeq_{R_O}^{R_I} s_2$ in time*

$$\mathcal{O}(|S|^4 \cdot |R_I| \cdot |\Gamma/_{R_O}|)\,,$$

under the assumption that $\overline{\delta}(s, a, x) = \bigcup_{c \in x} \delta(s, a, c)$ is given as an array indexed by $s \in S$, $a \in \Sigma$, and $x \in \Gamma/_{R_O}$.

Proof. It suffices to show that a single application of Φ can be computed in time $\mathcal{O}(|S|^3 \cdot |R_I| \cdot |\Gamma/_{R_O}|)$. Due to Theorem 3, a fixed point can be obtained by iteratively applying Φ. As $\Phi(\pi) \leq \pi$ for every partition π, this process terminates within at most $|S|$ applications.

We assume $S = \{s_1, \ldots, s_n\}$ and that the equivalence class of each state is given by a representative s_i with minimal i, and by a distinguished symbol $* \notin S$ if the state is outside the domain of the relation. For example, in the case of $\pi_\top = \langle \{S\}, \emptyset \rangle$, the canonical representative for every state is s_1. Suppose now we are given a partial partition $\pi = \langle R, C \rangle$ and we want to compute $\Phi(\pi) = \langle R', C' \rangle$. To decide whether two states s_i and s_j relate in R', we perform the following procedure: for all $(a_1, a_2) \in R_I$, and for all $x \in \Gamma/_{R_O}$, we compare the corresponding sets of R-equivalence classes of the target states. If all of the corresponding sets coincide, s_i and s_j are in the same R'-equivalence class. By iterating i stepwise from 1 to n, we perform this check for every $j \in \{1, \ldots, n\}$. Under this ordering, the canonical representative of the R'-equivalence class of each s_j is the s_i with minimal index such that equivalence of s_i and s_j can be established, and $*$ if there is no such s_i. In this way, each application of Φ can be computed in time $\mathcal{O}(|S|^3 \cdot |R_I| \cdot |\Gamma/_{R_O}|)$. □

3.3 Compositionality

Compositionality is a prerequisite for scaling our analysis method to larger systems. In this section we use the example of sequential composition to show how the guarantees obtained from analyzing sub-circuits can be combined to a guarantee for the entire system. To this end, we first define an operator that connects the output signals of a machine M_1 to the input signals of a machine M_2. M_2's transition function is total, hence communication never blocks. This notion of composition models a sequential connection of two synchronous circuits with a common clock.

Definition 6. *Let $M_i = (S_i, \Sigma_i, \Gamma_i, \delta_i, s_{0,i})$, with $i \in \{1,2\}$, be automata with output and let $\Gamma_1 \subseteq \Sigma_2$. Then $M_1 \cdot M_2$ is the automaton $(S_1 \times S_2, \Sigma_1, \Gamma_2, \delta', (s_{0,1}, s_{0,2}))$, where*

$$\begin{aligned}\delta' = \{((s_1, s_2), a, b, (t_1, t_2)) \mid \exists c \in \Gamma_1.\ & (s_1, a, c, t_1) \in \delta_1 \ \wedge \\ & (s_2, c, b, t_2) \in \delta_2\}\,.\end{aligned}$$

If the observational equivalence relation on the input alphabet of M_2 is coarser than the one on the output alphabet of M_1, then we can safely compose the two machines, as the following theorem shows.

Theorem 4. *Let $M_i = (S_i, \Sigma_i, \Gamma_i, \delta_i, s_{0,i})$, with $i \in \{1,2\}$, be automata with output and let $R_I \subseteq \Sigma_1 \times \Sigma_1$, $R_O \subseteq \Gamma_1 \times \Gamma_1$, $Q_I \subseteq \Sigma_2 \times \Sigma_2$, and $Q_O \subseteq \Gamma_2 \times \Gamma_2$ be equivalence relations. Let $s_1, s_2 \in S_1$ and $t_1, t_2 \in S_2$. If $\Gamma_1 \subseteq \Sigma_2$, $R_O \subseteq Q_I$, $s_1 \simeq^{R_I}_{R_O} s_2$, and $t_1 \simeq^{Q_I}_{Q_O} t_2$, then*

$$(s_1, t_1) \simeq^{R_I}_{Q_O} (s_2, t_2) \ \text{in}\ M_1 \cdot M_2\,.$$

Proof. Since $s_1 \simeq_{R_O}^{R_I} s_2$ and $t_1 \simeq_{Q_O}^{Q_I} t_2$, there are relations $\mathcal{R}_1 \subseteq S_1 \times S_1$ and $\mathcal{R}_2 \subseteq S_2 \times S_2$ that satisfy Property (1) of Definition 2, where $(s_1, s_2) \in \mathcal{R}_1$ and $(t_1, t_2) \in \mathcal{R}_2$. It suffices to show that $\mathcal{R}_{1,2} := \{((s,t),(s',t') \mid |(s,s') \in \mathcal{R}_1 \wedge (t,t') \in \mathcal{R}_2\}$ also fulfills Property (1). To this end, let $((s,t),(s',t')) \in \mathcal{R}_{1,2}$ and let $(a,b) \in R_I$. Choose $((s,t),a,c,(p,q)) \in \delta'$, where δ' is the transition function of $M_1 \cdot M_2$. From the definition of δ', there is an $e \in \Gamma_1$ such that $(s,a,e,p) \in \delta_1$ and $(t,e,c,q) \in \delta_2$. As $\mathcal{R}_1$ satisfies (1) of Definition 2, there is a $(s',b,d,p') \in \delta_1$, with $(p,p') \in \mathcal{R}_1$ and $(e,d) \in R_O$. As $R_O \subseteq Q_I$ and $(t,t') \in \mathcal{R}_2$, there is a $(t',d,c',q') \in \delta_2$ with $(c,c') \in Q_O$ and $(q,q') \in \mathcal{R}_2$. From the definition of δ', we have $((s',t'),b,c',(p',q')) \in \delta'$ with $(c,c') \in Q_O$ and $((p,q),(p',q')) \in \mathcal{R}_{1,2}$, as required. □

Example 9. Suppose $M_i = (S_i, \Sigma_i, \Gamma_i, \delta_i, s_{0,i})$, for $i \in \{1,2\}$, are automata with output, $\Sigma_1 = \Sigma_H \times \Sigma_L$, and $\Gamma_1 = \Sigma_2 = \{0,1\}^n$. If the output of M_1 is not distinguishable with respect to the Hamming weight, i.e. $s_{0,1} \simeq_{\Psi}^{\mathrm{All}_H \times \mathrm{Id}_L} s_{0,1}$, (where $\Psi = \{(a,b) \in \{0,1\}^n \mid \|a\| = \|b\|\}$) and if M_2 does not leak anything other than possibly the Hamming weight, i.e. $s_{0,2} \simeq_{\mathrm{Id}_{\Gamma_2}}^{\Psi} s_{0,2}$, then the composition $M_1 \cdot M_2$ does not leak any information, i.e. the initial state relates to itself under $\simeq_{\mathrm{Id}_L}^{\mathrm{All}_H \times \mathrm{Id}_{\Gamma_2}}$. ◇

Analogous results hold for the parallel composition of two circuits.

4 Experimental Results

Below we report on two case studies: a simple circuit for bit-serial multiplication of nonnegative integers and a circuit for exponentiation in the field $\mathbb{F}_{2^k}$. Exponentiation over $\mathbb{F}_{2^k}$ is relevant, for example, in the generalized ElGamal encryption scheme, where decryption consists of one exponentiation and one multiplication step [16]. We implemented and tested both circuits in the hardware description language Gezel. Instead of implementing a search procedure by hand, we used the symbolic model checker Smv to automate the search on the product automaton from Definition 4.[1] Note that we translated the Gezel implementations to the input language of Smv by hand. However, the semantic gap between both languages is so small that an automated translation would be straightforward.

4.1 The Circuits

Bit-serial multiplication. For multiplying two natural numbers m and n bitwise, consider the representation $n = \Sigma_{i=0}^{k-1} n_i 2^i$, where n_i denotes the ith bit of n. The product $m \cdot \Sigma_{i=0}^{k-1} n_i 2^i$ can be expanded to

$$(\ldots((n_{k-1} \cdot m) \cdot 2 + n_{k-2} \cdot m) \cdot 2 + \ldots) \cdot 2 + n_0 \cdot m\,,$$

[1] The Gezel and Smv-code is given in the accompanying technical report [?].

which can easily be turned into an algorithm: starting with $p = 0$, one iterates over all the bits of n, beginning with the most significant bit. If $n_i = 1$, one updates p by adding m and then doubling p's value. Alternatively, if $n_i = 0$, one updates p by just doubling its value. At the end of the loop, $p = m \cdot n$.

We implemented two versions of this algorithm. In the first version, the doubling and adding operations each take one clock cycle. Hence, the running time reflects the number of 1-bits in n. In the second version, we introduce a dummy step whenever no addition takes place. In this way, the running time is independent of the operands. In our SMV implementations, the input signals are called `hi_in` and `lo_in` and they are initialized during the first clock cycle with the values of n and m, respectively. Input values of subsequent cycles are ignored. We use two output signals: one for the result `p` and a flag `done`, which signals termination.

Exponentiation in a finite field. We analyzed a hardware implementation of the finite field exponentiation algorithm from [6]. Basically, it consists of the following three building blocks:

1. To compute the *exponentiation of a field element* x with exponent $a = \Sigma_{i=0}^{n-1} a_i 2^i$, one iterates over all bits of the exponent

$$x^a = (\dots(((x^{a_{n-1}})^2 \cdot x^{a_{n-2}})^2 \cdot x^{a_{n-3}})^2 \cdot \dots)^2 \cdot x^{a_0} \,. \tag{3}$$

   In finite fields, every element x is represented by the coefficients of a polynomial, and thus each square and each multiplication operation in Equation 3 is again implemented by a loop.
2. *Multiplication of polynomials* q and $x = \Sigma_{j=0}^{r-1} x_j T^j$ is computed using the expansion $(\dots((x_{r-1} \cdot q) \cdot T + x_{r-2} \cdot q) \cdot T + \dots) + x_0 \cdot q$ in a loop similar to the one for bit-serial multiplication.
3. At the bit level, *multiplication by* T of a polynomial represented by coefficients $s = (s_{r-1}, \dots, s_0)$ can be implemented as follows. If $s_{r-1} = 0$, left-shift s by one. If $s_{r-1} = 1$, left-shift s by one and XOR the result with the coefficients of the field polynomial.

In our SMV-implementation of this exponentiation algorithm, the input signals `hi_in` and `lo_in` are initialized during the first clock cycle with a and x, respectively. Input values of subsequent cycles are ignored. We use two output signals, `p` and `done`, to represent the result x^a and termination, respectively.

4.2 Security Properties

We analyzed the multiplication and the exponentiation circuits from Section 4.1 with respect to two different security properties.

Property 1. We specify that a circuit's running time is independent of the confidential part of the input, the input signal `hi_in`. Recall that, in the case of serial multiplication, `hi_in` and `lo_in` are initialized with the operands n and m, respectively. In the case of exponentiation, `hi_in` and `lo_in` are initialized with the exponent a and the basis x, respectively. For verifying that the execution time is independent of the high input, we are only interested in when the computation terminates, that is, when the flag `done` is set, and we ignore all other output. This is specified by the relation $\simeq^{\mathrm{All}_{\Sigma_H}\times\mathrm{Id}_{\Sigma_L}}_{\mathrm{All}_{\Gamma_H}\times\mathrm{Id}_{\Gamma_L}}$. Here, $\Sigma_H = \{0,1\}^k$ denotes the range of `hi_in`, and $\Sigma_L = \{0,1\}^l$ denotes the range of `lo_in`. The `done`-flag ranges over $\Gamma_L = \{0,1\}$, and Γ_H stands for all output that is not considered.

Figure 1 demonstrates how the product construction of Definition 4 can be encoded in a few lines of SMV-code. The system to be analyzed is a module that we call `circuit`, which we instantiate twice in line 4. Both instances, `sys1` and `sys2`, are provided with the same low input (as specified by Id_{Σ_L}), and are provided with all possible combinations of high inputs (as specified by All_{Σ_H}). This is reflected in lines 6 and 7. In fact, all such input combinations are considered, as no assignments are made to the variables `lo`, `hi1`, and `hi2`.

Reachability of a falsifying state of the product automaton corresponds to a violation of the CTL-formula `!EF(!sys1.done=sys2.done)` in line 9. If we reach a state in which one instance's `done` flag is set before the other instance terminates, then we have found a falsifying state of the product automaton. In this case, SMV computes a counterexample, namely, two $\mathrm{All}_{\Sigma_H} \times \mathrm{Id}_{\Sigma_L}$-equivalent input sequences that lead to a distinguishable output.

```
1   MODULE main
2   VAR
3    lo,hi1,hi2 : array (SIZE-1)..0 of boolean;
4    sys1 : circuit; sys2 : circuit;
5   ASSIGN
6    sys1.lo_in:=lo; sys1.hi_in:=hi1;
7    sys2.lo_in:=lo; sys2.hi_in:=hi2;
8
9  SPEC !EF(!sys1.done=sys2.done)
```

Fig. 1. Product construction in SMV

Property 2. While it is easy to see that the running times of the multiplication and exponentiation algorithms depend on the input to the `hi_in`-signal, it is less clear what these dependencies are. We now specify and check a second property that formalizes that the running time only depends on the Hamming weight of the `hi_in` input (see also Example 4). That is, if the system is provided with two input sequences that are indistinguishable with respect to the Hamming weight of corresponding inputs to `hi_in`, then we require that the system has equivalent timing behavior. This is specified by the relation $\simeq^{\Psi\times Id_{\Sigma_L}}_{\mathrm{All}_{\Sigma_H}\times Id_{\Gamma_L}}$, where $\Psi = \{(a,b) \in \Sigma_H \times \Sigma_H \mid \|a\| = \|b\|\}$. Here again, $\Sigma_H = \{0,1\}^k$ denotes the range of hi_in, and $\Sigma_L = \{0,1\}^l$ denotes the range of lo_in. The done-flag ranges over $\Gamma_L = \{0,1\}$, and Γ_H stands for all output that is not considered.

The SMV-implementation of Property 2 follows along the same lines as the implementation of Property 1. The only difference is that we modify the input to hi_in of sys2 in line 7 of Figure 1 in the following way:

```
sys2.hi_in:=
case
  hi1[0]+...+hi1[SIZE-1]=hi2[0]+...+hi2[SIZE-1] : hi2;
  1 : hi1;
esac;
```

The variables hi1 and hi2 both take all possible values in their range. Only when their Hamming weight coincides is sys2 fed with hi2. Otherwise its input is hi1. In this way, we ensure that the inputs to both instances of circuit always have the same Hamming weight and that all such combinations are considered.

4.3 Results

Security Analysis. The table in Figure 2 presents the results of our analysis. The first column corresponds to the serial multiplication algorithm where dummy steps are inserted to avoid timing leaks. The second column corresponds to the multiplication algorithm without dummy steps, and the third column contains the results for the finite-field exponentiation algorithm. The rows correspond to Properties 1 and 2 described in Section 4.2. An entry ✓ denotes that the model is secure with respect to the corresponding notion of security, whereas $\times$ denotes that this is not the case.

The first column reflects what was intended by inserting dummy computation steps into the design: the circuit's running time is independent of the input to the signal hi_in. In particular, as Example 6 shows, arbitrary input sequences do not lead to distinguishable behavior.

The second column shows that the running time of the multiplication algorithm without dummy computation depends on the input to the signal hi_in. However, if the implementation is only run on inputs with equal Hamming weight, then we cannot observe any differences between the running times. Example 8 shows that, if the high environment provides input only during the first clock cycle, no more than the Hamming weight of the input can be leaked. Note that this actually holds in an arbitrary environment, as the circuit ignores input during all but the first clock ticks.

The third column shows that the running time of the exponentiation algorithm depends on the input to the signal hi_in, which corresponds to the exponent. The result of the analysis with respect to inputs of equal Hamming weights is surprising. When only considering loop 1 (see Section 4.1), one might expect the same result as for serial multiplication. However, the second row states that this is not the case: even when provided with input of the same Hamming weight, the system shows differences in its running times. This means that information other than the Hamming weight can be leaked. We have not yet undertaken a precise characterization of this leak. The counterexample computed by SMV suggests that this might be nontrivial: the first difference between the sequences of states reached in both instances of `circuit` occurs after 20 steps, and distinguishable output is not produced until 36 steps.

	Multiplication (padded)	Multiplication	Exponentiation
$\simeq^{\mathrm{All}_H \times Id_L}_{\mathrm{All}_H \times Id_L}$	✓	×	×
$\simeq^{\Psi \times Id_L}_{\mathrm{All}_H \times Id_L}$	✓	✓	×

Fig. 2. Results of Analysis

Performance. We performed our experiments on a 2.4 GHz machine with 3 gigabytes of RAM. In the case of serial multiplication, we were able to analyze designs up to 10 bits per operand within one minute. In the case of exponentiation, we were able to analyze designs with up to 3 bits per operand within 2 minutes.[2] For larger bit-widths the running times increased notably. Note that these numbers were obtained by using SMV "out of the box", that is, without applying one of the many existing optimization techniques. We expect a significant performance gain by tailoring the search procedure to our specific problem instance, for example by adopting abstraction techniques for handling bit-vectors.

5 Related Work

Both timing-aware security definitions and decidability results exist in process algebraic settings, e.g., [7, 14], to name just a few. Their standard model of communication is event-based and differs significantly from our time-synchronous model. Likewise, security definitions for process algebras usually restrict the detection of secret events by low-level observers, while R_I/R_O-security aims at protecting a stream of confidential data. While formal connection between language-based and process algebraic approaches can be made [8], we focus on methods from language-based security as they are more directly related to our work.

Several authors use bisimulations to express timing-sensitive notions of secure information flow, e.g., [28, 1, 21]. The use of arbitrary equivalence relations for capturing partial information flow has been proposed

[2] This corresponds to a state-space size of approximately 2^{52} for the product automaton.

in a timing-insensitive context [2, 9]. In this context, the notion of *independent composition* [2] is related to our product construction $\times^{R_I}_{R_O}$. R_I/R_O-security marries the timing-awareness of the bisimulation-based approaches with the accuracy of the parameterized approaches. In [10], a parameterized and timing-aware definition of secure information flow is given. However, it does not allow for input sequences of arbitrary length and it is unclear whether it can be efficiently checked. The idea of quantifying information by the number of distinguishable behaviors has been proposed by Lowe [14] as an over-approximation for Shannon's information-theoretic measure.

Programming language-based approaches to counter timing leaks usually assume infinite-state transition systems, which leads to undecidable analysis problems. One way to approximate undecidable security conditions is to use syntax-driven techniques, such as security type systems. Several security type systems for dealing with timing-sensitive notions of secure information flow for programming languages have been proposed [1, 28, 21, 13, 3]. We exploit the fact that the state spaces in our setting are finite to develop a method for efficiently deciding system security.

Tolstrup et al. [27] present an information flow analysis method for the hardware description language VHDL that does not consider timing issues. A recent follow-up paper [26] also incorporates timing and provides a type-system to approximate a semantic definition of security. Analyzing hardware on the level of VHDL has the advantage of being very concrete, but it also means that one has to deal with artifacts such as processes and δ-time. Our automata-based model is more abstract and it allows for a clean separation of program semantics and security definitions. Moreover, our approach has the advantage of an efficient decision procedure.

6 Conclusions and Outlook

The results presented in this paper are both theoretical and practical. On the theoretical side, we have developed a parametric notion of security for an automaton model for synchronous systems and have given algorithms and complexity bounds for its decision problem. In the deterministic case, we have derived quantitative bounds for the confidential information that a system may reveal to an attacker. On the practical side, we have shown that our definitions encompass a number of interesting security properties and applied our techniques to verify (or detect timing leaks in) nontrivial hardware implementations of cryptographic algorithms.

While the notion of R_I/R_O-security proposed appears to be a general and useful parametric notion of information flow, counting distinguishable behaviors provides only an approximate measure of the quantity of information that a system may leak. It should not be difficult though to incorporate probability distributions on the inputs to give more concrete, information-theoretic bounds, e.g. along the lines of [5].

Another area for future work concerns algorithms and abstractions that can help us manage both larger systems and those with infinite state-spaces. It would also be interesting to use our security notions as a starting point for techniques to automatically correct insecure systems.

References

1. J. Agat. Transforming out Timing Leaks. In *Proc. POPL '00*, pages 40–53.
2. G. Barthe, P. D'Argenio, and T. Rezk. Secure Information Flow by Self-Composition. In *Proc. CSFW '04*, pages 100–114.
3. G. Barthe, T. Rezk, and M. Warnier. Preventing Timing Leaks Through Transactional Branching Instructions. In *Proc. QAPL '05*.
4. D. Boneh and D. Brumley. Remote Timing Attacks are Practical. In *Proc. USENIX Security Symposium '03*, 2003.
5. D. Clark, S. Hunt, and P. Malacaria. Quantitative Information Flow, Relations and Polymorphic Types. *J. Log. Comput.*, 18(2):181–199, 2005.
6. M. Davio, J. P. Deschamps, and A. Thayse. *Digital Systems with Algorithm Implementation*. John Wiley & Sons, Inc., 1983.
7. R. Focardi, R. Gorrieri, and F. Martinelli. Information Flow Analysis in a Discrete-Time Process Algebra. In *Proc. CSFW '00*, pages 170–184.
8. R. Focardi, S. Rossi, and A. Sabelfeld. Bridging Language-Based and Process Calculi Security. In *Proc. FoSSaCS '05*, LNCS 3829, pages 299–315.
9. R. Giacobazzi and I. Mastroeni. Abstract non-interference: Parameterizing non-interference by abstract interpretation. In *Proc. POPL'04*, pages 186–197.
10. R. Giacobazzi and I. Mastroeni. Timed Abstract Non-Interference. In *Proc. FORMATS'05*, LNCS 3829, pages 289–303.
11. P. Kanellakis and S. Smolka. CCS Expressions, Finite State Processes, and Three Problems of Equivalence. *Information and Computation*, 86:43–68, 1990.
12. P. C. Kocher. Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. In *Proc. CRYPTO '96*, LNCS 1109, pages 104–113.
13. B. Köpf and H. Mantel. Eliminating implicit information leaks by transformational typing and unification. In *In Proc. FAST'05*, LNCS 3866, pages 42–62, 2006.
14. G. Lowe. Quantifying Information Flow. In *Proc. CSFW '02*, pages 18–31.
15. J. D. McLean. A General Theory of Composition for Trace Sets Closed under Selective Interleaving Functions. In *Proc. IEEE Symp. on Security and Privacy '94*, pages 79–93.
16. A. Menezes, P. van Oorschot, and S. Vanstone. *Handbook of Applied Cryptography.* CRC Press, 1996.
17. D. A. Osvik, A. Shamir, and E. Tromer. Cache Attacks and Countermeasures: the Case of AES. In *Proc. CT-RSA '06*, LNCS 3860, pages 1–20.
18. R. Paige and R. E. Tarjan. Three Partition Refinement Algorithms. *SIAM J. Comput.*, 6(16):973–989, 1987.
19. P. Puschner and A. Burns. A Review of Worst-Case Execution-Time Analysis. *Real-Time Systems*, 18(2/3):115–128, 2000.
20. A. Sabelfeld and A. C. Myers. Language-based Information-Flow Security. *J. Selected Areas in Communication*, 21(1):5–19, 2003.
21. A. Sabelfeld and D. Sands. Probabilistic Noninterference for Multi-threaded Programs. In *Proc. CSFW '00*, pages 200–215.
22. A. Sabelfeld and D. Sands. A PER Model of Secure Information Flow in Sequential Programs. *Higher-Order and Symbolic Computation*, 14(1):59–91, 2001.
23. P. Schaumont and I. Verbauwhede. Domain-Specific Codesign for Embedded Security. *IEEE Computer*, 36(4):68–74, 2003.
24. P. Schaumont and I. Verbauwhede. The Descriptive Power of GEZEL. Technical report, 2005.
25. G. Smith and D. Volpano. Secure Information Flow in a Multi-Threaded Imperative Language. In *Proc. POPL '98*, pages 355–364.
26. T. Tolstrup and F. Nielson. Analyzing for Absence of Timing Leaks in VHDL. In *Proc. WITS '06 (to appear)*.
27. T. Tolstrup, F. Nielson, and H. Nielson. Information Flow Analysis for VHDL. In *Proc. PaCT '05*, LNCS 3606, pages 79–98.
28. D. Volpano and G. Smith. Eliminating Covert Flows with Minimum Typings. In *Proc. CSFW '97*, pages 156–168.

A Finite Field Exponentiation

In this section we present two implementations of an algorithm for exponentiation in the finite field $\mathbb{F}_{2^k}$ as in [6]. The first implementation is given in the synchronous hardware description language GEZEL [24]. The second implementation is the translation into the input language of the symbolic model checker SMV. State names are chosen in according to the textbook description.

A.1 GEZEL-Code

```
dp circuit(in  x_in : ns(4) ; in a_in: ns(4) ;
           out done : ns(1) ; out p :ns(4)) {

 reg x : ns(4); // field element
 reg a : ns(4);         // exponent
 reg m : ns(4);         // field polynomial
 reg r,q,s:ns(4);       // additional registers
 reg i,j : ns(3);       // counters for 0..4 (a,m,r,q,s)

sfg init {
    x=x_in;
    a=a_in;
m=0b0111;       // T^4+T^2+T+1: Field Polynomial
}

sfg sig0  {r=1;}
sfg sig1  {q=0;}
sfg sig2  {i=0;}
sfg sig3  {j=0;}
sfg sig4  {i=i+1;}
sfg sig5  {j=j+1;}
sfg sig6  {s=r;}
sfg sig7  {r=q;}
sfg sig8  {s=x;}
sfg sig9  {q=q<<1;}
sfg sig10 {s=s<<1;}
sfg sig11 {a=a<<1;}
sfg sig12 {q=q^m;}
sfg sig13 {q=q^r;}
sfg cont  {done=0;
     p=0;}
sfg term  {done=1;
   p=r;
   $display(" p=", r,  " done");
   $finish;}
}

fsm ctl_circ(circuit) {

 initial s1;
 state s2,s3,s4,s5,s6,s8,s9,s10,s11,s12,end;

 @s1 (init,sig0,sig2,cont) -> s2;

 @s2 if (i==4)
```

```
then (term) -> end;
else (sig1,sig3,sig6,cont) -> s3;

 @s3 if (j==4)
then (sig7,cont) -> s8;
else (cont) -> s4;

 @s4 if (q[3])
then (sig9,cont) -> s5;
else (sig9,cont) -> s6;

 @s5 (sig12,cont) -> s6;

 @s6 if (s[3])
then (sig5,sig10,sig13,cont) -> s3;
else (sig5,sig10,cont) -> s3;

 @s8 if (a[3])
then (sig1,sig3,sig8,cont) -> s9;
else (sig4,sig11,cont) -> s2;

 @s9 if (j==4)
then (sig4,sig7,sig11,cont) -> s2;
else (cont)-> s10;

 @s10 if (q[3])
then (sig9,cont) -> s11;
else (sig9,cont) -> s12;

 @s11 (sig12,cont) -> s12;

 @s12 if (s[3])
then (sig5,sig10,sig13,cont) -> s9;
  else (sig5,sig10,cont) -> s9;

 @end (term) -> end;
}

dp test_circ(out bs: ns(4); out es : ns(4)) {
  sfg run {
    bs = 0b1111;
    es = 2;
  }
}
hardwired ctl_test(test_circ) {run; }

dp test_test{
 sig b,r : ns(4);
 sig e : ns(4);
 sig d : ns(1);

 use circuit (b,e,d,r);
 use test_circ (b,e);
}

system S {
  test_test;
}
```

A.2 Smv-Code

```
#define EWIDTH 4
#define FWIDTH 4

MODULE circuit

VAR

 x, lo_in : array (FWIDTH-1)..0 of boolean;
 a, hi_in : array (EWIDTH-1)..0 of boolean;
 m,r,q,s : array (FWIDTH-1)..0 of boolean;
 i : array 2..0 of boolean;
 j : array 2..0 of boolean;
 done : boolean;
 state : {s1,s2,s3,s4,s5,s6,s8,s9,s10,s11,s12,end};

ASSIGN

 init(state):=s1;
 next(state):=
   case
    state=s1 : s2;
    state=s2 & i=EWIDTH : end;
    state=s2 : s3;
    state=s3 & j=FWIDTH : s8;
    state=s3 : s4;
    state=s4 & q[FWIDTH-1] :s5;
    state=s4 : s6;
    state=s5 : s6;
    state=s6 : s3;
    state=s8 & a[EWIDTH-1] : s9;
    state=s8 : s2;
    state=s9 & j=FWIDTH : s2;
    state=s9 : s10;
    state=s10 & q[FWIDTH-1] : s11;
    state=s10 : s12;
    state=s11 : s12;
    state=s12 : s9;
   esac;

 next(x):=
  case
    state=s1 : lo_in;
    1:x;
  esac;

 next(a):=
  case
    state=s1 : hi_in;
    state=s8 & (!a[EWIDTH-1]) : a<<1;
    state=s9 & j=FWIDTH : a<<1;
    1:a;
  esac;

 next(m):=
  case
   state=s1 : 7;   // Coefficients of FP T^4+T^2+T+1
```

```
  1:m;            // without T^4-term (0 1 1 1)
 esac;

next(r):=
 case
  state=s1 : 1;
  state=s3 & j=FWIDTH : q;
  state=s9 & j=FWIDTH : q;
  1:r;
 esac;

next(q):=
 case
  state=s2 & !(i=EWIDTH) : 0;
  state=s8 & a[EWIDTH-1] : 0;
  state=s4 : q<<1;
  state=s10 : q<<1;
  state=s5 : q^m;
  state=s11 : q^m;
  state=s6 & s[FWIDTH-1] : q^r;
  state=s12 & s[FWIDTH-1] : q^r;
  1:q;
 esac;

next(s):=
 case
  state=s2 & (!i=EWIDTH) : r;
  state=s8 & a[EWIDTH-1] : x;
  state=s6 : s<<1;
  state=s12 : s<<1;
  1:s;
 esac;

next(i):=
 case
   state=s1 : 0;
   state=s8 & (!a[EWIDTH-1]) : i+1;
   state=s9 & j=FWIDTH : i+1;
   1:i;
 esac;

next(j):=
 case
   state=s2 & !(i=EWIDTH) : 0;
   state=s8 & a[EWIDTH-1] : 0;
   state=s6 : j+1;
   state=s12 : j+1;
   1:j;
 esac;

init(done):=0;
next(done):=
 case
  state=s2 & i=EWIDTH : 1;
  1:done;
 esac;
```