On Aggregation of Information in Timing Attacks

Itsaka Rakotonirina
INRIA Nancy Grand-Est, LORIA
itsaka.rakotonirina@inria.fr

Boris Köpf
Microsoft Research
boris.koepf@microsoft.com

***Abstract*—A key question for characterising a system's vulnerability against timing attacks is whether or not it allows an adversary to *aggregate* information about a secret over multiple timing measurements. Existing approaches for reasoning about this aggregate information rely on strong assumptions about the capabilities of the adversary in terms of measurement and computation, which is why they fall short in modelling, explaining, or synthesising real-world attacks against cryptosystems such as RSA or AES.**

In this paper we present a novel model for reasoning about information aggregation in timing attacks. The model is based on a novel abstraction of timing measurements that better captures the capabilities of real-world adversaries, and a notion of compositionality of programs that explains attacks by divide-and-conquer. Our model thus lifts important limiting assumptions made in prior work and enables us to give the first uniform explanation of high-profile timing attacks in the language of information-flow analysis.

***Index Terms*—side-channels, timing attacks, information flow, compositionality**

I. Introduction

Side-channel attacks break the security of systems by exploiting physical characteristics such as power consumption [1], electromagnetic emissions [2], [3], or execution time [4], [5]. Among those characteristics, execution time poses a particular threat to open, distributed systems such as the Internet because it can be measured and exploited remotely [6].

Security against timing attacks is often expressed in terms of *noninterference*, which is the requirement that the execution time of a program must not depend on secret inputs. However, while noninterferent timing behaviour implies resilience against timing attacks, the converse is not always true: a violation of noninterference needs not give rise to an attack that effectively recovers the secret from timing measurements. For secrets with few alternatives (such as a single bit), recovery may require only a single timing measurement. However, for secrets with many alternatives (such as cryptographic keys), recovery usually requires aggregating timing information across multiple executions. Hence, the key question for characterising a program's vulnerability against timing attacks is whether or not it allows an adversary to perform such an aggregation.

A number of approaches have been proposed for expressing and computing this aggregate information for real programs. Some focus on deriving sound bounds on what an attacker can in principle achieve [7]–[10], others focus on generating actual attacks [11]–[13]. Despite their technical differences, those approaches rely on two common assumptions, namely:

1) that the adversary has access to *accurate models* of the target system's timing behaviour. These models can be deterministic or probabilistic and be obtained from specifications or profiling.
2) that the adversary has *unbounded computational resources*. In particular, they rely on computations of preimages or the enumeration of all paths of a program, both of which are computationally intractable operations for cryptographic code with realistic key lengths.

Assertions about the aggregate information based on these assumptions are sound, in the sense that they over-estimate what a real-world adversary can achieve. However, they fall short in modelling, explaining, or synthesising real-world attacks against vulnerable implementations of cryptosystems such as RSA and AES [4], [5], [14]–[16], which recover aggregate information from implementations without accurate timing models or expensive computations.

Approach In this paper we propose a novel approach for reasoning about information aggregation in timing attacks that lifts the assumptions described above. Our approach is based on three technical contributions:

1) a novel discrete abstraction of timing measurements that captures an adversary with only an approximate model of the target system's timing behaviour;
2) a novel structural condition on the timing behaviour of programs that facilitates information aggregation by divide-and-conquer; and
3) a rigorous generic analysis of the cost of aggregation in terms of measurement and computation.

Below we explain each of those contributions in detail.

A discrete abstraction of timings Our abstraction of timing measurements relies on the assumption that the adversary can only learn information about a long-term secret by *comparing* the program's execution time on different public inputs, rather than by measuring *absolute* execution time. As a consequence, the resulting notions of leakage are invariant to unknown constant shifts of the execution time. Focusing on comparisons captures how timing measurements are used in documented attacks [4], [5], [14] and gives a more realistic account of the information that can be extracted without a complete model of execution time.

A structural condition for divide-and-conquer We identify conditions on the structure of programs that enable an adversary to aggregate secret information by divide-and-conquer. Essentially, the conditions require that the program is sequentially composed of a number of sub-programs that are *independent*, in the sense that there are adversary inputs that trigger secret-dependent execution time of each specific sub-program while keeping constant the execution time of the remaining code.

We formalise two versions of independence:

- A *possibilistic* one that explains the core idea behind information aggregation in timing attacks using discrete mathematics and forms a promising basis for automatic crafting of adversary inputs, i.e. generation of attacks.
- A *probabilistic* one that is inspired by real-world attacks against RSA [5] and allows to replace (potentially expensive) crafting of attacker inputs by random sampling.

Cost of aggregation by divide-and-conquer Based on these independence conditions, we give generic descriptions of timing attacks, together with a rigorous analysis of their cost in terms of the number of timing measurements and computation steps.

- In the deterministic case we show how the cost of an attack can go down from exponential to linear in the number of components.
- In the probabilistic case we show how the cost can go down to $\mathcal{O}(n \ln \frac{n}{\varepsilon})$ random samples, where n is the number of components, to recover a secret with probability $1 - \varepsilon$. We provide a formal proof of this bound, and we validate it by empirical simulations.

We conclude with case studies showing how our model encompasses high-profile attacks on RSA and AES [4], [5], [14], for the first time exhibiting their common core.

In summary, this paper contributes a model for reasoning about efficient information aggregation in timing attacks, based on a novel abstraction of timing measurements and a novel notion of compositionality. Our model lifts important assumptions made in previous work and enables us to give the first uniform explanation of high-profile timing attacks in the language of information-flow analysis.

II. Modelling Timing Leaks

In this section we introduce a mathematical model of timing leaks. It is based on an adversary that can repeatedly run a program on different inputs and observe the corresponding execution time. We first recall a characterisation of leaks to an adversary that has full knowledge of the implementation and access to perfect measurements of absolute execution time [7], before we refine this model to one where the adversary can only learn secret information by observing *differences* in time between different executions.

A. Programs and Timing Models

Let K be a finite set of *secrets*, M a finite set of *messages*, and O a set of *observations*. We characterise program *implementations* as pairs of functions (f, t) consisting of a *functionality* $f\colon K \times M \to M$ and a *timing model* $t\colon K \times M \to O$. The functionality f represents the input-output behaviour of the program. The timing model t captures a deterministic notion of execution time that encompasses real time ($O \subseteq \mathbb{R}^+$), numbers of clock ticks, or executed instructions ($O \subseteq \mathbb{N}^+$).

We also often refer to *oracles* $t^*\colon M \to O$ which are timing models where the secret parameter has been fixed. The aim of a timing attacker is to infer the value of the secret parameter from queries to the oracle.

Example 1. Consider the code, where $K = M = \{0,1\}^n$ are sets of bitvectors and where x_i denotes the i^{th} bit of vector x:

```
for i = 0 to n − 1 do
  if k_i ≠ m_i then g()
done
```

Under the assumption that each individual instruction (including the call to $g()$) consumes a constant amount of time, the program's execution time is proportional to the number of bits in which k and m differ, i.e., their *Hamming distance*. That is, for some constant $c, \alpha \in O$,

$$t(k, m) = c + \alpha \sum_{i=0}^{n-1} k_i \oplus m_i \,.$$

Example 2. Consider the same program but replace the guard $k_i \neq m_i$ by the guard $k_i = 1$. The resulting timing model t is independent of m and proportional to the number of 1-bits in k, i.e. the *Hamming weight* of k.

Examples 1 and 2 represent patterns of timing behaviour that are common in algorithms for bit-serial multiplication and exponentiation, and we will use them as running examples throughout the paper. We leave functionalities implicit for now; they will be needed for tracking state when we compose timing models in Section III.

B. Leaks Under Absolute Measurements

We characterise the secret information that is leaked by the timing behaviour of a program. We begin by modeling a strong adversary that has full knowledge of the implementation and access to perfect measurements of absolute execution time.

Technically, we assume that $k \in K$ is a long-term secret that stays constant over multiple invocations of (f, t), and that the adversary repeatedly runs (f, t) with different messages $m \in M$ to narrow down the potential values of k from timing measurements. For a single query to oracle (f, t) on m, the adversary sees $o = t(k, m)$; this observation allows it to conclude that

$$k \in \{k' \in K \mid t(k', m) = o\} \,.$$

This set describes all secrets that are *coherent* with the observation o under m: it is is the preimage of o under the function $t(\cdot, m)$. In particular, two secrets k and k' are *indistinguishable under* m if they produce the same observation when queried with m, i.e.

$$t(k, m) = t(k', m) \ .$$

For every m, *indistinguishability under* m is an equivalence relation $\simeq_m$ on K.

The equivalence classes of $\simeq_m$ therefore form a partition on K which characterises the information an adversary can deduce about a secret k upon invoking the oracle $t(k, \cdot)$ on message m. The coarsest partition $\top = \{K\}$ captures that execution time reveals no secret information, whereas the finest partition $\bot = \{\{k\} \mid k \in K\}$ captures that the secret is entirely determined by the execution time. The notations $\top$ and $\bot$ may indistinctly refer to their relational counterparts, i.e. $\top$ is the universal equivalence relation and $\bot$ is the equality. Intermediate partitions capture partial knowledge about the secret, where finer means more knowledge.

Modelling knowledge in terms of partitions (resp. equivalence relations) is common in information-flow analysis [17] and has close connections to notions of declassification [18].

Example 3. Consider the Hamming weight from Example 2, $t(k, m) = \sum_{i=0}^{n-1} k_i$, with $n = 2$. For any message m, the information that t leaks about the secret input is characterised by the partition

$$\{\{00\}, \{01, 10\}, \{11\}\} \, ,$$

where the equivalence classes correspond to secrets of Hamming weights 0, 1, and 2, respectively.

The aggregate information that an adversary can extract by exhaustively running the program on all messages $m \in M$ can be described as the intersection of the corresponding equivalence relations, which we denote as

$$\mathsf{Leak}(t) = \bigcap_{m \in M} \simeq_m \ .$$

If we consider equivalence relations on a subset $K' \subseteq K$ we make this explicit by writing $\mathsf{Leak}(t, K')$.

With this, two keys k, k' are in the same equivalence class of $\mathsf{Leak}(t)$ *iff* they produce the same oracle. That is,

$$[k] = [k'] \text{ in } \mathsf{Leak}(t) \quad \textit{iff} \quad t(k, \cdot) = t(k', \cdot) \, . \tag{1}$$

Here, $[k]$ refers to the class of $\mathsf{Leak}(t)$ enclosing k.

Example 4. Consider $t(k, m) = \sum_{i=0}^{n-1} k_i \oplus m_i$ the Hamming distance from Example 1, with $n = 2$. For $m_0 = 00$ and $m_1 = 01$, we obtain the equivalence relations $\simeq_{m_0}$ and $\simeq_{m_1}$ defined by the equivalence classes:

$$\simeq_{m_0} : \{00\}, \{01, 10\}, \{11\} \quad \simeq_{m_1} : \{01\}, \{00, 11\}, \{10\} \, .$$

For both partitions, the blocks correspond to secrets of Hamming distance to m_i of 0, 1 and 2, respectively. Their intersection yields $\bot$, i.e. the secret is entirely determined by aggregate timing information of queries m_0 and m_1. More generally, we have $\mathsf{Leak}(t) = \bot$ for the Hamming distance model on bitvectors of any length.

C. Leaks Under Differential Measurements

The characterisation of leaks presented in Section II-B relies on the assumption that the timing measurement is a function of the secret input. In most real-world attacks, however, timing measurements capture a secret-dependent signal together with irrelevant (and often noisy) computation. We now characterize leaks to a weaker adversary that can learn secret information only by *comparing* the execution times on different messages. This notion of leakage is robust to unknown offsets in the timing measurements and hence more faithfully captures the capabilities of a real-world timing adversary.

On a technical level, we capture this adversary in terms of a new indistinguishability relation, where $k \simeq_{(m,m')} k'$ if and only if

$$t(k, m) - t(k, m') = t(k', m) - t(k', m') \, .$$

That is, two secrets are indistinguishable under $\simeq_{(m,m')}$ if they show identical *differences* in execution times when queried with m and m'.

With this, we define $\mathsf{dLeak}(t)$ characterising the information an adversary can gain by observing the difference between execution times for any pair of messages.

$$\mathsf{dLeak}(t) = \bigcap_{(m,m') \in M \times M} \simeq_{(m,m')}$$

As for $\mathsf{Leak}(t)$, we often write $\mathsf{dLeak}(t, K')$ when considering equivalence only on a subset $K' \subseteq K$.

As before, we can characterise the equivalence classes of $\mathsf{dLeak}(t)$ by a comparison of the corresponding oracles:

$$[k] = [k'] \text{ in } \mathsf{dLeak}(t) \quad \textit{iff} \quad \|t(k, \cdot) - t(k', \cdot)\| = 0 \tag{2}$$

where $\|t^*\| = \max t^* - \min t^*$ is the amplitude of the bounded function or vector t^*. That is, two keys k, k' are in the same class of $\mathsf{dLeak}(t)$ if they produce the same oracle up to constant shift.

From Equations (1) and (2), we see that the information leakage induced by observing only differences is coarser than the leakage by absolute measurements:

Proposition 1. *For all timing models t, we have the inclusion of relations* $\mathsf{Leak}(t) \subseteq \mathsf{dLeak}(t)$.

Example 5. For the Hamming distance from Example 1, we have $\mathsf{dLeak}(t) = \bot$, meaning that differential-time and absolute-time adversaries can both learn the full secret via timing. On the contrary, for the Hamming weight from Example 2 we obtain $\mathsf{dLeak}(t) = \top$. That is, differential-time adversaries do not learn any information via timing, whereas absolute-time adversaries can learn the Hamming weight of the secret.

D. Queries to Oracles

The characterisations of leaks presented so far, namely the equivalence relations $\mathsf{Leak}(t)$ and $\mathsf{dLeak}(t)$, are based on the aggregation of timing information over all possible (pairs of) messages. Now we give an account of the redundancy of the information leaked through different timing measurements. This will form the basis of the attacks we present later, consistently with the adversary's goal of minimising the number of measurements needed to recover the secret.

The basis of the characterisation are equivalence relations on messages that capture the redundancy between oracle queries. Specifically, we define $m_1 \eqsim_k m_2$ if and only if $t(k, m_1) = t(k, m_2)$. Intuitively, it is sufficient to pick one representative of each equivalence class of $\eqsim_k$ to exercise all behaviours of an oracle $t(k, \cdot)$.

Based on this family of equivalence relations, we compile messages in two different fashions:

- *Intersection.* We intersect the equivalences $\eqsim_k$ for all potential keys k to get a refined equivalence relation on M.
$$\mathsf{Query}(t) = \bigcap_{k \in K} \eqsim_k .$$
- *Samples.* We gather equivalence classes of all equivalence relations $\eqsim_k$ into a collection of (potentially-overlapping) samples of messages.
$$\mathsf{Sample}(t) = \bigcup_{k \in K} M_{/\eqsim_k}$$

If we consider only a subset of keys $K' \subseteq K$, we make this explicit by writing $\mathsf{Query}(t, K')$ and $\mathsf{Sample}(t, K')$.

The equivalence relation $\mathsf{Query}(t)$ forms a basis for a discrete approximation of t, in the sense that oracles to t are entirely determined by their images on one representative of each class of $\mathsf{Query}(t)$. In contrast, the collection of samples $\mathsf{Sample}(t)$ forms a basis for a statistical approximation of t, in the sense that oracles to t are entirely determined by their expectations on these samples. This is formalised in the differential-time model by the following proposition.

Proposition 2. *Let* $[m_1], \ldots, [m_p]$ *be the equivalence classes of* $\mathsf{Query}(t)$ *and* $\mathsf{Sample}(t) = \{M_1, \ldots, M_q\}$. *The following points are equivalent:*

(i) $[k] = [k']$ *in* $\mathsf{dLeak}(t)$

(ii) $\|o_k - o_{k'}\| = 0$ *with* $o_k = (t(k, m_1), \ldots, t(k, m_p))$

(iii) $\|\bar{o}_k - \bar{o}_{k'}\| = 0$ *with* $\bar{o}_k = (\mu_1, \ldots, \mu_q)$ *and*
$$\mu_i = \frac{1}{|M_i|} \sum_{m \in M_i} t(k, m)$$

Proof. $(i) \Rightarrow (ii)$ and $(i) \Rightarrow (iii)$ easily follow from Equation (2). $(ii) \Rightarrow (i)$ is also straightforward since m and m' equivalent for $\mathsf{Query}(t)$ means $t(\cdot, m) = t(\cdot, m')$. For $(iii) \Rightarrow (i)$ we prove in Appendix A that, up to constant shift, if two oracles $t(k, \cdot)$ and $t(k', \cdot)$ coincide in expectation on the M_is (i.e. $\|\bar{o}_k - \bar{o}_{k'}\| = 0$) but are not equal (i.e. $\|t(k, \cdot) - t(k', \cdot)\| \neq 0$), then one can construct an infinite sequence of distinct messages, yielding a contradiction with the finiteness of M. □

As a generalisation of Example 5, we can for example formalise in our model that a differential-time adversary does not learn anything by observing the timing of an implementation whose execution time cannot be influenced by choosing different public inputs:

Proposition 3. $\mathsf{Query}(t) = \top$ *entails* $\mathsf{dLeak}(t) = \top$.

In the next sections we show how both characterisations can be used as a basis for compositionality results for the differential-time model.

III. Compositionality in Timing Attacks

In this section we introduce a novel notion of decomposition of programs into sub-programs that is a sufficient condition for aggregation of timing leakage by divide-and-conquer. Our notion is inspired by informal considerations made in timing attacks on RSA [4], [5]. In this paper we give the first formalisation of composition in two flavours, possibilistic and probabilistic, together with the core property that gives rise to efficient attacks.

A. Composition of Implementations

The sequential composition of a functionality $f \colon K \times M \to M$ with a function $g \colon K \times M \to E$ of arbitrary return type E is defined by
$$(f \,;g)(k, m) = g(k, f(k, m))$$
This is a standard function composition, except that secret arguments are passed along unmodified. Then, consider implementations $(f_1, t_1), \ldots, (f_n, t_n)$. We define

- $\bar{f}_i = f_1 \,; f_2 \,; \cdots \,; f_i$, that is, the composition of functionalities up to i
- $\bar{t}_i = \bar{f}_{i-1} \,; t_i$, that is, the timing of the block i in the context of the program execution, i.e. based on input given by $\bar{f}_{i-1}$.

Definition 1. We say that an implementation (f, t) is *composed of* implementations $(f_1, t_1), \ldots, (f_n, t_n)$ when $f = \bar{f}_n$ and $t = \sum_{i=1}^{n} \bar{t}_i$.

Definition 1 requires that f is the standard composition of the functionalities f_i, and that t is the sum of their execution times. We call (f_i, t_i) a *block* of (f, t) and write $\bar{t}_{-i} = \sum_{j \neq i} \bar{t}_j$, the timing of all blocks of (f, t) except i.

Example 6. The following extension of Example 1 is the common structure of several serial-bit algorithms such as our running example (Hamming distance), but also implementations of RSA and AES (see case studies, Section VI). Here

f_i refers to a functionality and $t_i : K \times M \to \{0,1\}$ to a predicate seen as a timing model.

$s = m$
for $i = 0$ to $n-1$ do
  $s = f_i(k, s)$
  if $t_i(k, s) = 1$ then $g()$
done

This is modelled in our formalism as the sequential composition of the blocks $(f_1, t_1), \ldots, (f_n, t_n)$. The Hamming distance may be seen as an instance of this pattern with $f_i(k, s) = s$ and $t_i(k, s) = k_i \oplus s_i$.

B. Aggregation of Timing Information

Proposition 4. *If the implementation (f, t) is composed of $(f_1, t_1), \ldots, (f_n, t_n)$, then*

$$\mathsf{dLeak}(t) \supseteq \bigcap_{i=1}^{n} \mathsf{dLeak}(\bar{t}_i)$$

The proposition follows because whenever two secrets that are indistinguishable in terms of the execution time of each individual block, they are also indistinguishable in terms of the overall execution time. The converse is not necessarily true, because variations of the execution time of two individual blocks may cancel each other out. That is, secrets that are distinguishable in each block need not be distinguishable in t.

We next exhibit structural conditions on the program that guarantee that timing leaks of different blocks do not cancel out. With those conditions, we obtain equality in Proposition 4.

C. Possibilistic Independence

We define a possibilistic notion of independence of blocks. Intuitively, blocks are independent if the adversary is able to induce variations in the timing of a specific target block $\bar{t}_i$ while keeping the timing of the other blocks $\bar{t}_{-i}$ constant. This is technically formalised as follows.

Definition 2. The composition of $(f_1, t_1), \ldots, (f_n, t_n)$ is *independent* if, for all $i \in \mathbb{N}_n$, there exists M' an equivalence class of $\mathsf{Query}(\bar{t}_{-i})$ such that for all M'' equivalence class of $\mathsf{Query}(\bar{t}_i)$, $M' \cap M'' \neq \emptyset$. We call the set M' a *calibrator* for t_i in t.

Equivalently, independence implies that for each i there is a set of messages—the calibrator—that allows for exercising all behaviours of $\bar{t}_i$ while keeping its complement $\bar{t}_{-i}$ constant.

Example 7. The blocks of the Hamming distance timing model (see Example 6) are independent. To see this, observe that *any* pair of messages m, m' with $m_i \neq m'_i$ is a representative system for the two equivalence classes of $\mathsf{Query}(\bar{t}_i)$. For ensuring that m and m' are equivalent for $\mathsf{Query}(\bar{t}_{-i})$ we additionally require $m_j = m'_j$ for all $j \neq i$. That is, any pair of messages $\{m, m'\}$ that differ only in the i^{th} bit form a calibrator for t_i in t.

The fundamental idea behind possibilistic independence is to allow queries to oracles of a single block $\bar{t}_i$ from a query to the whole timing model t:

Proposition 5. *Let an implementation (f, t) be composed of blocks $(f_1, t_1), \ldots, (f_n, t_n)$. If m_0 and m_1 are equivalent for $\mathsf{Query}(\bar{t}_{-i})$, then*

$$t(k, m_0) - t(k, m_1) = \bar{t}_i(k, m_0) - \bar{t}_i(k, m_1)\,.$$

This formalises the simple idea that, when restricted to a domain where $\bar{t}_{-i}$ is constant, comparing execution times w.r.t. t is equivalent to comparing execution times w.r.t. $\bar{t}_i$. This enables information aggregation by divide-and-conquer under the independence assumption, hence equality in Proposition 4:

Corollary 6. *Let (f, t) be independently composed of the blocks $(f_1, t_1), \ldots, (f_n, t_n)$. Then*

$$\mathsf{dLeak}(t) = \bigcap_{i=1}^{n} \mathsf{dLeak}(\bar{t}_i)$$

D. Probabilistic Independence

We now state a probabilistic variant of the notion of independence of blocks introduced above. Probabilistic independence is a property about the probability distribution of timings. This allows for reconstructing the timing behaviour of a single block *in expectation* from the overall execution time, thus avoiding the task of identifying calibrators (see previous section).

Definition 3. Let X be a uniform random variable on M. We say that the blocks $(f_1, t_1), \ldots, (f_n, t_n)$ are *probabilistically independent* if for all $k_1, \ldots, k_n \in K$, the random variables $\bar{t}_1(k_1, X), \ldots, \bar{t}_n(k_n, X)$ are mutually independent.

Example 8. The blocks of the Hamming distance timing model (see Example 7) are probabilistically independent. This is because, for any $k_0, \ldots, k_{n-1} \in \{0,1\}$ and $X = X_0 \cdots X_{n-1}$ returning vectors of n independent and uniformly distributed bits, the bits $k_0 \oplus X_0, \ldots, k_{n-1} \oplus X_{n-1}$ are also independent and uniformly distributed.

The fundamental idea behind possibilistic independence is to allow for computing expectations of $\bar{t}_i$ from queries to the whole timing model t:

Proposition 7. *Let (f, t) be composed of probabilistically independent blocks $(f_1, t_1), \ldots, (f_n, t_n)$. If $M' \subseteq M$ is non-empty, we write $\mu(t, M') = \frac{1}{|M'|} \sum_{m \in M'} t(k, m)$ the expectation of $t(k, \cdot)$ on M'. Then for all $M_0, M_1 \in \mathsf{Sample}(\bar{t}_i)$,*

$$\mu(t, M_0) - \mu(t, M_1) = \mu(\bar{t}_i, M_0) - \mu(\bar{t}_i, M_1)$$

Proof. It sufficies to remark that $\mu(t, M')$ is the conditional expectation $\mathbb{E}[t(k, X) \mid X \in M']$, and that the events "$X \in M_b$" are equivalent to events of the form "$\bar{t}_i(k_b, X) = o_b$" for some $k_b \in K$ and $o_b \in O$. The result then follows from the

assumption of independence, more precisely from the fact that for all $b \in \{0,1\}$,

$$\mathbb{E}[\bar{t}_{-i}(k,X) \mid \bar{t}_i(k_b,X) = o_b] = \mathbb{E}[\bar{t}_{-i}(k,X)]\,. \qquad \square$$

This propositions formalises the idea that, over the preimage samples of $\bar{t}_i$, comparing timing expectations w.r.t. t is equivalent to comparing timing expectations w.r.t. $\bar{t}_i$. This therefore enables information aggregation by divide-and-conquer, hence equality in Proposition 4:

Corollary 8. *Let (f,t) be composed of the probabilistically independent blocks $(f_1,t_1),\ldots,(f_n,t_n)$. Then*

$$\mathsf{dLeak}(t) = \bigcap_{i=1}^{n} \mathsf{dLeak}(\bar{t}_i)$$

IV. Deterministic Timing Attacks

A timing attack against an implementation (f,t) is an algorithm which, for unknown k^*, outputs the class $[k^*]$ of $\mathsf{dLeak}(t)$, given only oracle access to $t(k^*,\cdot)$.

In this section we describe and analyse a timing attack by divide-and-conquer against implementations that are composed of independent blocks. To illustrate the gain in efficiency by divide-and-conquer, we present a bruteforce attack that serves as a baseline for comparison. We begin by discussing the cost model we use for evaluating attacks in this paper.

A. Cost model

We refer to the equivalence classes of $\mathsf{dLeak}(t)$ and $\mathsf{Query}(t)$ as the *reduced value table* of t. For reasoning about the cost of attacks we assume that the adversary can access equivalence classes of $\mathsf{dLeak}(t)$ and $\mathsf{Query}(t)$ in time $\mathcal{O}(1)$. For convenience of notation, we write $|\mathsf{dLeak}(t)|$ and $|\mathsf{Query}(t)|$ to refer to the correponding numbers of equivalence classes.

We express the cost of attack against a composition of implementations $(f_1,t_1),\ldots,(f_n,t_n)$ in terms of

- number of calls to the oracle $t(k^*,\cdot)$ (*online* steps)
- number of accesses to the reduced value tables of the $\bar{t}_i$ (*offline* steps)

This cost model captures an adversary that can make use of obvious redundancy in a value-table representation of t, but that cannot exploit any deeper mathematical structure of t. We currently do not take into account the cost of constructing the tables. Note that bounds w.r.t this notion of cost do *not* represent lower bounds for performing attacks against functions with richer mathematical structure.

The cost analysis we perform is parametric in the sizes of the reduced value tables of each component $\bar{t}_i$. For bit-serial implementations with information-flow across the blocks, leveraging the knowledge $K' \subseteq K$ gained by attacking blocks $\bar{t}_0 \ldots \bar{t}_{i-1}$ when attacking block $\bar{t}_i$ can significantly reduce the cost of an attack. In our model, this means considering $\mathsf{dLeak}(\bar{t}_i, K')$ and $\mathsf{Query}(\bar{t}_i, K')$ instead of $\mathsf{dLeak}(\bar{t}_i)$ and $\mathsf{Query}(\bar{t}_i)$. This reduction is orthogonal to the one considered in the algebraic statements of Corollaries 6 and 8 and the remainder of the paper, and we use it only as a heuristic.

B. Attack by Brute Force

The attack follows the ideas developed in Section II-D, in particular Proposition 2 (ii). It queries $t^* = t(k^*,\cdot)$ on representatives of each equivalence class of $\mathsf{Query}(t)$, and compares the resulting vector to those that would be obtained for different equivalence classes of $\mathsf{dLeak}(t)$. This is detailed in Algorithm 1.

Algorithm 1: A timing attack by brute force

Parameters
- (f,t) : an implementation
- t^* : an oracle to $t(k^*,\cdot)$ with unknown k^*
- $K' \subseteq K$: prior attacker knowledge on k^*
- $M' \subseteq M$: restriction of the set of queries

def $\mathsf{BFAttack}(t, t^*, K', M')$ **=**
- Pick $m_1,\ldots,m_s \in M'$, representants of the equivalence classes of $\mathsf{Query}(t,K')$
- Compute $o^* = (t^*(m_1),\ldots,t^*(m_s))$
- **for** $[k]$ *equiv. class of* $\mathsf{dLeak}(t,K')$ **do**
  - Compute $o = (t(k,m_1),\ldots,t(k,m_s))$
  - **if** $\|o^* - o\| = 0$ **then return** $[k]$

Overall, Algorithm 1 recovers the equivalence class of the secret in one traversal of the reduced value table.

Proposition 9. *For every $k^* \in K$, Algorithm 1 recovers the equivalence class $[k^*]$ of $\mathsf{dLeak}(t)$ with $|\mathsf{Query}(t)|$ online steps and $|\mathsf{dLeak}(t)|\,|\mathsf{Query}(t)|$ offline steps.*

The bruteforce algorithm is in general impractical. Both $\mathsf{dLeak}(t)$ and $\mathsf{Query}(t)$ may contain as many equivalence classes as K, which makes exhaustive exploration infeasible for targets such as cryptographic algorithms. We next discuss divide-and-conquer attacks that make efficient use of the structure of composed programs.

C. Attack by Divide-and-Conquer

We next describe attacks against implementations that are composed of independent blocks $(f_1,t_1),\ldots,(f_n,t_n)$. The core idea is to perform bruteforce attacks on the subprograms $\bar{t}_i$ by choosing inputs keeping $\bar{t}_{-i}$ constant, as described in Section III-C (Proposition 5). Algorithm 2 gives a formal account of this procedure.

Proposition 10. *Algorithm 2 recovers the equivalence class $[k^*]$ of $\mathsf{dLeak}(t)$ with $\sum_{i=1}^{n} |\mathsf{Query}(\bar{t}_i)|$ online steps and $\sum_{i=1}^{n} |\mathsf{dLeak}(\bar{t}_i)|\,|\mathsf{Query}(\bar{t}_i)|$ offline steps.*

Algorithm 2: An attack by divide and conquer

PARAMETERS
$(f_i, t_i)_{i \in \mathbb{N}_n}$: independent blocks of (f, t)
t^* : an oracle to $t(k^*, \cdot)$ with unknown k^*.

def $\mathsf{DCAttack}(t, t^*)$ =
$K' = K$
for $i \in \mathbb{N}_n$ **do**
Pick a calibrator M' for t_i in t
$K' = \mathsf{BFAttack}(\bar{t}_i, t^*, K', M')$
return K'

While Definition 2 guarantees the existence of witnesses for t_i in t, note that Algorithm 2 abstracts from the cost of identifying such inputs—which may be intractable for some programs—in the count of offline steps. We leave an investigation of this problem to future work. The following example illustrates the potential gain in efficiency of aggregation by divide-and-conquer.

Example 9. Consider again the Hamming distance and its decomposition in Example 7. Each pair m_1, m_2 that differs only in bit i form a calibrator $M' = \{m_1, m_2\}$ for $\bar{t}_i$ in t. Hence, by Proposition 10, Algorithm 2 recovers the key with

$$\sum_{i=0}^{n-1} |\mathsf{Query}(\bar{t}_i)| = 2n \qquad \text{online steps}$$

$$\sum_{i=0}^{n-1} |\mathsf{dLeak}(\bar{t}_i)|\,|\mathsf{Query}(\bar{t}_i)| = 4n \qquad \text{offline steps}$$

Typically, one instance of the attack can be rephrased as the more intelligible decision criterion

$$k_i^* = 0 \quad \textit{iff} \quad t^*(0) < t^*(2^i)\,.$$

In contrast, the bruteforce Algorithm 1, performs

$$\prod_{i=0}^{n-1} |\mathsf{Query}(\bar{t}_i)| = 2^n \qquad \text{online steps}$$

$$\prod_{i=0}^{n-1} |\mathsf{dLeak}(\bar{t}_i)|\,|\mathsf{Query}(\bar{t}_i)| = 4^n \qquad \text{offline steps.}$$

which illustrates the efficiency gained by exploiting the program structure. Note that here 2^n is the number of symbolic executions of the program. Hence, on such examples, symbolic approaches such as [10], [12], [13], [19] enumerating all symbolic executions are not likely to scale.

V. RANDOMISED TIMING ATTACKS

We follow the same approach as the previous section, but resorting to the probabilistic variant of independence. The key difference to the deterministic setting is that attacks do not rely on identifying calibrators. In this context, the attacker chooses the number of (random) online steps, depending on their resources. Our goal is to study the relation between the number r of online steps and the probability of success of the attack.

A. Randomised Attack by Brute Force

The attack follows the ideas developed in Section II-D, Proposition 2 (iii). By randomly querying the oracle $t^* = t(k^*, \cdot)$, it approximates the expectation of t^* on the samples of $\mathsf{Sample}(t)$, and compares the resulting vector to the vectors that would be obtained for different classes $[k]$ of $\mathsf{dLeak}(t)$. The attack is detailed in Algorithm 3.

Algorithm 3: Randomised attack by bruteforce

PARAMETERS
(f, t) : an implementation
t^* : an oracle to $t(k^*, \cdot)$ with unknown k^*
$K' \subseteq K$: prior attacker knowledge on k^*
M' : multiset of r random messages

def $\mathsf{BFRAttack}(t, t^*, K', M')$ =
$o^* = \left(\frac{1}{|M' \cap S|} \sum_{m \in M' \cap S} t^*(m) \right)_{S \in \mathsf{Sample}(t, K')}$
for $[k]$ *in* $\mathsf{dLeak}(t, K')$ **do**
$o_k = \left(\frac{1}{|S|} \sum_{m \in S} t(k, m) \right)_{S \in \mathsf{Sample}(t, K')}$
return $\underset{[k] \in \mathsf{dLeak}(t, K')}{\operatorname{argmin}} \|o^* - o_k\|$

Intuitively, the attack works because the empirical vector o^* obtained from timing measurements gets close to the right ideal vector o_{k^*} with high probability if the number of random samples r is great enough. In particular, the attack is guaranteed to succeed if the distance between the two vectors goes below a certain threshold

$$\delta(t) = \frac{1}{2} \min\{\|o_k - o_{k'}\| \neq 0\}\,.$$

The actual cost analysis is stated in Proposition 11 below, where M' is seen as a random variable obtained from r independent, uniform samplings in M.

Proposition 11. *Algorithm 3 recovers the equivalence class* $[k^*]$ *of* $\mathsf{dLeak}(t)$ *with*

$$|\mathsf{dLeak}(t)|\,(r + |\mathsf{dLeak}(t)|\,|\mathsf{Query}(t)|)$$

offline steps, and probability of failure bounded by

$$|\mathsf{dLeak}(t)|\,\frac{3\alpha^r}{p^2}$$

with $p = \frac{\min\{|S| \;\mid\; S \in \mathsf{Sample}(t)\}}{|M|}$ *and* $\alpha = 1 - p + pe^{-\frac{2\delta(t)^2}{\|t\|^2}}$.

Proof. Regarding the count of offline steps, the computation of o^* can be done in $|\mathsf{dLeak}(t)|$ traversals of the sample M', one for each different equivalence relation $\approx_k$. This requires

$|\mathsf{dLeak}(t)|\, r$ offline steps. Then computing the vectors o_k requires $|\mathsf{dLeak}(t)|$ traversals of the reduced value table of t, adding $|\mathsf{dLeak}(t)|^2\,|\mathsf{Query}(t)|$ steps to the count.

As for the failure probability, the idea is to bound it by $\mathbb{P}[\|o^* - o_{k^*}\| \geqslant \delta(t)]$. Then, to bound this quantity by $|\mathsf{dLeak}(t)|\,\frac{3\alpha^r}{p^2}$, we apply the *law of total expectations* to fix the size of $M' \cap S$ and then *Hoeffding's inequality*. The technical details are provided in Appendix B. □

As a corollary, we can fix the maximal probability of failure tolerated ε, and deduce a number of random online steps r sufficient to obtain it. This is formalised below, and the technical computations are detailed in Appendix B.

Corollary 12. *For all $\varepsilon > 0$, Algorithm 3 has a probability of failure of at most ε, provided that*

$$r \geqslant \frac{a}{p}\left(\ln\frac{1}{\varepsilon} + b\right),$$

where $a = \frac{3\|t\|^2}{2\delta(t)^2}$ and $b = 2\ln\frac{|\mathsf{dLeak}(t)|}{p} + \ln 3$.

In the count of online and offline steps, the dominant terms are $\frac{1}{p}$ and the size of the reduced value table. For example for the Hamming distance in Example 1, we have $\frac{1}{p} = |K| = 2^n$ for bitvectors of length n. We next discuss divide-and-conquer attacks that make efficient use of probabilistic independence.

B. Attack by Divide-and-Conquer

We next describe attacks against composition of probabilistically independent blocks $(f_1, t_1), ..., (f_n, t_n)$. The core idea is that the bruteforce attack on $\bar{t}_i$ can be performed with oracle to the whole t without the need to account for the noise induced by the complement $\bar{t}_{-i}$, as described in Section III-D (Proposition 7). Algorithm 4 gives a formal account of this simple procedure.

Algorithm 4: Randomised attack by divide & conquer

PARAMETERS
- $(f_i, t_i)_{i\in\mathbb{N}_n}$: probab. indep. blocks of (f, t)
- t^* : an oracle to $t(k^*, \cdot)$ with unknown k^*
- r : number of random online steps

def $\mathsf{DCRAttack}(t, t^*, r)$ **=**
- M' = sample of r messages
- $K' = K$
- **for** $i \in \mathbb{N}_n$ **do**
  - $K' = \mathsf{BFRAttack}(\bar{t}_i, t^*, K', M')$
- **return** K'

The cost of this attack is mostly the sum of the costs of the attacks on each independent blocks.

Proposition 13. *Algorithm 4 recovers the equivalence class $[k^*]$ of $\mathsf{dLeak}(t)$ with*

$$\sum_{i=1}^{n} |\mathsf{dLeak}(\bar{t}_i)|\,(r + |\mathsf{dLeak}(\bar{t}_i) \times \mathsf{Query}(\bar{t}_i)|)$$

offline steps, and probability of failure bounded by

$$\sum_{i=1}^{n} |\mathsf{dLeak}(\bar{t}_i)|\,\frac{3\alpha_i^r}{p_i^2},$$

$p_i = \frac{\min\{|S| \;\mid\; S\in\mathsf{Sample}(\bar{t}_i)\}}{|M|}$ and $\alpha_i = 1 - p_i + p_i e^{-\frac{2\delta(\bar{t}_i)^2}{\sum_j \|\bar{t}_j\|^2}}$.

As for the bruteforce attack, inverting the relation between r and the probability of failure gives an upper bound on the number of random online steps needed to guarantee a success rate of at least $1 - \varepsilon$.

Corollary 14. *For all $\varepsilon > 0$, Algorithm 4 has a probability of failure of at most ε, provided that*

$$r \geqslant \frac{an}{\min_i p_i}\left(\ln\frac{n}{\varepsilon} + b\right),$$

where $a = \frac{3\max_i \|\bar{t}_i\|^2}{2\min_i \delta(\bar{t}_i)^2}$ and $b = 2\ln\frac{\max_i|\mathsf{dLeak}(\bar{t}_i)|}{\min_i p_i} + \ln 3$.

The following example illustrates the potential gain in efficiency of aggregation by divide-and-conquer. We support it by an experimental evaluation detailed below.

Example 10. Consider once again the Hamming distance and its independent decomposition in Example 7. We have $\mathsf{Sample}(\bar{t}_i) = \{M_i^0, M_i^1\}$, where M_i^b is the set of bitvectors whose i^{th} bit is b. That is,

$$M_i^b = \{m \in \{0,1\}^n \mid m_i = b\}.$$

In particular we have $p_i = \frac{1}{2}$ for all i and Proposition 13 and corollary 14 therefore justify that

$$\mathcal{O}\left(n\ln\frac{n}{\varepsilon}\right) \qquad \text{random online steps}$$
$$\mathcal{O}\left(n^2\ln\frac{n}{\varepsilon}\right) \qquad \text{offline steps}$$

are sufficient to carry out an attack against this timing model with probability of success at least $1-\varepsilon$. Typically, the attack can be rephrased as the more intelligible decision criterion

$$k_i^* = 0 \quad \textit{iff} \quad \frac{1}{|M_i^0|}\sum_{m\in M_i^0} t^*(m) < \frac{1}{|M_i^1|}\sum_{m\in M_i^1} t^*(m).$$

Experimental validation We conclude this section with an experimental validation of the above online-cost analysis of the attack against the Hamming distance. We do so by computing the probability of success of Algorithm 4 on a simulated Hamming distance timing model. Our goal is to estimate the number of measurements needed to achieve a desired probability of success $1-\varepsilon$ for bitvectors of different lengths n.

To this end, we fix the length n of the secret bit-vectors and a target success probability of $1-\varepsilon$ (in our case: 75%). For a

constant number of rounds N, we randomly select a key k^*, sample r independent messages from a uniform distribution, and compute the success probability of Algorithm 4 over these N rounds. We then adapt the number of queries r by binary search, and repeat the process with the goal of matching the target success probability $1-\varepsilon$. We finally return the average number of online steps obtained this way for several keys k^*.

Figure 1 depicts the curve of the r that achieve a probability of success of $1-\varepsilon$ as a function of the input size n, together with an interpolation in $\mathcal{O}(n \ln n)$.

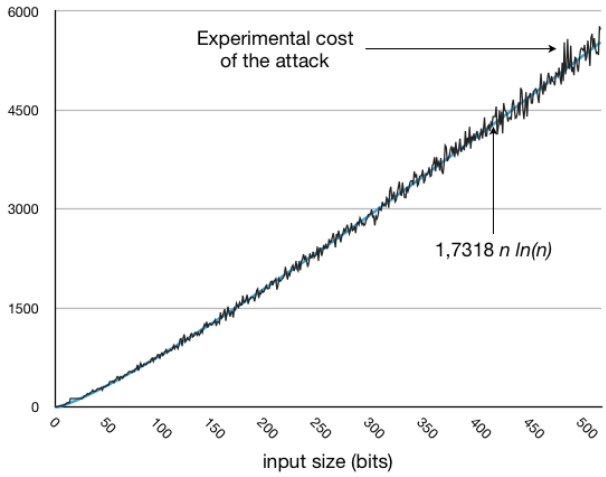

Fig. 1. Number of online steps needed to ensure success probability of 0.75 for Algorithm 4 when targetting the Hamming distance timing model as a function of input size n.

VI. Case Studies

Our model explains timing attacks retrieving secret data part by part from measurements of the whole execution time of a program; see high-profile attacks of the literature against implementations of RSA [4], [5], AES [14], [15] or ECDH [20]. In this section we detail two such examples [5], [14]. In both cases we describe the target, the referenced attack against it, and explain in what sense it is an instance of Algorithm 4.

A. RSA With Montgomery Multiplication

The target (reference) We consider an implementation of RSA based on bit-serial modular exponentiation. Here

$$K = \{k \in \{0,1\}^n \mid k_{n-1} = 1\}$$

is the set of bitvectors whose most significant bit is set to 1, and $M = \mathbb{Z}/p\mathbb{Z}$ is the set of integers modulo p. Multiplication in $\mathbb{Z}/p\mathbb{Z}$ is replaced by the more efficient *Montgomery multiplication*, which boils down to a group isomorphism ϕ from $(\mathbb{Z}/p\mathbb{Z}, \times)$ to a group $(G, \otimes)$ where multiplications are easier to compute. The pseudocode for modular exponentiation is given by Algorithm 5.

There is a vulnerability in that $\otimes$ is not constant-time: computing $a \otimes b$ takes longer for some operands a, b because

Algorithm 5: Computing $m^k \pmod p$

```
x = φ(m)
for i = n − 2 downto 0 do
    x = x ⊗ x
    if k_i = 1 then x = x ⊗ φ(m)
return φ^{-1}(x)
```

of a conditional subtraction, usually called a *Montgomery reduction*. The differential execution time of the whole exponentiation is thus proportional to the number of Montgomery reductions that happened during the computation. We model this behaviour using $T : G \times G \to \{0,1\}$ as the timing model of $\otimes$, where $T(a,b) = 1$ means that a Mongomery reduction occurred during the computation of $a \otimes b$.

The attack (reference) We recall an attack presented in [5] that targets the square instruction at loop index $i-1$ to recover k_i^* the i^{th} bit of the secret. The core idea is the following:

(1) Assuming bits $k^*_{n-2}, k^*_{n-3}, \ldots, k^*_{i+1}$ have already been recovered from previous attack steps, one can potentially compute for any initial input m the value x_m of the variable x just before the execution of the instruction "if $k_i = 1$ then $x = x \otimes \phi(m)$".

(2) After this intruction, the value of x may be either x_m or $x_m \otimes \phi(m)$, depending on the value of k_i^*. For each of these two potential values, one splits the set M accordingly to whether an element $m \in M$ induces a Montgomery reduction or not during the next instruction $x \otimes x$. That is, writing $C_x^b = \{m \in \mathbb{Z}/p\mathbb{Z} \mid T(x,x) = b\}$, we compute the two partitions

$$P_0 = \{C^0_{x_m}, C^1_{x_m}\} \quad P_1 = \{C^0_{x_m \otimes \phi(m)}, C^1_{x_m \otimes \phi(m)}\}$$

(3) The intuition is then that the split $P_{k_i^*}$ should be empirically more significant than $P_{1-k_i^*}$. That is, the attack criterion is that $k_i^* = 0$ if and only if

$$\mu(C^1_{x_m}) - \mu(C^0_{x_m}) > \mu(C^1_{x_m \otimes \phi(m)}) - \mu(C^0_{x_m \otimes \phi(m)})$$

where $\mu(S) = \frac{1}{|S|} \sum_{m \in S} t^*(m)$. In practice, $\mu(S)$ is approximised using a sample of messages drawn independently and uniformly beforehand.

Note that an earlier attack sharing common grounds with the above one also exists [4]. It follows a similar development except that the multiply instructions are targeted instead of the square instructions.

The target (in our model) We consider a decomposition where each block corresponds to one instruction in Algorithm 5. There are two main kinds of blocks:

- the square blocks $(f_i^{\mathsf{sq}}, t_i^{\mathsf{sq}})$ performing the instruction, at step i, "$x = x \otimes x$" and
- the multiply blocks $(f_i^{\mathsf{mul}}, t_i^{\mathsf{mul}})$ performing the instruction "if $k_i = 1$ then $x = x \otimes \phi(m)$".

To fit into our model, low inputs are pairs (m, x) where m is the initial input passed to the whole function and x is the intermediary store. The formal description of the block decomposition is provided in Appendix C. In the attack literature [4], [5], the execution times of the different multiplications are assumed to satisfy a kind of independence property, which is what we formalise as probabilistic independence of this block decomposition.

The attack (in our model) The squaring instruction at loop index $i-1$ leaking the value of k_i^* provided prior knowledge of $k_{n-2}^*, \ldots, k_{i+1}^*$ is modelled by

$$\mathsf{dLeak}(\bar{t}_{i-1}^{\mathsf{sq}}, K') = \{K_i^0 \cap K', K_i^1 \cap K'\}$$

with $K_i^b = \{k \in K \mid k_i = b\}$ and $K' = \bigcap_{j=i+1}^{n-2} K_j^{k_j^*}$. In particular this emphasises the need for taking profit of the knowledge aggregated during previous attack steps to keep the search space constant in size at each step, see Section IV-A.

The randomised attack described by Algorithm 4 thus leads to the attack described above. In particular, step (2) describes how to compute

$$\mathsf{Sample}(\bar{t}_{i-1}^{\mathsf{sq}}, K') = P_0 \cup P_1 \,.$$

Then step (3) rephrases more intelligibly the identification of the class $[k^*]$ of $\mathsf{dLeak}(\bar{t}_{i-1}^{\mathsf{sq}}, K')$, which was formalised in our model by generic comparisons of vectors using $\|\cdot\|$.

Experiments A proof of concept of this attack against a smartcard emulator has already been implemented in [5]. However, the focus was on the estimation of the online cost (i.e. the number of measurements needed) rather than on the efficiency of the offline part (i.e. the attack time, measurements excluded). We give here an insight of the growth of offline worktime when the keysize increases.

Attacks derived from Algorithm 4 separates between an initial online phase where all measurements are made, and offline computations that do not depend on how measurements were obtained. For simplicity, we therefore implemented a simulation of the attack where we replaced measurements of real time by a counter of Montgomery reductions. From the same number of measurements as in [5], our simulation recovered a 512-bit RSA key after 6 minutes of offline work.

Besides, such attacks can significantly benefit from parallelism. Indeed they mainly compute partitions and expectations, tasks that can easily be distributed by assigning to parallel subprocesses a fraction of the initial random sample M'. Experimental results are collected in Figure 2.

Keysize	Online steps	Offline work	
		1 core	35 cores
256	70,000	25s	2s
512	300,000	6min	30s
1024	1,200,000	1h40	6min

Fig. 2. Cost of the attack against RSA with Montgomery multiplications

B. AES With Precomputed Tables

The target (reference) We consider an implementation of AES [21] using precomputed T-tables. The timing model of this implementation takes the CPU cache into account, mechanism that is at the core of the vulnerability. We write $(\mathbb{Z}_8, \oplus)$ the group of 8-bit integers (bytes) equipped with xor. Here $K = M = \mathbb{Z}_{128}$ is the set of 128-bit integers seen as sequences of 16 bytes (most significant byte first). The i^{th} byte of $a \in \mathbb{Z}_{128}$ is written a_i.

Tables $T_i : \mathbb{Z}_8 \to (\mathbb{Z}_8)^4$, $0 \leqslant i \leqslant 7$, are precomputed as well as round keys $k^{(r)} \in K$, $1 \leqslant r \leqslant 10$, that are derived from the secret $k \in K$ (details in the specification [21]). Besides we define, if $x \in \mathbb{Z}_{128}$:

$$L_0(x) = (x_0, x_4, x_8, x_{12}) \qquad L_2(x) = (x_{10}, x_{14}, x_2, x_6)$$
$$L_1(x) = (x_5, x_9, x_{13}, x_1) \qquad L_3(x) = (x_{15}, x_3, x_7, x_{11})$$

If $T : \mathbb{Z}_8 \to (\mathbb{Z}_8)^4$ and $L(x) = (x_{\ell_1}, x_{\ell_2}, x_{\ell_3}, x_{\ell_4})$ we write $T[L(x)] \in \mathbb{Z}_{128}$ the integer obtained by concatenation of the four outputs $T[x_{\ell_1}], T[x_{\ell_2}], T[x_{\ell_3}], T[x_{\ell_4}]$. Encryption of message $m \in M$ is then described in algorithm 6.

Algorithm 6: 128-bit AES with precomputed tables

$x \leftarrow k \oplus m$

for $r = 1$ **to** *9* **do**
  $x = k^{(r)} \oplus \bigoplus_{\ell=0}^{3} T_\ell[L_\ell(x)]$

$x = k^{(10)} \oplus \bigoplus_{\ell=0}^{3} T_{\ell+4}[L_\ell(x)]$

return x

The control flow of this program is not dependent on the input. All timing variations are due to the effect of the program on the cache.

If a table entry is requested but is not in the cache, it has to be fetched from main memory, producing a slow cache *miss*. If a table entry is already in the cache it can be served from there, producing a fast cache *hit*. In the differential model, the timing model t captures the total number of cache misses occurring during the program execution; that is, we use the convention that the execution time of a memory access is 0 or 1 depending on whether it is a cache hit or a cache miss.

The attack (reference) We recall the two-round attack presented in [14] that targets 20 memory accesses spread over the first two rounds of the loop of Algorithm 6. We only present here the attack on the memory accesses of the first round, as the second-round attack follows the exact same development if not for more intricate reduced value tables. Omitted details can easily be inferred from the attack description in [14]. The core idea is the following:

(1) The first round of the loop contains 16 memory accesses, 12 of which inducing timing variations (the first access to each table $T_0, \ldots, T_3$ is always a miss).

(2) For each of these 12 blocks, targeted by order of appearance, whether a cache hit occurs is determined by an equation of the form

$$\langle k_i^* \oplus k_j^* \rangle = g(m)$$

where $\langle \alpha \rangle = \lfloor \frac{\alpha}{16} \rfloor$ for common microarchitectures. For each of the 16 potential values of $\langle k_i^* \oplus k_j^* \rangle$, one splits the set M accordingly to whether input $m \in M$ induces a cache hit or a cache miss. That is, writing

$$C_\ell^{\sim} = \{m \in M \mid \langle k_i^* \oplus k_j^* \rangle \sim g(m)\},$$

we compute the 16 partitions

$$P_\ell = \{C_\ell^{=}, C_\ell^{\neq}\} \qquad 0 \leqslant \ell \leqslant 15$$

(3) The intuition is then that the split $P_{\langle k_i^* \oplus k_j^* \rangle}$ should be empirically more significant than the 15 others. That is, given oracle t^*, the criterion is

$$\langle k_i^* \oplus k_j^* \rangle = \operatorname*{argmax}_{0 \leqslant \ell \leqslant 15} \mu(C_\ell^{\neq}) - \mu(C_\ell^{=})$$

where $\mu(S) = \frac{1}{|S|} \sum_{m \in S} t^*(m)$. In practice, $\mu(S)$ is approximised using a sample of messages drawn independently and uniformly beforehand.

This permits to recover the equivalent of 48 bits of the secret. The 80 remaining bits are recovered similarly by targeting the four memory accesses to T_0 of the second round of the loop. This is however a costly attack as these four blocks exhibit quite large reduced value tables (sizes 2^{12} to 2^{32}).

The target (in our model) We consider a decomposition where each block corresponds to one of the 160 memory accesses of the implementation. Messages are pairs (C, x), where C is the state of the cache and x is the intermediary state of the computation. A formal description of the blocks is provided in Appendix C. In [14], the memory accesses are assumed to verify an independence property, which is what we formalise as probabilistic independence of this block decomposition.

In [14], the assumption is made that the target is located on a distant server with enough workload to guarantee that any data related to AES is evicted of the cache between two measurements. For local targets, the attacker may also fetch garbage data into the cache by meaningless memory accesses to force this eviction. This makes the timing behaviour of t only depend on arguments k and m, and not on previous invocations of the oracle.

The attack (in our model) If (f_n, t_n) is a block corresponding to one of the 12 leaking memory accesses of the first round, and if K' is an equivalence class of $\mathsf{dLeak}(\bar{t}_{n-1})$, then $\mathsf{dLeak}(\bar{t}_n, K')$ has 16 equivalence classes. This models the fact that targeting a memory access of the first round permits to identify the value of a $\langle k_i \oplus k_j \rangle$ among the 16 possible such.

The randomised attack described by Algorithm 4 thus leads to the attack of [14]. In particular, step (2) describes how to compute

$$\mathsf{Sample}(\bar{t}_n, K') = \bigcup_{\ell=0}^{15} P_\ell \,.$$

Then step (3) rephrases more intelligibly the identification of the class $[k^*]$ of $\mathsf{dLeak}(\bar{t}_n, K')$, which was formalised in our model by generic comparisons of vectors using $\|\cdot\|$.

VII. Related Work

A number of models for expressing aggregate leakage are based on discrete representations of adversary knowledge as partitions, such as [7], [22]–[24]. Other approaches capture knowledge probabilistically, in terms of adversary beliefs [25], [26], which are updated by Bayesian revision.

Both kinds of approaches have been automated. The first approach for the discrete case is [8], which uses expensive quantifier elimination on linear arithmetic to intersect partitions corresponding to different runs. More recent approaches use symbolic execution to [11], [12], [19] compute preimages along the paths of a program. Automation for the probabilistic model has been considered in [9], [10], [13].

Our approach differs from previous work in that we aim to model real-world attacks rather than to derive worst-case bounds. On a technical level this is reflected in two aspects: first, our differential timing model does not require having an exact (probabilistic or deterministic) model of the execution time. Second, our approach does not rely on the enumeration of secrets or paths, which is why it can model and explain efficient attacks on programs with exponentially many paths, such as RSA. We currently investigate the use of our model as a basis for automatic vulnerability analysis.

A line of work investigates the special case of aggregation via the termination behaviour of programs. While termination behaviour can in principle leak unbounded amounts of information [27], this leakage is limited to one bit if the adversary cannot control the termination behaviour [28]. In our model, this observation corresponds to a noninterference result w.r.t. differential observers, see Proposition 3.

Technically different but similar in spirit to our work is an analytic model that explains the efficiency of timing attacks for the special case of AES [29], and a comprehensive approach for reasoning about the efficiency of power analysis attacks [30].

Finally, while our model is cast in the language of information-flow analysis, some central ideas (such as probabilistic independence as a basis for attacks by divide-and-conquer) are inspired from the literature on attacks, in particular [5].

VIII. Conclusion

We presented a novel model for reasoning about information aggregation in timing attacks that bridges attack research with information-flow analysis. From the point of view of attack research, we provide the first uniform explanation of high-profile timing attacks in a simple formal model. From the point of view of information-flow analysis, we provide novel notions of timing measurements and compositionality that are a promising basis for improving realism and scalability of automatic vulnerability analysis and attack generation.

Acknowledgments We thank Michele Boreale for initial discussions and the anonymous reviewers for their constructive feedback. Part of this work was carried out at the IMDEA Software Institute, supported by a grant from the Intel Corporation, Ramón y Cajal grant RYC-2014-16766, Spanish projects TIN2015-70713-R DEDETIS and TIN2015-67522-C3-1-R TRACES, Madrid regional project S2013/ICE-2731 N-GREENS. Work at INRIA Nancy Grand-Est was supported by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation program (grant 645865-SPOOC), and the French National Research Agency (ANR) project TECAP (ANR-17-CE39-0004-01).

References

[1] P. Kocher, J. Jaffe, and B. Jun, "Differential power analysis," in *Annual International Cryptology Conference (CRYPTO)*, 1999.

[2] J.-J. Quisquater and D. Samyde, "Electromagnetic analysis (ema): Measures and counter-measures for smart cards," in *Smart Card Programming and Security*, 2001.

[3] K. Gandolfi, C. Mourtel, and F. Olivier, "Electromagnetic analysis: Concrete results," in *International workshop on cryptographic hardware and embedded systems*, 2001.

[4] P. C. Kocher, "Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems," in *Annual International Cryptology Conference (CRYPTO)*, 1996.

[5] J.-F. Dhem, F. Koeune, P.-A. Leroux, P. Mestré, J.-J. Quisquater, and J.-L. Willems, "A practical implementation of the timing attack," in *International Conference on Smart Card Research and Advanced Applications (CARDIS)*, 1998.

[6] D. Brumley and D. Boneh, "Remote timing attacks are practical," *Computer Networks*, 2005.

[7] B. Köpf and D. Basin, "An Information-Theoretic Model for Adaptive Side-Channel Attacks," in *ACM Conference on Computer and Communications Security (CCS)*, 2007.

[8] M. Backes, B. Köpf, and A. Rybalchenko, "Automatic Discovery and Quantification of Information Leaks," in *IEEE Symposium on Security and Privacy (S& P)*, 2009.

[9] P. Mardziel, S. Magill, M. Hicks, and M. Srivatsa, "Dynamic enforcement of knowledge-based security policies," in *IEEE Computer Security Foundations Symposium (CSF)*, 2011.

[10] P. Malacaria, M. Khouzani, C. Pasareanu, Q.-S. Phan, K. Luckow *et al.*, "Symbolic side-channel analysis for probabilistic programs," in *IEEE Computer Security Foundations Symposium (CSF)*, 2018.

[11] Q. H. Do, R. Bubel, and R. Hähnle, "Exploit generation for information flow leaks in object-oriented programs," in *ICT Systems Security and Privacy Protection*, 2015.

[12] Q. Phan, L. Bang, C. S. Pasareanu, P. Malacaria, and T. Bultan, "Synthesis of adaptive side-channel attacks," in *IEEE Computer Security Foundations Symposium (CSF)*, 2017.

[13] L. Bang, N. Rosner, and T. Bultan, "Online synthesis of adaptive side-channel attacks based on noisy observations," in *IEEE European Symposium on Security and Privacy (EuroS&P)*, 2018.

[14] O. Acıiçmez, W. Schindler, and Ç. K. Koç, "Cache-based remote timing attack on the AES," in *Cryptographers' Track at the RSA Conference (CT-RSA)*, 2007.

[15] D. Bernstein, "Cache-timing attacks on AES," document available at https://cr.yp.to/antiforgery/cachetiming-20050414.pdf, 2005.

[16] D. A. Osvik, A. Shamir, and E. Tromer, "Cache attacks and countermeasures: the case of AES," in *CT-RSA*, 2006.

[17] J. Landauer and T. Redmond, "A lattice of information," in *IEEE Computer Security Foundations Symposium (CSF)*, 1993.

[18] A. Sabelfeld and D. Sands, "Declassification: Dimensions and principles," *Journal of Computer Security*, 2009.

[19] P. Malacaria, C. S. Pasareanu, and Q. Phan, "Multi-run side-channel analysis using symbolic execution and max-SMT," in *IEEE Computer Security Foundations Symposium (CSF)*, 2016.

[20] T. Kaufmann, H. Pelletier, S. Vaudenay, and K. Villegas, "When constant time source yields variable-time binary: Exploiting curve25519-donna built with msvc 2015," in *International Conference on Cryptology and Network Security (CANS)*, 2016.

[21] J. Daemen and V. Rijmen, "The design of rijndael: Aes-the advanced encryption standard," 2013.

[22] A. Askarov and A. Sabelfeld, "Gradual release: Unifying declassification, encryption and key release policies," in *IEEE Symposium on Security and Privacy (S& P)*, 2007.

[23] M. Boreale and F. Pampaloni, "Quantitative information flow under generic leakage functions and adaptive adversaries," *Logical Methods in Computer Science*, 2015.

[24] M. S. Alvim, M. E. Andrés, and C. Palamidessi, "Quantitative information flow in interactive systems," *Journal of Computer Security*, 2012.

[25] M. R. Clarkson, A. C. Myers, and F. B. Schneider, "Quantifying information flow with beliefs," *Journal of Computer Security*, 2009.

[26] P. Mardziel, M. S. Alvim, M. W. Hicks, and M. R. Clarkson, "Quantifying information flow for dynamic secrets," in *IEEE Symposium on Security and Privacy (S&P)*, 2014.

[27] A. Askarov, S. Hunt, A. Sabelfeld, and D. Sands, "Termination-insensitive noninterference leaks more than just a bit," in *European Symposium on Research in Computer Security (ESORICS)*, 2008.

[28] A. Birgisson and A. Sabelfeld, "Multi-run security," in *European Symposium on Research in Computer Security (ESORICS)*, 2011.

[29] K. Tiri, O. Aciiçmez, M. Neve, and F. Andersen, "An analytical model for time-driven cache attacks," in *Fast Software Encryption (FSE)*, 2007.

[30] F. Standaert, T. Malkin, and M. Yung, "A unified framework for the analysis of side-channel key recovery attacks," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT)*, 2009.

Appendix A
On the Characterisation of Timing Leaks

In Section II-D, we formalised criterions to identify whether two secrets were producing the same differential-time oracles (Proposition 2). The proof of the implication $(iii) \Rightarrow (i)$ was left pending in the paper body. We prove the following lemma.

Lemma 15. *Let* $\mathsf{Sample}(t) = \{M_1, \ldots, M_q\}$ *and the notation* $\mu_i(k) = \frac{1}{|M_i|} \sum_{m \in M_i} t(k, m)$. *If for all* $i \in [\![1, q]\!]$, $\mu_i(k) = \mu_i(k')$, *then* $t(k, \cdot) = t(k', \cdot)$.

This lemma easily implies the missing proof, keeping in mind that $[k] = [k']$ in $\mathsf{dLeak}(t)$ *iff* there exists a constant c such that $t(k, \cdot) = t(k', \cdot) + c$ (Equation (2)). We prove the following more general result on oracles:

Lemma 16. *Let* h, h' *be two oracles s.t. for all* $o \in O$,

(i) *if* $h'^{-1}(o) \neq \emptyset$, *then* $\frac{1}{|h'^{-1}(o)|} \sum_{m \in h'^{-1}(o)} h(m) = o$;

(ii) *if* $h^{-1}(o) \neq \emptyset$, *then* $\frac{1}{|h^{-1}(o)|} \sum_{m \in h^{-1}(o)} h'(m) = o$.

Then $h = h'$.

This lemma implies Lemma 15. To see that it sufficies to choose $h = t(k,\cdot)$ and $h' = t(k',\cdot)$, and remark that $\mathsf{Sample}(t)$ contains all non-empty samples $h^{-1}(o)$ and $h'^{-1}(o)$. With $h^{-1}(o) = M_i$, note in particular the argument that $\frac{1}{|h^{-1}(o)|}\sum_{m\in h^{-1}(o)} h'(m) = \mu_i(k')$ and $o = \mu_i(k)$.

Proof of Lemma 16. Let us exhibit a contradiction under the assumptions (i), (ii) and $h \neq h'$. By symmetry, assume $h(m_0) < h'(m_0)$ for some $m_0 \in M$. Then there needs be $m_1 \in M$ such that $h'(m_0) = h'(m_1) < h(m_1)$, otherwise we would have the following contradiction with (i) for $o = h'(m_0)$:

$$\frac{1}{|h'^{-1}(o)|}\sum_{m\in h'^{-1}(o)} h(m) < o\,.$$

With a symmetric reasoning, the hypothesis (ii) gives a message $m_2 \in M$ such that $h(m_1) = h(m_2) < h'(m_2)$. By iterating this two-step argument, we can thus construct inductively a sequence of messages $(m_i)_{i\in\mathbb{N}}$ such that

$$h'(m_0) < h(m_1) < h'(m_2) < h(m_3) < h'(m_4) < \cdots\,.$$

In particular, the messages m_{2i}, $i \in \mathbb{N}$, are pairwise disjoint, in contradiction with the finiteness of M. □

Appendix B
Cost Analysis for the Randomised Attacks

In Section V we presented randomised timing attacks and an analysis of their costs, but most technical arguments have been omited. The detailed arguments in this section.

Most of the results follow from the following technical statement (with the notations of Algorithm 3 and seeing the multiset M' is seen as a random variable obtained from r independent, uniform samplings in M).

Technical Lemma 17. *Let (f,t) be composed of n probabilistically independent blocks $(f_1,t_1),\ldots,(f_n,t_n)$. Let also $S \subseteq M$ non empty, $t^* = t(k^*,\cdot)$, and*

$$o = \frac{1}{|M'\cap S|}\sum_{m\in M'\cap S} t^*(m) \quad \textit{and} \quad \bar{o} = \frac{1}{|S|}\sum_{m\in S} t^*(m)\,.$$

Then for all $\delta > 0$, we have

$$\mathbb{P}[|o-\bar{o}| \geqslant \delta] \leqslant \frac{3\alpha^r}{p}$$

where $p = \frac{|S|}{|M|}$ and $\alpha = 1 - p + pe^{-\frac{2\delta^2}{\sum_i \|\bar{t}_i\|^2}}$.

This means that in Algorithm 3, the probability of each component of o^* differing from the corresponding component of o_{k^*} beyond a fixed threshold δ can be bounded by a quantity that decreases exponentially as the number of random online steps r increases. The proof of this bound, detailed below, essentially consists of applying the *law of total expectations* to fix the size of $M' \cap S$ and then *Hoeffding's inequality*.

Proof of Technical Lemma 17. First we observe that the random variable o has the same distribution as

$$o' = \frac{1}{N}\sum_{j=1}^{N} t^*(X_j)$$

where X_j is uniformly distributed on S, N has a binomial distribution of parameters r and p, and all are mutually independent. In particular

$$\mathbb{P}[|o-\bar{o}| \geqslant \delta] \leqslant \mathbb{P}[N=0] + \mathbb{P}[|o'-\bar{o}| \geqslant \delta \mid N \neq 0]\,. \quad (3)$$

Then by the law of total expectations, we write

$$\mathbb{P}[|o'-\bar{o}| \geqslant \delta \mid N \neq 0] = \mathbb{E}[\mathbb{P}[|o'-\bar{o}| \geqslant \delta \mid N, N \neq 0]]\,.$$

Besides, the random variables $(\bar{t}_i(k^*, X_j))_{i,j}$ are mutually independent as the blocks $(f_1,t_1),\ldots,(f_n,t_n)$ are probabilistically independent. Therefore, by Hoeffding's inequality,

$$\begin{aligned}\mathbb{P}[|o'-\bar{o}| \geqslant \delta \mid N \neq 0] &\leqslant \mathbb{E}[2e^{\frac{-2\delta^2 N}{\sum_i \|\bar{t}_i^2\|}} \mid N \neq 0] \\ &\leqslant \frac{2}{\mathbb{P}[N\neq 0]}\mathbb{E}[e^{\frac{-2\delta^2 N}{\sum_i \|\bar{t}_i\|^2}}] \\ &\leqslant \frac{2\alpha^r}{p}\end{aligned}$$

by recognising the moment-generating function of N, and because $\mathbb{P}[N \neq 0] = 1-(1-p)^r \geqslant p$. Hence the final result by (3), as $\mathbb{P}[N=0] = (1-p)^r \leqslant \frac{\alpha^r}{p}$. □

This characterises the probability of failure of Algorithm 3 in that it can be bounded by the probability of at least one difference $|o-\bar{o}|$ going beyond the limit threshold

$$\delta(t) = \frac{1}{2}\min\{\|o_k - o_{k'}\| \neq 0\}\,.$$

This is the core argument to prove Proposition 11 which we recall below.

Proposition 11. *Algorithm 3 recovers the equivalence class $[k^*]$ of $\mathsf{dLeak}(t)$ with*

$$|\mathsf{dLeak}(t)|\,(r + |\mathsf{dLeak}(t)|\,|\mathsf{Query}(t)|)$$

offline steps, and probability of failure bounded by

$$|\mathsf{dLeak}(t)|\,\frac{3\alpha^r}{p^2}$$

with $p = \frac{\min\{|S| \mid S\in\mathsf{Sample}(t)\}}{|M|}$ and $\alpha = 1 - p + pe^{-\frac{2\delta(t)^2}{\|t\|^2}}$.

Proof. The count of offline steps is already detailed in the paper body. As for the failure probability, by Proposition 2, if the attack fails then $\|o^* - o_{k^*}\| \geqslant \delta(t)$. Besides,

$$\begin{aligned}&\mathbb{P}[\|o^* - o_{k^*}\| \geqslant \delta(t)] \\ &\quad\leqslant \mathbb{P}[\exists S \in \mathsf{Sample}(t),\, |o(S) - \bar{o}(S)| \geqslant \delta(t)] \\ &\quad\leqslant \sum_{S\in\mathsf{Sample}(t)} \mathbb{P}[|o(S)-\bar{o}(S)| \geqslant \delta(t)]\end{aligned}$$

where we have $o(S) = \frac{1}{|M' \cap S|} \sum_{m \in M' \cap S} t^*(m)$ as well as $\bar{o}(S) = \frac{1}{|S|} \sum_{m \in S} t^*(m)$. Hence the conclusion by Technical Lemma 17, as $|\mathsf{Sample}(t)| \leqslant \frac{1}{p} |\mathsf{dLeak}(t)|$. □

Then Corollary 12 uses this result to exhibit an upper bound on the number of random measurements to guarantee a probability of success $1 - \varepsilon$ in the attack. We prove this result here.

Corollary 12. *For all $\varepsilon > 0$, Algorithm 3 has a probability of failure of at most ε, provided that*

$$r \geqslant \frac{a}{p} \left(\ln \frac{1}{\varepsilon} + b \right) ,$$

where $a = \frac{3\|t\|^2}{2\delta(t)^2}$ and $b = 2 \ln \frac{|\mathsf{dLeak}(t)|}{p} + \ln 3$.

Proof. By Proposition 11, it sufficies to choose r such that

$$|\mathsf{dLeak}(t)| \frac{3\alpha^r}{p^2} \leqslant \varepsilon .$$

By composing with ln, this gives, since $\alpha < 1$,

$$r \geqslant \frac{\ln |\mathsf{dLeak}(t)| + \ln 3 + 2 \ln \frac{1}{p} + \ln \frac{1}{\varepsilon}}{- \ln \alpha} = A .$$

It therefore sufficies to prove that $A \leqslant \frac{a}{p}(\ln \frac{1}{\varepsilon} + b)$, where $a = \frac{3\|t\|^2}{2\delta(t)^2}$ and $b = 2 \ln \frac{|\mathsf{dLeak}(t)|}{p} + \ln 3$.

First we have $A \leqslant \frac{\ln \frac{1}{\varepsilon} + b}{- \ln(\alpha)}$. By applying successively the following identities (whose justifications are omitted):

$$\forall t \in]0,1[, \ \frac{1}{-\ln(1-t)} \leqslant \frac{1}{t} \quad \forall t \in \mathbb{R}_+^*, \ \frac{1}{1 - \mathrm{e}^{-t}} \leqslant 1 + \frac{1}{t} .$$

we obtain $A \leqslant \frac{1}{p}(\ln \frac{1}{\varepsilon} + b)(1 + \frac{\|t\|^2}{2\delta(t)^2})$. Hence the conclusion since $0 < \delta(t) \leqslant \|t\|$, and thus $1 + \frac{\|t\|^2}{2\delta(t)^2} \leqslant a$. □

The cost analyses for the randomised attacks by divide-and-conquer (Section V-B) follow along the same lines. In particular the statement of Technical Lemma 17 is general enough, already taking into account probabilistic independence.

Appendix C
Formal Block Decomposition (Case Studies)

In Section VI, we recalled timing attacks from the literature and outlined sequential decompositions the target implementations. We provide technical details here.

A. RSA With Montgomery Multiplication

The decomposition consists of two blocks performing the computations of ϕ and ϕ^{-1}, and $2(n-1)$ blocks performing the multiplications of the main loop. Formally, we have the $2n-1$ blocks $(\alpha, 0)$, $(f^{\mathsf{sq}}_{n-2}, t^{\mathsf{sq}}_{n-2})$, $(f^{\mathsf{mul}}_{n-2}, t^{\mathsf{mul}}_{n-2})$, …, $(f^{\mathsf{sq}}_0, t^{\mathsf{sq}}_0)$, $(f^{\mathsf{mul}}_0, t^{\mathsf{mul}}_0)$, $(\omega, 0)$, where

- the input set is $M = G \times (\mathbb{Z}/p\mathbb{Z})$ where G is the group for Montgomery multiplications and (x, m) models intermediary store x and initial input m;
- $\alpha(k, (\_, m)) = (\phi(m), m)$ is the initial block;
- $\omega(k, (x, m)) = (\phi^{-1}(x), m)$ is the final block;
- $f^{\mathsf{sq}}_i(k, (x, m)) = x \otimes x$ and $t^{\mathsf{sq}}_i(k, (x, m)) = T(x, x)$;
- $f^{\mathsf{mul}}_i(k, (x, m)) = \begin{cases} (x \otimes m, m) & \text{if } k_i = 1 \\ (x, m) & \text{otherwise} \end{cases}$
  $t^{\mathsf{mul}}_i(k, (x, m)) = \begin{cases} T(x, m) & \text{if } k_i = 1 \\ 0 & \text{otherwise} \end{cases}$

B. AES With Precomputed Tables

For the model, we use the following model of cache:

1) The cache is partitioned into so-called *cache lines* whose size is 64 bytes. One always loads a full line into the cache: since the tables T_i store data of 4 bytes, their entries are therefore always copied into the cache by groups of $\frac{64}{4} = 16$ entries.
2) Within a given table T_i, each entry is associated to a cache line determined by its adress, more precisely by its $\log_2(16) = 4$ most significant bits. That is, using the notation of [14], by $\langle \alpha \rangle = \lfloor \frac{\alpha}{16} \rfloor$.
3) To sum up, given a table T and an entry $\alpha \in \mathsf{Dom}(T)$, two situations can arise when a query $T[\alpha]$ is made:
   a. Cache hit: the entry is already in the cache and $T[\alpha]$ is obtained quickly.
   b. Cache miss: the entry is fetched slowly from the main memory. The full corresponding line (all entries $\beta \in \mathsf{Dom}(T)$ such that $\langle \beta \rangle = \langle \alpha \rangle$) is then loaded into the cache. If the cache capacity is exceeded, some previously-stored data is evicted from the cache.

As in [14] we assume that (1) there is enough workload on the attacked hardware so that AES-table data is evicted between two invocations of the program, and (2) the cache has enough capacity so that data loaded in the cache is not evicted during the same execution of AES. Under these assumptions, there is a cache hit for a query $T[\alpha]$ *iff* a query $T[\beta]$ such that $\langle \beta \rangle = \langle \alpha \rangle$ has already been performed during the execution.

The decomposition eventually consists of one initialisation block, 160 blocks corresponding each to one memory access of the implementation, and 10 blocks updating the store between each round. Formally, we have the 171 blocks $(\alpha, 0)$, B, $(\mathsf{up}, 0)$, …, B, $(\mathsf{up}, 0)$, B_{10}, $(\mathsf{up}_{10}, 0)$, where

- the input set is $M = [\![1, 11]\!] \times 2^{[\![0,7]\!] \times \mathbb{Z}_8} \times \mathbb{Z}_{128}$ where $(r, C, x) \in M$ models the round number r, intermediary store x, and $(i, y) \in C$ defines an entry of T_i in the cache;
- $\alpha(k, (\_, \_, m)) = (1, \emptyset, k \oplus m)$ initialises the store;
- B is the sequence of 16 blocks performing the memory accesses of a round r, named (f^0_0, t^0_0), (f^4_0, t^4_0), (f^8_0, t^8_0), (f^{12}_0, t^{12}_0), (f^5_1, t^5_1), (f^9_1, t^9_1), …
- a memory access to $T_i[x_j]$ is modelled by
  - $\mathsf{up}(k, (r, C, x)) = (r + 1, C, k^{(r)} \oplus \bigoplus_{\ell=0}^{3} T_\ell[L_\ell(x)])$
  - $f^j_i(k, (r, C, x)) = (r, C \cup \{(i, \langle x_j \rangle)\}, x)$
  - $t^j_i(k, (r, C, x)) = \begin{cases} 0 & \text{if } (i, \langle x_j \rangle) \in C \\ 1 & \text{otherwise} \end{cases}$
- B_{10} and up_{10} are analogous to B and up for last round.