PROOFS workshop: pre-proceedings

"Security Proofs for Embedded Systems"

Leuven, Belgium
Thursday, September 13rd, 2012

Preface

On behalf of the steering committee, we are glad to welcome you to this first edition of PROOFS.

The goal of the PROOFS workshop is to promote methodologies that increase the confidence level in the security of embedded systems.

Embedded system security too frequently consists in security by obscurity solutions (except, of course, for high-security solutions produced by specialized firms, for instance in the smartcard industry). This has obvious drawbacks:

- it requires costly black-box evaluation,
- there is no certainty about the correctness of the security, etc.

Formal methods allow to increase the trust level of digital systems, especially those that embed cryptography. They are very appealing, for the following reasons:

- they are mature in theory, and there are off-the-shelf tried and tested methods and tools,
- they have been applied both on software and hardware for a long time, mainly for safety and conformance tests, but also sometimes for security assessment.

Some important security features (random number generation, physically unclonable functions, side-channel resistance, etc.) rely on analog devices. Their correct functioning can be ascertained by techniques such as physical modeling and unitary experimental testing. But in general, physical models are better evaluated by mathematical methods, which encompass "formal methods".

An important objective for the PROOFS workshop is to bridge the gap between both topics, and therefore to pave the way to "security by clarity" in the design and the evaluation of embedded systems.

Forewords

For this first edition, the PROOFS workshop will be held on one day, with a program that includes:

- three contributed talks,
- three invited talks,
- two short talks in the "WiP" (work in progress) sessions,
- one round-table.

Enjoy the workshop!

Venue of the PROOFS'2012 workshop: College de Valk.

Acknowledgements: We are grateful to TELECOM-ParisTech for the generous sponsorship of the workshop, and K.U.Leuven for the perfect help in its organization. Personal thanks to Jean-Luc Danger for the logo and more ..., and to Svetla Nikova for her care to make PROOFS workshop an enjoyable day!

The programme committee (PC, listed at page 111) reviewed the papers, using the easychair conference management system. Each submission has been evaluated by three PC members. The submissions that involved at least one PC member as co-author have been evaluated by four PC members (of course excluding those who cosigned the submission).

Up-to-date information can be found on the workshop permanent web site:

`http://www.proofs-workshop.org/`

Contents

Program of PROOFS

Leuven, Belgium. Thursday September 13rd, 2012

8h15–9h00	Registration, at College de Valk
9h00–9h15	Opening – Welcome, presentation of PROOFS
9h15–10h15	Invited talk #1 Chair: Stefan Mangard.
• "**Understanding the reasons for the side-channel leakage is indispensable for secure design**", by *Werner Schindler*.	
10h15–10h30	Coffee break
10h30–12h00	Submitted papers session Chair: Svetla Nikova.
• Contributed talk #1, "**A formal study of two physical countermeasures against side channel attacks**", by *Sébastien Briais, Sylvain Guilley and Jean-Luc Danger*.	
• Contributed talk #2, "**Formal verification of an implementation of CRT-RSA Vigilant's algorithm**", by *Maria Christofi, Boutheina Chetali, Louis Goubin and David Vigilant*.	
• Contributed talk #3, "**Toward A Taxonomy of Communications Security Models**", by *Mark Brown*.	
12h00–13h30	Lunch, at Alma cafeteria
13h30–14h30	Invited talk #2 Chair: Éliane Jaulmes.
• "**Toward Formal Design of Cryptographic Processors Based on Galois Field Arithmetic**", by *Naofumi Homma*.	
14h30–15h30	Invited talk #3 Chair: Louis Goubin.
• "**Analysing Cryptographic Hardware Interfaces with Tookan**", by *Graham Steel*.	
15h30–16h00	Coffee break
16h00–16h30	Round-table and Q&A with the audience
16h30–16h35	Wrap-up

A formal study of two physical countermeasures against side channel attacks

Sébastien Briais[1], Sylvain Guilley[2], and Jean-Luc Danger[2]

[1] Secure-IC
sebastien.briais@secure-ic.com
[2] Telecom Paristech
sylvain.guilley@telecom-paristech.fr,
jean-luc.danger@telecom-paristech.fr

Abstract. Secure electronic circuits must implement countermeasures against a wide range of attacks. Often, the protection against side channel attacks requires to be tightly integrated within the functionality to be protected. It is now part of the designer's job to implement them. But this task is known to be error-prone, and with current development processes, countermeasures are evaluated often very late (at circuit fabrication).
In order to improve the confidence of the designer in the efficiency of the countermeasure, we suggest in this article to resort to formal methods early in the design flow for two reasons. First of all, we intend to check that the process of transformation of the design from the vulnerable description to the protected one does not alter the functionality. Second, we wish to prove that the security properties (that can derive from a formal security functional specification) are indeed met after transformation. Our first contribution is to show how such a framework can be setup (in COQ) for netlist-level protections. The second contribution is to illustrate that this framework indeed allows to detect vulnerabilities in dual-rail logics.

1 Introduction

More and more electronic circuits are entrusted with security functions. In particular, they must make sure the information they process, that can be sensitive, is well kept secret. For this reason, electronic circuits must be prepared to be attacked. Thus, it is important that they are properly protected against a wide range of attacks, in particular against side-channel attacks. In practice, to thwart those attacks, extra logic is required: its role is to mask the sensitive data or balance the leakage. From a design point of view, the countermeasure is either coded manually, or implemented automatically by a tool.

In both cases, it would be relevant to ascertain that the functionality remains unchanged after the application of the countermeasure, and that the countermeasure is implemented as intended. But currently, these verifications are seldom carried out: mostly, real attacks are tried after the product is produced, without further formal investigations of the countermeasure after it is applied. The purpose of this paper is to illustrate that the application of a countermeasure can

be formally verified. All the modelisations and proofs of lemmas given in this paper have been obtained in the COQ [4] formal proof assistant.

Some efforts have already been led in order to formally study electronic circuits and their correctness [10, 5, 2]. The present article differs from these previous works in the sense that its objective is not only to study functional correctness of circuits but also to study security properties of hardware countermeasures.

The rest of the paper is structured as follows. In Sec. 2, the studied countermeasures are presented, and described informally. In Sec. 3, a framework to reason on combinational circuits is detailed. Some convenient circuits are defined in appendix (Sec. A). The application of these tools to the formal description of a single-to-dual-rail transformation is carried out in Sec. 4. It allows to show weaknesses present in WDDL but absent from BCDL. Finally, conclusions are given in Sec. 5. This last section also extends the presented methodology to other kinds of dual-rail circuits.

2 Dual-rail Precharge Logic

2.1 Overview

Dual-rail Precharge Logic (DPL) is a class of logic-level countermeasures. It aims at making the device activity constant and independent of the data being processed. In this logic style, a signal is represented by a pair of wires, hence the *dual-rail* qualifier. A cycle of computation is composed of two phases: (1) a *precharge* phase where each pair of wires is discharged by propagating the NULL value through the combinational part of circuit and (2) an *evaluation* phase where the data is processed by the combinational part of the circuit and in which exactly one wire among each pair toggles its state, depending on the logical value the dual-rail conveys. This protocol is depicted in Fig. 1.

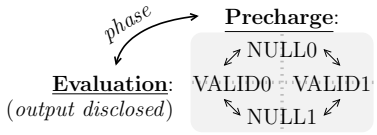


Fig. 1. Separable dual-rail encoding with precharge, yielding a constant activity

Several DPL have been invented along the years: WDDL [13], MDPL [11], DRSL [3], STTL [12], BCDL [8] and SDDL [7] to cite a few.

Each of these proposals succeeds in making the device activity constant. Nonetheless, they differ on some implementation-level aspects. We focus in the sequel on the specific characteristics of WDDL and BCDL.

2.2 Two examples

In this section, an informal description of WDDL (Wave Dynamic Differential Logic) and Balanced Cell-based Differential Logic (BCDL) is provided.

WDDL consists in a separable logic to implement the *true* and *false* halves. Glitches are partial transitions of the nets that are due to races between signals. It is known that they can be responsible for data-dependent leakage [9]. To avoid glitches, WDDL focuses on positive gates. Thus, only AND and OR primitives are used.

Now, AND and OR functions have "short-cut" evaluation. If one input is one, the AND gate has to wait for the second input before evaluating, whereas the OR gate can evaluate one at once. This effect can happen in both evaluation and precharge phases, and cause data-dependent toggling date. During one phase (evaluation or precharge), the activity is constant, but decomposes into events that are data-dependent within the phase (due to small delays between the signals).

BCDL ensures the data-independence of the gates toggling date, thanks to a synchronisation of the inputs at evaluation. The precharge has the functionality to reset all the nodes. Thus, it can be always anticipated, which is implemented by a global signal. This way, BCDL fixes the early propagation effect, and also exhibits no glitches, since by construction a BCDL gate is evaluated only once.

2.3 Vulnerabilities

In DPL styles, the registers can be balanced easily, because they merely implement the "identity function". One way to balance them is to be especially careful at the place-and-route stage. For instance, both in ASIC and FPGA technologies, it is possible to set placement constraints on the two dual registers. If placement constraints are not enough (for instance because the routing would also deserve a similar symmetry), another solution consists in applying a dedicated countermeasures built on top of DPL circuits. A strategy such as "path switching" [1] can be easily applied to the registers. It consist in saving the *true* (resp. *false*) variable in either the *true* (resp. *false*) or the *false* (resp. *true*) register half, depending on a random variable. The functionality remains unchanged, but the leakage is balanced at the first order).

Therefore, in the sequel, we focus on the leakage generated by the combinational logic. These gates are more delicate to perfectly balance, and are more susceptible to cause two major flaws previously mentioned: data dependent glitches and early propagation.

3 Combinational circuits

In this section, we define a formalism that allows to reason about combinational circuits.

3.1 Syntax

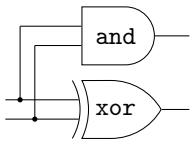
Intuitively, a combinational circuit is a directed acyclic graph whose nodes represent logical gates, and whose edges, which represent wires, are ordered. Despite the fact that this definition is perfectly rigorous from a mathematical point of view, it already involves some heavy mathematical notions which will not ease further reasoning about these objects. So, rather than using this rough definition, we define the combinational circuits as being terms of a process algebra that we define below.

Definition 1 (Combinational circuits). *Let $\mathcal{G}$ be a set of logical gates. The set of* combinational circuits *over $\mathcal{G}$ is defined inductively by (1) the empty circuit $\mathbf{0}$ is a combinational circuit, (2) any gate $g \in \mathcal{G}$ is a combinational circuit, (3) a single wire $\mathbf{I}$ is a combinational circuit, (4) a fork $\mathbf{Y}$, which duplicates a single wire in two wires, is a combinational circuit, (5) a swap $\mathbf{X}$ of two wires is a combinational circuit, and if P and Q are two combinational circuits then so are (6) their parallel composition $P \,|\, Q$ and (7) their sequential composition $P \,;\, Q$.*

Note that by definition, a combinational circuit does not contain any loop.

Regarding the wiring primitives, we have made the choice of minimality. Thus, we have considered a single wire $\mathbf{I}$ instead of a ribbon cable, a single swap $\mathbf{X}$ instead of a generalised permutation operator, and a simple fork $\mathbf{Y}$ instead of a generalised fork that would have replicated a single wire multiple times. This choice, which has no impact on the expressiveness of the calculus (since the generalised operators can easily be defined in terms of these simple ones), allows us to easily define transformations on circuits. Some useful circuits that rotate wires, that interleave or deinterleave wires, or that duplicate wires are thus defined in Section A.

Example 1. We assume that the set of gates contains a `xor` gate, and an `and` gate, i.e. we assume that $\{\texttt{xor}, \texttt{and}\} \subseteq \mathcal{G}$. The half-adder depicted below is represented by the term $\textsc{Half} := (\mathbf{Y} \,|\, \mathbf{Y}) \,;\, (\mathbf{I} \,|\, \mathbf{X} \,|\, \mathbf{I}) \,;\, (\texttt{and} \,|\, \texttt{xor})$.



As a convenient circuit, we define, for $n \in \mathbb{N}$, $\mathbf{I}^n$ which intuitively represents a straight ribbon cable composed of n wires as $\mathbf{I}^0 := \mathbf{0}$ and $\mathbf{I}^{n+1} := \mathbf{I} \,|\, \mathbf{I}^n$.

In the sequel, we let $\mathcal{G}$ be a fixed set of gates and we consider combinational circuits over $\mathcal{G}$, unless stated otherwise.

3.2 Well-formedness

By definition, a combinational circuit does not contain any loop. However, some circuits might be ill-formed. This can happen when composing sequentially two

circuits P and Q if the *number of outputs* of P is different from the *number of inputs* of Q. To exclude these ill-formed circuits, we define a simple type system on combinational circuits. For this, we assume that each gate $g \in \mathcal{G}$ has a type (m_g, n_g) where m_g is the *fan-in* (number of inputs) of the gate g and n_g is the *fan-out* (number of outputs) of the gate g. In other words, we assume that we have a typing function $\mathcal{T} : \mathcal{G} \to \mathbb{N} \times \mathbb{N}$.

Definition 2 (well-formed circuits). *Let P be a combinational circuits over $\mathcal{G}$. We say that P is* well-formed *and* has m inputs and n outputs *— written $P : m \otimes n$ — if and only if there exists a typing derivation using the rules of the type system given below. Otherwise, P is said to be* ill-formed.

$$\frac{\mathcal{T}(g) = (m, n)}{g : m \otimes n}\ g \in \mathcal{G} \qquad \frac{}{\mathbf{0} : 0 \otimes 0} \qquad \frac{}{\mathbf{I} : 1 \otimes 1} \qquad \frac{}{\mathbf{Y} : 1 \otimes 2} \qquad \frac{}{\mathbf{X} : 2 \otimes 2}$$

$$\frac{P_1 : m_1 \otimes n_1 \qquad P_2 : m_2 \otimes n_2}{P_1 \,|\, P_2 : m_1 + m_2 \otimes n_1 + n_2} \qquad \frac{P_1 : m \otimes n \qquad P_2 : n \otimes p}{P_1 \,;\, P_2 : m \otimes p}$$

We comment briefly the rule for sequential composition as it is the source of potential ill-formedness. This rule states that for $P\,;\,Q$ to be well-formed, we must have that (1) P is well-formed, (2) Q is well-formed and (3) the number of outputs of P is equal to the number of inputs of Q.

Example 2. Continuing Example 1, we assume that the and gate `and` and the xor gate `xor` have 2 inputs and 1 output, i.e. that $\mathcal{T}(\texttt{and}) = (2, 1)$ and $\mathcal{T}(\texttt{xor}) = (2, 1)$. Then HALF is well-formed and has 2 inputs and 2 outputs, i.e. HALF : $2 \otimes 2$.

Lemma 1 (uniqueness of type). *Let P be a circuit. If $P : n \otimes m$ and $P : n' \otimes m'$ then $n = n'$ and $m = m'$.*

3.3 Semantics

We interpret combinational circuits by partial functions on words over an *alphabet* Σ. Before defining the formal semantics of circuits, we recall briefly some definitions about languages to fix terminology and notations.

An *alphabet* is a finite set, whose elements are called *letters*. A word u over an alphabet Σ is a finite sequence of letters $u = u_1 \cdot \ldots \cdot u_n$ where $u_i \in \Sigma$ for any i. We note Σ^* the set of words over Σ. If $u = u_1 \cdot \ldots \cdot u_n$ is a word over Σ, we note $|u| = n$ its *length*. The set of words of length $n \in \mathbb{N}$ is written Σ^n. We note ϵ the *empty* word, i.e. the unique word of length 0. If $u = u_1 \cdot \ldots \cdot u_n \in \Sigma^*$ and $v = v_1 \cdot \ldots \cdot v_p \in \Sigma^*$, the *concatenation* $u \bullet v$ of u and v is defined by $u \bullet v = u_1 \cdot \ldots \cdot u_n \cdot v_1 \cdot \ldots \cdot v_p$ of length $|u \bullet v| = |u| + |v|$. A language L over Σ is a subset of Σ^*. If $L_1, L_2 \subseteq \Sigma^*$, we let $L_1 \bullet L_2 := \{u \bullet v \mid u \in L_1 \wedge v \in L_2\}$. If $L \subseteq \Sigma^*$ and $n \in \mathbb{N}$, we define L^n as $L^0 := \{\epsilon\}$ and $L^{n+1} := L \bullet L^n$. Finally, we define the Kleene closure of L to be $L^* := \bigcup_{i \in \mathbb{N}} L^i$.

In the following, we let Σ be an alphabet. By abuse of notations, we will identify each letter $a \in \Sigma$ with the word $a \in \Sigma^*$ of length 1.

In order to define the semantics of circuits, we assume that each gate $g \in \mathcal{G}$ is interpreted by a partial function $\mathcal{E}(g) : \Sigma^* \rightharpoonup \Sigma^*$, which is defined consistently with respect to the type of g, i.e. if $\mathcal{T}(g) = (m, n)$ then the definition domain of $\mathcal{E}(g)$ is Σ^m and its image is included in Σ^n.

Definition 3. *Let P be a combinational circuit over $\mathcal{G}$ and $x, y \in \Sigma^*$. We say that P computes y on x — written $P \Vdash x \rightsquigarrow y$ — if and only if there exists a derivation of $P \Vdash x \rightsquigarrow y$ according to the following inductive rules.*

$$\frac{x \in \Sigma^* \qquad \mathcal{E}(g)(x) = y \in \Sigma^*}{g \Vdash x \rightsquigarrow y}\ g \in \mathcal{G} \qquad \frac{}{\mathbf{0} \Vdash \epsilon \rightsquigarrow \epsilon} \qquad \frac{}{\mathbf{I} \Vdash a \rightsquigarrow a}\ a \in \Sigma$$

$$\frac{}{\mathbf{Y} \Vdash a \rightsquigarrow aa}\ a \in \Sigma \qquad \frac{}{\mathbf{X} \Vdash ab \rightsquigarrow ba}\ a, b \in \Sigma$$

$$\frac{P_1 \Vdash x_1 \rightsquigarrow y_1 \qquad P_2 \Vdash x_2 \rightsquigarrow y_2}{P_1 \parallel P_2 \Vdash x_1 \bullet x_2 \rightsquigarrow y_1 \bullet y_2} \qquad \frac{P_1 \Vdash x \rightsquigarrow y \qquad P_2 \Vdash y \rightsquigarrow z}{P_1 \,;\, P_2 \Vdash x \rightsquigarrow z}$$

The next lemma summarises some important results about the semantics of circuits.

Lemma 2. *Let P be a combinational circuit, $x, y, z \in \Sigma^*$ and $m, n \in \mathbb{N}$.*

- *Computation is deterministic.*
  If $P \Vdash x \rightsquigarrow y$ and $P \Vdash x \rightsquigarrow z$ then $y = z$.
- *Existence of a computation implies well-formedness.*
  If $P \Vdash x \rightsquigarrow y$ then $P : |x| \otimes |y|$.
  As a consequence, we have that if $P \Vdash x \rightsquigarrow y$ and $P : m \otimes n$ then $|x| = m$ and $|y| = n$.
- *A well-formed circuit with m inputs and n outputs computes over Σ^m.*
  If $P : m \otimes n$ and $|x| = m$ then there exists y such that $P \Vdash x \rightsquigarrow y$.
  According to the previous results, we thus have that the definition domain of a well-formed circuit with m inputs and n outputs is Σ^m and its image is included in Σ^n.

3.4 Functional equivalence, structural congruence

Functional equivalence Intuitively, functional equivalence relates any two circuits which compute the same function. It is formally defined below.

Definition 4 (functional equivalence). *Two circuits P and Q are* functionally equivalent*, written $P \simeq Q$, if and only if*

$$\forall x, y \in \Sigma^* : P \Vdash x \rightsquigarrow y \iff Q \Vdash x \rightsquigarrow y$$

An important result, that allows compositional reasoning, is that functional equivalence is a *congruence*. We formally state this result below.

Definition 5 (contexts, congruence). *A* context $C[\,]$ *is a circuit with a* hole $[\,]$ *inside. Formally, the syntax of contexts is given below.*

$$C[\,] ::= [\,] \mid C[\,] \,|\, Q \mid P \,|\, C[\,] \mid C[\,] \,;\, Q \mid P \,;\, C[\,]$$

If P is a circuit and $C[\,]$ is a context, we write $C[P]$ the circuit obtained by syntactically replacing the hole in $C[\,]$ with P.

A congruence $\mathcal{R}$ *is an equivalence relation over $\mathcal{C}_{\mathcal{G}}$ that is preserved by every context, i.e. such that whenever $P\mathcal{R}Q$ then for any context $C[\,]$, we have $C[P]\mathcal{R}C[Q]$.*

Theorem 1. *$\simeq$ is a congruence.*

Another property of $\simeq$ is that it identifies all the ill-formed circuits. Formally, if P and Q are ill-formed, then $P \simeq Q$.

Structural congruence Intuitively, structural congruence identifies circuits that only differ in some minor wiring details.

Definition 6. *The* structural congruence $\equiv$ *is the smallest congruence that satisfies the following axioms:*

1. *for any P, Q and R, $(P \,|\, Q) \,|\, R \equiv P \,|\, (Q \,|\, R)$*
2. *for any P, $P \,|\, \mathbf{0} \equiv \mathbf{0} \,|\, P \equiv P$*
3. *for any P, Q and R, $(P \,;\, Q) \,;\, R \equiv P \,;\, (Q \,;\, R)$*
4. *for any P, if $P : n \otimes m$ then $\mathbf{I}^n \,;\, P \equiv P \,;\, \mathbf{I}^m \equiv P$*
5. *for any P, Q, R and S, if $P : n \otimes m$ and $Q : m \otimes p$ then $(P \,;\, Q) \,|\, (R \,;\, S) \equiv (P \,|\, R) \,;\, (Q \,|\, S)$*
6. $\mathbf{Y} \,;\, (\mathbf{I} \,|\, \mathbf{Y}) \equiv \mathbf{Y} \,;\, (\mathbf{Y} \,|\, \mathbf{I})$
7. $\mathbf{Y} \,;\, \mathbf{X} \equiv \mathbf{Y}$
8. $\mathbf{X} \,;\, \mathbf{X} \equiv \mathbf{I} \,|\, \mathbf{I}$
9. $\mathbf{X} \,;\, (\mathbf{Y} \,|\, \mathbf{Y}) \equiv (\mathbf{Y} \,|\, \mathbf{Y}) \,;\, (\mathbf{I} \,|\, \mathbf{X} \,|\, \mathbf{I}) \,;\, (\mathbf{X} \,|\, \mathbf{X}) \,;\, (\mathbf{I} \,|\, \mathbf{X} \,|\, \mathbf{I})$

Structural congruence preserves the well-formedness. Formally, if $P \equiv Q$ then for any m and n, we have $P : m \otimes n \iff Q : m \otimes n$.

Structural congruence is also a convenient proof method for showing functional equivalence as stated in the following theorem.

Theorem 2. *We have $\equiv \subseteq \simeq$.*

4 Formalisation of WDDL and BCDL

4.1 Dual-rail Precharge Logic

Before defining formally the WDDL and BCDL transformations, we need some more definitions.

In the following, let $\Sigma = \{0, 1\}$. The set of dual-rail words is $\left(\Sigma^2\right)^*$. Let $N := 00$, $T := 10$, $F := 01$ and $E := 11$. Let NULL $:= \{N\}$, VALID $:= \{T, F\}$ and

FAULT$:=\{E\}$. A word $u \in \Sigma^*$ is said to be *null* if and only if $u \in \text{NULL}^*$, to be *valid* if and only if $u \in \text{VALID}^*$, to be *error-free* if and only if $u \in (\text{NULL} \cup \text{VALID})^*$. If $u \in \Sigma^*$, its corresponding value in dual-rail representation is $[u] \in \text{VALID}^*$. It is defined by induction on u by $[\epsilon] := \epsilon$, $[0 \cdot u] := F \bullet [u]$ and $[1 \cdot u] := T \bullet [u]$. Clearly, we have $|[u]| = 2|u|$.

As mentioned before, DPL alternates precharge phase and computation phase. When a switch of phase occurs, input signals acquire their respective values. Thus when switching from precharge to computation, each input signal N becomes either the token T or the token F. To model this, we define on NULL $\cup$ VALID the order $\preceq$ such that for any $x \in \text{NULL} \cup \text{VALID}$ we have $x \preceq x$ and for any $x \in \text{NULL}$ and $y \in \text{VALID}$ we have $x \preceq y$. Note that by construction, the elements of VALID are maximal. Due to routing differences, input signals are likely to acquire their respective logic value at different times. To model this evolution when switching from precharge to computation, we extend the definition of $\preceq$ to error-free words as follows: $u \preceq v$ if and only if there exists $n \in \mathbb{N}$ such that $u = t_1 \cdots t_n$, $v = t'_1 \cdots t'_n$ with $\forall i : t_i, t'_i \in \text{NULL} \cup \text{VALID}$ and for any $1 \leq i \leq n$, $t_i \preceq t'_i$. For example, we have $NN \preceq TN \preceq TF$. By construction, since elements of VALID are maximal, we have that if $x \in \text{VALID}^*$ and $x \preceq y$ then $y = x$.

We define the equivalence relation $\sim$ that equates dual-rail words of same length where each corresponding signals has the same nature: null, valid or faulty. Formally, we let $\sim$ be defined on Σ^2 by $x \sim y$ if and only if both x and y are null (i.e. $x, y \in \text{NULL}$) or both x and y are valid (i.e. $x, y \in \text{VALID}$) or both x and y are faulty (i.e. $x, y \in \text{FAULT}$). In other words, $\sim$ is the equivalence relation on Σ^2 such that its equivalence classes are NULL, VALID and FAULT. We extend this definition to dual-rail words as follows: $u \sim v$ if and only if there exists $n \in \mathbb{N}$ such that $u = t_1 \cdots t_n$, $v = t'_1 \cdots t'_n$ with $\forall i : t_i, t'_i \in \Sigma^2$ and for any $1 \leq i \leq n$, $t_i \sim t'_i$.

To define semantics of circuits, we define the following Boolean operators:

- $\neg$ is the unary operator on Σ such that $\neg x = 1$ if and only if $x = 0$.
- $\wedge$ is the binary operator on $\Sigma \times \Sigma$ such that $x \wedge y = 1$ if and only if $x = y = 1$.
- $\vee$ is the binary operator on $\Sigma \times \Sigma$ such that $x \vee y = 0$ if and only if $x = y = 0$.

4.2 Wave Dynamic Differential Logic (WDDL)

WDDL transformation process is not defined for circuits built with arbitrary logical gates. In the following, we thus assume that the circuits to be secured with WDDL are built over the set $\mathcal{G} = \{\texttt{and}, \texttt{not}\}$. The transformation produces a circuit of $\mathcal{C}_{\mathcal{G}'}$ where $\mathcal{G}' = \{\texttt{and}_{\text{WDDL}}\}$.

We assume the following types: $\mathcal{T}_{\mathcal{G}}(\texttt{and}) = (2, 1)$, $\mathcal{T}_{\mathcal{G}}(\texttt{not}) = (1, 1)$ and $\mathcal{T}_{\mathcal{G}'}(\texttt{and}_{\text{WDDL}}) = (4, 2)$.

We assume that the interpretation functions of these gates are defined by:

- $\mathcal{E}_{\mathcal{G}}(\texttt{and})(a \cdot b) := a \wedge b$ for $a, b \in \Sigma$.
- $\mathcal{E}_{\mathcal{G}}(\texttt{not})(a) := \neg a$ for $a \in \Sigma$.
- $\mathcal{E}_{\mathcal{G}'}(\texttt{and}_{\text{WDDL}})(a_t \cdot a_f \cdot b_t \cdot b_f) := (a_t \wedge b_t) \cdot (a_f \vee b_f)$ for $a_t, a_f, b_t, b_f \in \Sigma$.

We are now ready to define the WDDL securisation process. This process is illustrated on Figure 2.

Definition 7. *We define by induction on* $C \in \mathcal{C}_{\mathcal{G}}$ *the WDDL-secured circuit* $\mathrm{WDDL}(C) \in \mathcal{C}_{\mathcal{G}'}$ *by*

$$\begin{aligned}
\mathrm{WDDL}(\mathbf{0}) &:= \mathbf{0} \\
\mathrm{WDDL}(\mathbf{I}) &:= \mathbf{I} \,|\, \mathbf{I} \\
\mathrm{WDDL}(\mathbf{X}) &:= (\mathbf{I} \,|\, \mathbf{X} \,|\, \mathbf{I}) \,;\, (\mathbf{X} \,|\, \mathbf{X}) \,;\, (\mathbf{I} \,|\, \mathbf{X} \,|\, \mathbf{I}) \\
\mathrm{WDDL}(\mathbf{Y}) &:= (\mathbf{Y} \,|\, \mathbf{Y}) \,;\, (\mathbf{I} \,|\, \mathbf{X} \,|\, \mathbf{I}) \\
\mathrm{WDDL}(\texttt{and}) &:= \texttt{and}_{\mathrm{WDDL}} \\
\mathrm{WDDL}(\texttt{not}) &:= \mathbf{X} \\
\mathrm{WDDL}(C_1 \,|\, C_2) &:= \mathrm{WDDL}(C_1) \,|\, \mathrm{WDDL}(C_2) \\
\mathrm{WDDL}(C_1 \,;\, C_2) &:= \mathrm{WDDL}(C_1) \,;\, \mathrm{WDDL}(C_2)
\end{aligned}$$

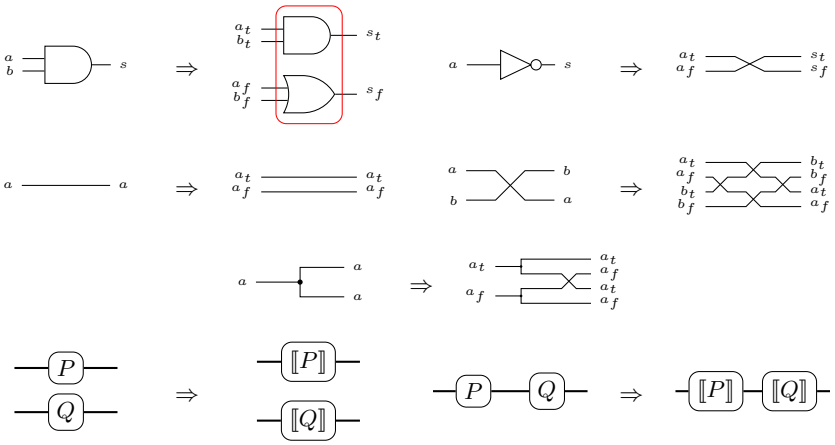


Fig. 2. WDDL securisation process

The WDDL securisation process produces a well-formed circuit if and only if the input circuit is well-formed, as stated below.

Lemma 3. *Let* $C \in \mathcal{C}_{\mathcal{G}}$*. Then*

- *if* $C : n \otimes m$ *then* $\mathrm{WDDL}(C) : 2n \otimes 2m$*.*
- *if* $\mathrm{WDDL}(C) : n' \otimes m'$ *then there exists* n *and* m *such that* $C : n \otimes m$ *and* $n' = 2n$ *and* $m' = 2m$*.*

The following lemma states that a WDDL circuit fulfils the DPL invariants: it propagates the NULL state and the VALID state.

Lemma 4. *Let $C \in \mathcal{C}_\mathcal{G}$ and assume that* $\mathrm{WDDL}(C) \Vdash x \rightsquigarrow y$. *Then*

- *if* $x \in \mathrm{NULL}^*$ *then* $y \in \mathrm{NULL}^*$.
- *if* $x \in \mathrm{VALID}^*$ *then* $y \in \mathrm{VALID}^*$.

We prove with the following lemma that the WDDL securisation process is sound. In other words, a WDDL secured circuit computes at least as the original circuit.

Lemma 5. *Let $C \in \mathcal{C}_\mathcal{G}$. If $C \Vdash x \rightsquigarrow y$ then* $\mathrm{WDDL}(C) \Vdash [x] \rightsquigarrow [y]$.

The next lemma states the converse result: the WDDL securisation process is complete. In other words, a WDDL secured circuit computes no more than the original circuit on valid inputs.

Lemma 6. *Let $C \in \mathcal{C}_\mathcal{G}$. If* $\mathrm{WDDL}(C) \Vdash x' \rightsquigarrow y'$ *and* $x' \in \mathrm{VALID}^*$ *then there exists $x, y \in \Sigma^*$ such that $x' = [x]$, $y' = [y]$ and $C \Vdash x \rightsquigarrow y$.*

4.3 Balanced Cell-based Differential Logic (BCDL)

Contrary to WDDL, BCDL transformation process can be defined for circuits build on top of arbitrary logical gates. We thus let $\mathcal{G}$ be the set of basic gates of the circuits to be protected. We assume to have a typing function $\mathcal{T}_\mathcal{G} : \mathcal{G} \to \mathbb{N} \times \mathbb{N}$ and an evaluation function $\mathcal{E}_\mathcal{G} : \mathcal{G} \to (\Sigma^* \rightharpoonup \Sigma^*)$. The transformation process produces a circuit of $\mathcal{C}_{\mathcal{G}'}$ where $\mathcal{G}' := \{g_{\mathrm{BCDL}} \mid g \in \mathcal{G}\} \cup \{\mathrm{U}_n \mid n \in \mathbb{N}\} \cup \{\mathrm{ANDN}\}$.

We assume the following types for the gates of $\mathcal{G}'$. If $g \in \mathcal{G}$ and $\mathcal{T}_\mathcal{G}(g) = (n, m)$ then $\mathcal{T}_{\mathcal{G}'}(g_{\mathrm{BCDL}}) = (n+1, 2m)$. The gate g_{BCDL} corresponding to g has an extra input that indicates whether evaluation is enabled or not and produces a dual-rail result, thus the $2m$ outputs. We also assume $\mathcal{T}_{\mathcal{G}'}(\mathrm{ANDN}) = (2, 1)$ and for $n \in \mathbb{N}$, $\mathcal{T}_{\mathcal{G}'}(\mathrm{U}_n) = (2n, 1)$.

Regarding the interpretation function, we assume that:

- $\mathcal{E}_{\mathcal{G}'}(\mathrm{ANDN})(a \cdot b) := (\neg a) \wedge b$ for $a, b \in \Sigma$.
- for $n \in \mathbb{N}$, and $x \in \Sigma^{2n}$, $\mathcal{E}_{\mathcal{G}'}(\mathrm{U}_n)(x) := 1$ if $x \in \mathrm{VALID}^n$ and $\mathcal{E}_{\mathcal{G}'}(\mathrm{U}_n)(x) := 0$ otherwise.
- for $g \in \mathcal{G}$, if $\mathcal{T}_\mathcal{G}(g) = (n, m)$ then for $x \in \Sigma^n$, $\mathcal{E}_{\mathcal{G}'}(g_{\mathrm{BCDL}})(0 \cdot x) := 0^{2m}$ and $\mathcal{E}_{\mathcal{G}'}(g_{\mathrm{BCDL}})(1 \cdot x) := [\mathcal{E}(g)(x)]$.

Before defining the BCDL securisation process on whole circuits, we focus on how a simple gate $g \in \mathcal{G}$ is secured. The idea of BCDL is that evaluation is enabled only once every dual-rail input signal becomes valid and when global precharge signal is low. Figure 3 shows how to achieve this. An unanimity gate U_n (circled on the figure) verifies that every dual-rail signal is valid and transmits the result to a gate ANDN which ands this signal with the negation of the precharge signal. The result is then transmitted to the dual-rail gate g_{BCDL} corresponding to g, which uses this signal to enable evaluation. Formally, for $g \in \mathcal{G}$, we define $C_g := (\mathbf{I} \,|\, (\mathrm{unint}_n \,;\, (\mathrm{dup}_n \,|\, \mathbf{I}^n) \,;\, (\mathbf{I}^n \,|\, (\mathrm{int}_n \,;\, \mathrm{U}_n)) \,;\, \mathrm{ror}_{n+1})) \,;\, (\mathrm{ANDN} \,|\, \mathbf{I}^n) \,;\, g_{\mathrm{BCDL}}$.

The BCDL securisation process is defined thereafter. One difficulty that arises when defining BCDL transformation process is the fact that a circuit only made

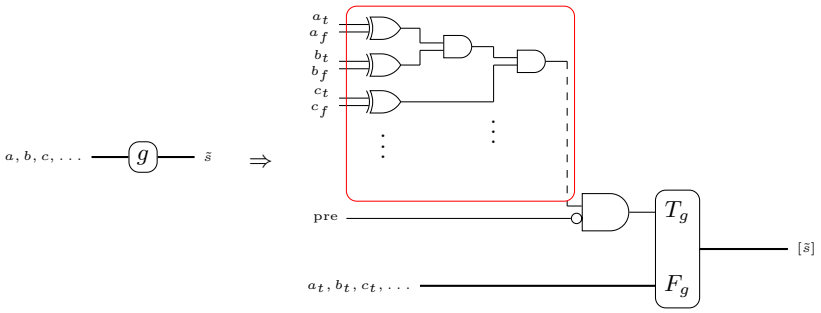


Fig. 3. Securing an arbitrary gate with BCDL

of wires (with no gates) does not need a global precharge signal. For this reason, the BCDL transformation function returns a triple (b, n, C) where $b \in \{\mathrm{tt}, \mathrm{ff}\}$ is a boolean, $n \in \mathbb{N}$ is an integer and C a BCDL-secured circuit. The boolean b indicates whether C has a global precharge signal. The integer n corresponds to the number of inputs of the unsecured circuit (while we do not assume it is well-formed).

Definition 8. *We define by induction on $C \in \mathcal{C}_\mathcal{G}$ the BCDL-secured circuit* $\mathrm{BCDL}(C) \in \mathcal{C}_{\mathcal{G}'}$ *by*

$$
\begin{aligned}
\mathrm{BCDL}(\mathbf{0}) &:= (\mathrm{ff}, 0, \mathbf{0}) \\
\mathrm{BCDL}(\mathbf{I}) &:= (\mathrm{ff}, 1, \mathbf{I} \,|\, \mathbf{I}) \\
\mathrm{BCDL}(\mathbf{X}) &:= (\mathrm{ff}, 2, (\mathbf{I} \,|\, \mathbf{X} \,|\, \mathbf{I}) \,;\, (\mathbf{X} \,|\, \mathbf{X}) \,;\, (\mathbf{I} \,|\, \mathbf{X} \,|\, \mathbf{I})) \\
\mathrm{BCDL}(\mathbf{Y}) &:= (\mathrm{ff}, 1, (\mathbf{Y} \,|\, \mathbf{Y}) \,;\, (\mathbf{I} \,|\, \mathbf{X} \,|\, \mathbf{I})) \\
\mathrm{BCDL}(g \in \mathcal{G}) &:= (\mathrm{tt}, n, C_g) && \text{if } \mathcal{T}_\mathcal{G}(g) = (n, m) \\
\mathrm{BCDL}(C_1 \,;\, C_2) &:= (\mathrm{tt}, n_1, (\mathbf{I} \,|\, C_1') \,;\, C_2') && \text{if } \mathrm{BCDL}(C_1) = (\mathrm{ff}, n_1, C_1') \text{ and } \mathrm{BCDL}(C_2) = (\mathrm{tt}, n_2, C_2') \\
&:= (\mathrm{tt}, n_1, (\mathbf{Y} \,|\, \mathbf{I}^{2n_1}) \,;\, (\mathbf{I} \,|\, C_1') \,;\, C_2') && \text{if } \mathrm{BCDL}(C_1) = (\mathrm{tt}, n_1, C_1') \text{ and } \mathrm{BCDL}(C_2) = (\mathrm{tt}, n_2, C_2') \\
&:= (b, n_1, C_1' \,;\, C_2') && \text{if } \mathrm{BCDL}(C_1) = (b, n_1, C_1') \text{ and } \mathrm{BCDL}(C_2) = (\mathrm{ff}, n_2, C_2') \\
\mathrm{BCDL}(C_1 \,|\, C_2) &:= (\mathrm{tt}, n_1 + n_2, (\mathrm{rol}_{2n_1+1} \,|\, \mathbf{I}^{2n_2}) \,;\, (C_1' \,|\, C_2')) \\
&\qquad\qquad \text{if } \mathrm{BCDL}(C_1) = (\mathrm{ff}, n_1, C_1') \text{ and } \mathrm{BCDL}(C_2) = (\mathrm{tt}, n_2, C_2') \\
&\quad (\mathrm{tt}, n_1 + n_2, (\mathbf{Y} \,|\, \mathbf{I}^{2n_1+2n_2}) \,;\, (\mathbf{I} \,|\, \mathrm{rol}_{2n_1+1} \,|\, \mathbf{I}^{2n_2}) \,;\, (C_1' \,|\, C_2')) \\
&\qquad\qquad \text{if } \mathrm{BCDL}(C_1) = (\mathrm{tt}, n_1, C_1') \text{ and } \mathrm{BCDL}(C_2) = (\mathrm{tt}, n_2, C_2') \\
&:= (b, n_1 + n_2, C_1' \,|\, C_2') \\
&\qquad\qquad \text{if } \mathrm{BCDL}(C_1) = (b, n_1, C_1') \text{ and } \mathrm{BCDL}(C_2) = (\mathrm{ff}, n_2, C_2')
\end{aligned}
$$

In the sequel, let $\delta_{\mathrm{ff}} := 0$ and $\delta_{\mathrm{tt}} := 1$. The following lemma states that BCDL securisation process produces a well-formed circuit if and only if the input circuit is well-formed.

Lemma 7. *Let $C \in \mathcal{C}_\mathcal{G}$ and b, n, C' such that* $\mathrm{BCDL}(C) = (b, n, C')$. *Then*

- *if $C : n' \otimes m$ then $n = n'$ and $C' : 2n + \delta_b \otimes 2m$.*
- *if $C' : n' \otimes m'$ then there exists m such that $C : n \otimes m$, $n' = 2n + \delta_b$ and $m' = 2m$.*

The next lemma states a result similar to that of Lemma 4: a BCDL circuit propagates the NULL state when the precharge signal is high and propagates the VALID state when the precharge signal is low.

Lemma 8. *Let* $C \in \mathcal{C}_{\mathcal{G}}$ *and* b, n, C' *such that* $\mathrm{BCDL}(C) = (b, n, C')$*. Then*

- *if* $b = \mathrm{tt}$ *and* $\mathrm{BCDL}(C) \Vdash 1 \cdot x \rightsquigarrow y$ *then*
  - *if* $x \in \mathrm{NULL}^*$ *then* $y \in \mathrm{NULL}^*$.
  - *if* $x \in \mathrm{VALID}^*$ *then* $y \in \mathrm{VALID}^*$.
- *if* $b = \mathrm{ff}$ *and* $\mathrm{BCDL}(C) \Vdash x \rightsquigarrow y$ *then*
  - *if* $x \in \mathrm{NULL}^*$ *then* $y \in \mathrm{NULL}^*$.
  - *if* $x \in \mathrm{VALID}^*$ *then* $y \in \mathrm{VALID}^*$.

Observe in the previous lemma that precharge signal is the *first* input, when the BCDL secured circuit routes such a signal (i.e. when $b = \mathrm{tt}$).

We can refine this result on circuits C_g where $g \in \mathcal{G}$ as shown in the next lemma. Hence, when precharge signal is high, C_g produces null. It also produces null when input is not valid, whatever the state of the precharge signal is.

Lemma 9. *Let* $g \in \mathcal{G}$*. Then*

- *if* $C_g \Vdash 1 \cdot x \rightsquigarrow y$ *then* $y \in \mathrm{NULL}^*$.
- *if* $C_g \Vdash p \cdot x \rightsquigarrow y$ *and* $x \notin \mathrm{VALID}^*$ *then* $y \in \mathrm{NULL}^*$.

The next lemma states that a BCDL secured circuit computes at least as the original circuit. This is a result analogous to Lemma 5.

Lemma 10. *Let* $C \in \mathcal{C}_{\mathcal{G}}$ *and* b, n, C' *such that* $\mathrm{BCDL}(C) = (b, n, C')$*. Then*

- *if* $b = \mathrm{tt}$ *and* $C \Vdash x \rightsquigarrow y$ *then* $C' \Vdash 0 \cdot [x] \rightsquigarrow [y]$.
- *if* $b = \mathrm{ff}$ *and* $C \Vdash x \rightsquigarrow y$ *then* $C' \Vdash [x] \rightsquigarrow [y]$.

The converse result is true, as it was the case for WDDL (see Lemma 6). However, note that in the case of BCDL, it is only sufficient to assume that the output is valid (and not the input) as long as the original circuit does not contain a gate g with no outputs.

Lemma 11. *Let* $C \in \mathcal{C}_{\mathcal{G}}$ *such that it does not contain a gate* g *with no outputs and let* b, n, C' *such that* $\mathrm{BCDL}(C) = (b, n, C')$*. Then*

- *if* $b = \mathrm{tt}$*,* $C' \Vdash 0 \cdot x' \rightsquigarrow y'$ *and* $y' \in \mathrm{VALID}^*$ *then there exists* x, y *such that* $C \Vdash x \rightsquigarrow y$ *and* $x' = [x]$ *and* $y' = [y]$.
- *if* $b = \mathrm{ff}$*,* $C' \Vdash x' \rightsquigarrow y'$ *and* $y' \in \mathrm{VALID}^*$ *then there exists* x, y *such that* $C \Vdash x \rightsquigarrow y$ *and* $x' = [x]$ *and* $y' = [y]$.

4.4 Security properties

In this section, we will illustrate how our formalism allows us to tackle some security properties such as glitches or early-evaluation.

Glitches An electronic glitch is an undesired transition that occurs before the signal settles to its intended value. In DPL, the precharge and the evaluation phase alternate. Thus, when switching from precharge to evaluation, input signals progressively change their value from NULL to VALID. And conversely, when switching from evaluation to precharge, input signals progressively change their value from VALID to NULL. This change of state is precisely modeled by the partial order $\preceq$ we introduced previously (see Section 4.1). Indeed, when switching from precharge to evaluation, input signals modeled by a word of $(\Sigma^2)^*$ take different values $x_1, \ldots, x_n$ where $x_1 \in \text{NULL}^*$, $x_n \in \text{VALID}^*$ and for all $1 \leq i < n$, $x_i \preceq x_{i+1}$. For instance, Figure 4 illustrates the transition of an input signals composed of two dual-rails $(a_t, a_f), (b_t, b_f)$ from the precharge phase to the evaluation phase taking values $NN \preceq TN \preceq TF$.

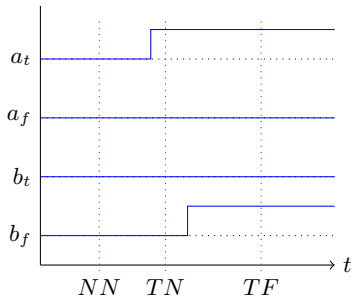


Fig. 4. $\preceq$ models transition of signals from null to valid

The next two lemmas state that WDDL as well as BCDL-secured circuits preserves the partial order $\preceq$.

Lemma 12. *Let $C \in \mathcal{C}_\mathcal{G}$.*
If $\text{WDDL}(C) \Vdash x \rightsquigarrow y$, $\text{WDDL}(C) \Vdash x' \rightsquigarrow y'$ *and* $x \preceq x'$, *then* $y \preceq y'$.

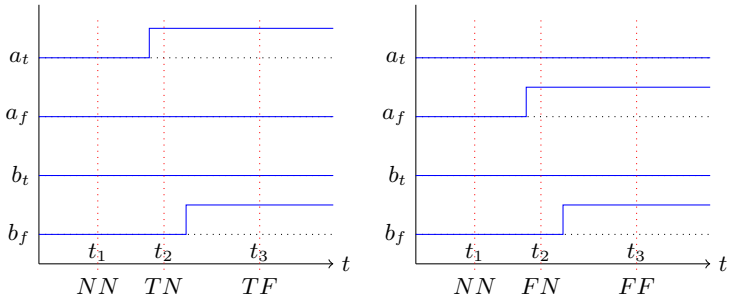
Lemma 13. *Let $C \in \mathcal{C}_\mathcal{G}$ and b, n, C' such that* $\text{BCDL}(C) = (b, n, C')$. *Then*

- *if $b = \text{tt}$ then for any $p \in \Sigma$, if $C' \Vdash p \cdot x \rightsquigarrow p \cdot y$, $C' \Vdash p \cdot x' \rightsquigarrow p \cdot y'$ and $x \preceq x'$, then $y \preceq y'$.*
- *if $b = \text{ff}$ then if $C' \Vdash x \rightsquigarrow y$, $C' \Vdash x' \rightsquigarrow y'$ and $x \preceq x'$, then $y \preceq y'$.*

The fact that WDDL and BCDL circuits preserve the partial order $\preceq$ intuitively means that when input signals acquire progressively their values then output signals also acquire progressively their values. By construction of $\preceq$, this means that once a dual-rail output signal has taken its value (in VALID), it won't change its value until the next switch of phase. This precisely means that no glitches are possible.

Early-evaluation In order to address the problem of early-evaluation, we compare the behaviour of circuits on equivalent inputs. Indeed, intuitively, a circuit does not suffer from the early-evaluation problem if it produces the same amount of work on equivalent inputs.

We have defined previously an equivalence relation $\sim$ that equate words which have the same amount of information, i.e. in which corresponding dual-rail signals have the same nature. For instance, on Figure 5, the pair of dual-rail signals (a_t, a_f),(b_t, b_f) in scenario #1 and scenario #2 conveys the same amount of information at time t_1 since $NN \sim NN$, at time t_2 since $TN \sim FN$ and at time t_3 since $TF \sim FF$.



(a) Scenario #1 (b) Scenario #2

Fig. 5. $\sim$ equates states where the nature of dual-rail signals is the same

The next lemma states that BCDL-secured circuit preserves the equivalence relation $\sim$.

Lemma 14. *Let $C \in \mathcal{C}_\mathcal{G}$ and b, n, C' such that* $\mathrm{BCDL}(C) = (b, n, C')$. *Then*

- *if $b = \mathrm{tt}$ then for any $p \in \Sigma$, if $C' \Vdash p \cdot x \rightsquigarrow p \cdot y$, $C' \Vdash p \cdot x' \rightsquigarrow p \cdot y'$ and $x \sim x'$, then $y \sim y'$.*
- *if $b = \mathrm{ff}$ then if $C' \Vdash x \rightsquigarrow y$, $C' \Vdash x' \rightsquigarrow y'$ and $x \sim x'$, then $y \sim y'$.*

On the contrary, a similar result for WDDL circuits does not hold. Indeed, consider the WDDL circuit $\mathtt{and}_{\mathrm{WDDL}}$. We have that $\mathtt{and}_{\mathrm{WDDL}} \Vdash FN \rightsquigarrow F$, $\mathtt{and}_{\mathrm{WDDL}} \Vdash TN \rightsquigarrow N$ and $FN \sim TN$. But $F \nsim N$. In other words, WDDL does not preserve $\sim$ as stated below.

Lemma 15. *There exists $C \in \mathcal{C}_\mathcal{G}$ and x, x', y, y' such that* $\mathrm{WDDL}(C) \Vdash x \rightsquigarrow y$, $\mathrm{WDDL}(C) \Vdash x' \rightsquigarrow y'$, *$x \sim x'$ and $y \nsim y'$.*

The problem of early-evaluation can also be seen in another manner. A DPL circuit does not suffer from the early-evaluation problem if it produces valid outputs only on valid inputs. It is insightful to compare Lemma 6 and Lemma 11

with this idea in mind. Indeed, in the case of BCDL, it is true that a BCDL-secured circuit produces valid outputs only on valid inputs (provided it does not contain gates with no outputs). On the contrary, this result does not hold for WDDL-secured circuit. Indeed, consider the circuit $\texttt{and}_{\text{WDDL}}$. Then we have $\texttt{and}_{\text{WDDL}} \Vdash FN \rightsquigarrow F$ and $F \in \text{VALID}^*$. But $FN \notin \text{VALID}^*$.

Measuring activity of circuits Hitherto, we have shown how problems such as glitches and early-evaluation effects can be detected by just looking at the functionality of a circuit. However, this approach is not sufficient because the principle of physical attacks is to look at the details of implementation and not only to abstract the functionality of circuits.

For instance, the circuit of Figure 6 is a WDDL "and" gate, but the result is conditioned by the fact that the input signals are valid. From the functionality point of view, this circuit does not suffer from early-evaluation effect w.r.t. the definition we suggested before: it preserves $\sim$ and it produces valid outputs only on valid inputs. But this circuit is problematic because it still suffers from early-evaluation. Indeed, the "and" and "or" gate computes whatever the nature of the input signals and then the result is accepted or rejected depending on the nature of the input signals. Hence, we can detect different speed of response of the circuit depending on the input data.

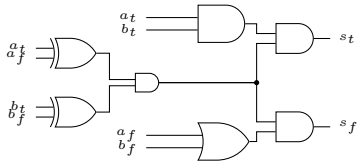


Fig. 6. A WDDL "and" gate with a synchronisation stage

In order to be able to compare different implementations of the same boolean function, we propose to measure activity of circuits. We assume to have a function $\mu^{\mathcal{G}} : \mathcal{G} \to \Sigma^* \to \mathbb{N}$ which measures the activity of each gate $g \in \mathcal{G}$, $\mu^{\mathcal{G}}(g, x)$ being the activity of g on input $x \in \Sigma^*$.

The following inductive rules define the predicate $\mu(C, x, n)$, which relates the activity n produced by a circuit C on input word x.

$$\frac{x \in \Sigma^* \qquad \mathcal{E}(g)(x) \in \Sigma^*}{\mu(g, x, \mu^{\mathcal{G}}(g, x))}\ g \in \mathcal{G} \qquad \frac{}{\mu(\mathbf{0}, \epsilon, 0)} \qquad \frac{}{\mu(\mathbf{I}, a, 0)}\ a \in \Sigma$$

$$\frac{}{\mu(\mathbf{Y}, a, 0)}\ a \in \Sigma \qquad \frac{}{\mu(\mathbf{X}, ab, 0)}\ a, b \in \Sigma \qquad \frac{\mu(P_1, x_1, n_1) \qquad \mu(P_2, x_2, n_2)}{\mu(P_1 \,|\, P_2, x_1 \bullet x_2, n_1 + n_2)}$$

$$\frac{\mu(P_1, x, n) \qquad P_1 \Vdash x \rightsquigarrow y \qquad \mu(P_2, y, m)}{\mu(P_1 \,;\, P_2, x, n + m)}$$

The next lemma gives basic properties of the activity predicate.

Lemma 16. *1. If $\mu(P, x, n)$ and $\mu(P, x, n')$ then $n = n'$.*
2. If $\mu(P, x, n)$ then there exists y such that $P \Vdash x \rightsquigarrow y$.
3. If $P \Vdash x \rightsquigarrow y$ then there exists n such that $\mu(P, x, n)$.

We now study the activity of BCDL-secured circuit. We assume that the activity of the basic gates is such that:

- for any $n \in \mathbb{N}$, for any $x, y \in \Sigma^*$, if $x \sim y$ then $\mu^{\mathcal{G}'}(\mathrm{U}_n, x) = \mu^{\mathcal{G}'}(\mathrm{U}_n, y)$.
- for any $g \in \mathcal{G}$, for any $x, y \in \Sigma^*$, and $s \in \Sigma$, if $|x| = |y|$ then we have $\mu^{\mathcal{G}'}(g_{\mathrm{BCDL}}, s \cdot x) = \mu^{\mathcal{G}'}(g_{\mathrm{BCDL}}, s \cdot y)$.

Then a BCDL-secured circuit has a constant activity on equivalent inputs, as stated in the next lemma.

Lemma 17. *Let $C \in \mathcal{C}_{\mathcal{G}}$ and b, n, C' such that $\mathrm{BCDL}(C) = (b, n, C')$. Then*

- *if $b = \mathrm{tt}$, then for any $p \in \Sigma$, for any $x, x' \in \Sigma^*$ and for any $k, k' \in \mathbb{N}$, if $\mu(C', p \cdot x, k)$, $\mu(C', p \cdot x', k')$ and $x \sim x'$ then $k = k'$.*
- *if $b = \mathrm{ff}$, then for any $x, x' \in \Sigma^*$ and for any $k, k' \in \mathbb{N}$, if $\mu(C', x, k)$, $\mu(C', x', k')$ and $x \sim x'$ then $k = k'$.*

Note that a similar result is not true for the circuit of Figure 6 if we take for $\mu^{\mathcal{G}}(g, x)$ the hamming weight of $\mathcal{E}(g)(x)$. For $(a_t, a_f), (b_t, b_f) = FN$, we would have an activity of 2 (one "xor" gate and one "or" gate react) whereas for $(a_t, a_f), (b_t, b_f) = TN$ we would have an activity of 1 (only one "xor" gate reacts).

5 Conclusions and Perspectives

This article has shown that the transformation from an unprotected to a side channel attack resistant netlist can be captured formally. The scheme described in the article allows to work on any kind of combinational circuit, unprotected or protected. To the authors' best knowledge, it is the first time a circuit-level countermeasure is processed formally. Furthermore, our scheme can be augmented with the verification of some properties. In particular, we formally describe the presence of glitches and of early propagations, properties that were previously only discussed informally or on examples in the embedded secure literature. These properties are tested on two dual-rail logic styles, namely WDDL and BCDL. It is shown that none have glitches, but that WDDL is flawed with early propagation.

As a perspective, we intend to apply these results on other dual-rail styles. For instance, SDDL [7] could be shown to be victim of both glitches and early propagation. Also, some other subtle bugs could be discovered if the inner structure of the logic gates was included in the modeling. Typically, DRSL [3] features a data-dependent glitch happening only internally inside a gate [6]. This flaw cannot be found in the current state of the scheme. In a nutshell, we believe the scope of formal methods can be broadened to detect early some future security troubles related to implementation-level attacks.

References

1. G. Fraidy Bouesse, Gilles Sicard, and Marc Renaudin. Path Swapping Method to Improve DPA Resistance of Quasi Delay Insensitive Asynchronous Circuits. In Louis Goubin and Mitsuru Matsui, editors, *CHES*, volume 4249 of *Lecture Notes in Computer Science*, pages 384–398. Springer, 2006.
2. Thomas Braibant. Coquet: A coq library for verifying hardware. In Jean-Pierre Jouannaud and Zhong Shao, editors, *CPP*, volume 7086 of *Lecture Notes in Computer Science*, pages 330–345. Springer, 2011.
3. Zhimin Chen and Yujie Zhou. Dual-Rail Random Switching Logic: A Countermeasure to Reduce Side Channel Leakage. In *CHES*, volume 4249 of *LNCS*, pages 242–254. Springer, October 10-13 2006. Yokohama, Japan, `http://dx.doi.org/10.1007/11894063_20`.
4. The Coq Development Team. *The Coq Proof Assistant Reference Manual Version 7.2*. INRIA-Rocquencourt, December 2001. http://coq.inria.fr/doc-eng.html.
5. Solange Coupet-Grimal and Line Jakubiec. Certifying circuits in type theory. *Formal Asp. Comput.*, 16(4):352–373, 2004.
6. Jean-Luc Danger, Sylvain Guilley, Shivam Bhasin, and Maxime Nassar. Overview of Dual Rail with Precharge Logic Styles to Thwart Implementation-Level Attacks on Hardware Cryptoprocessors, — *New Attacks and Improved Counter-Measures* —. In *SCS*, IEEE, pages 1–8, November 6–8 2009. Jerba, Tunisia. DOI: 10.1109/ICSCS.2009.5412599.
7. Wei He, Eduardo de la Torre, and Teresa Riesgo. An Interleaved EPE-Immune PA-DPL Structure for Resisting Concentrated EM Side Channel Attacks on FPGA Implementation. In Werner Schindler and Sorin A. Huss, editors, *COSADE*, volume 7275 of *Lecture Notes in Computer Science*, pages 39–53. Springer, 2012.
8. Maxime Nassar, Shivam Bhasin, Jean-Luc Danger, Guillaume Duc, and Sylvain Guilley. BCDL: A high performance balanced DPL with global precharge and without early-evaluation. In *DATE'10*, pages 849–854. IEEE Computer Society, March 8-12 2010. Dresden, Germany.
9. Svetla Nikova, Vincent Rijmen, and Martin Schläffer. Secure Hardware Implementation of Non-linear Functions in the Presence of Glitches. In *ICISC*, volume 5461 of *Lecture Notes in Computer Science*, pages 218–234. Springer, 2008. Seoul, Korea.
10. C. Paulin-Mohring. Circuits as streams in Coq : Verification of a sequential multiplier. In S. Berardi and M. Coppo, editors, *Types for Proofs and Programs, TYPES'95*, volume 1158 of *Lecture Notes in Computer Science*, 1996.
11. Thomas Popp and Stefan Mangard. Masked Dual-Rail Pre-charge Logic: DPA-Resistance Without Routing Constraints. In *CHES*, volume 3659 of *LNCS* , pages 172–186, 2005. `http://dx.doi.org/10.1007/11545262_13`.
12. Rafael Soares, Ney Calazans, Victor Lomné, Philippe Maurine, Lionel Torres, and Michel Robert. Evaluating the robustness of secure triple track logic through prototyping. In *SBCCI'08: Proceedings of the 21st symposium on Integrated circuits and system design*, pages 193–198, Gramado, Brazil, 2008. ACM.
13. K. Tiri and I. Verbauwhede. A logic level design methodology for a secure DPA resistant ASIC or FPGA implementation. In *Proceedings of DATE'2004*, pages 246–251, Paris, France., February 2004.

A Wiring

In this section, we define several circuits that manipulate wires and are convenient when defining more complex transformations.

A.1 Rotations

The *right rotation* $\overrightarrow{u}$ of a word $u \in \Sigma^*$ is ϵ if $u = \epsilon$ and $a \cdot v$ if there exists $a \in \Sigma$ and $v \in \Sigma^*$ such that $u = v \bullet a$. Similarly, the *left rotation* $\overleftarrow{u}$ of a word $u \in \Sigma^*$ is ϵ if $u = \epsilon$ and $v \bullet a$ if there exists $a \in \Sigma$ and $v \in \Sigma^*$ such that $u = a \cdot v$.

Clearly, we have for any word $u \in \Sigma^*$ that $\overleftrightarrow{u} = \overrightarrow{\overleftrightarrow{u}} = u$, i.e. left and right rotation on words are inverse operations. Moreover, rotations preserve length, i.e. for any $u \in \Sigma^*$, $|\overleftarrow{u}| = |\overrightarrow{u}| = |u|$.

We define by induction on $n \in \mathbb{N}$ the circuit ror_n by (1) $\mathrm{ror}_0 := \mathbf{0}$, (2) $\mathrm{ror}_1 := \mathbf{I}$, and (3) $\mathrm{ror}_{n+2} := (\mathbf{I}^n \,|\, \mathbf{X})\,;\, (\mathrm{ror}_{n+1} \,|\, \mathbf{I})$ for $n \in \mathbb{N}$.

The next lemma states that the circuit ror_n implements right rotation of words of length n.

Lemma 18. *Let $n \in \mathbb{N}$. Then*

- $\mathrm{ror}_n : n \otimes n$, *and*
- *for all $x, y \in \Sigma^*$, $\mathrm{ror}_n \Vdash x \rightsquigarrow y$ if and only if $|x| = n$ and $y = \overrightarrow{x}$.*

Similarly, we define by induction on $n \in \mathbb{N}$ the circuit rol_n by (1) $\mathrm{rol}_0 := \mathbf{0}$, (2) $\mathrm{rol}_1 := \mathbf{I}$, and (3) $\mathrm{rol}_{n+2} := (\mathrm{rol}_{n+1} \,|\, \mathbf{I})\,;\, (\mathbf{I}^n \,|\, \mathbf{X})$ for $n \in \mathbb{N}$.

The next lemma states that the circuit rol_n implements left rotation of words of length n.

Lemma 19. *Let $n \in \mathbb{N}$. Then*

- $\mathrm{rol}_n \,;\, \mathrm{ror}_n \equiv \mathrm{ror}_n \,;\, \mathrm{rol}_n \equiv \mathbf{I}^n$,
- $\mathrm{rol}_n : n \otimes n$, *and*
- *for all $x, y \in \Sigma^*$, $\mathrm{rol}_n \Vdash x \rightsquigarrow y$ if and only if $|x| = n$ and $y = \overleftarrow{x}$.*

A.2 Interleaving

The *interleaving* $u \,|||\, v$ of two words $u, v \in \Sigma^*$ of the same length is defined by induction on u and v by (1) $\epsilon \,|||\, \epsilon := \epsilon$, and (2) $(a \cdot u) \,|||\, (b \cdot v) := ab(u \,|||\, v)$.

We define by induction on $n \in \mathbb{N}$ the circuit int_n by (1) $\mathrm{int}_0 := \mathbf{0}$ and (2) $\mathrm{int}_{n+1} := (\mathbf{I} \,|\, \mathrm{ror}_{n+1} \,|\, \mathbf{I}^n)\,;\, (\mathbf{I} \,|\, \mathbf{I} \,|\, \mathrm{int}_n)$.

The next lemma states that the circuit int_n interleaves two ribbons of n wires.

Lemma 20. *Let $n \in \mathbb{N}$. Then*

- $\mathrm{int}_n : 2n \otimes 2n$, *and*

– *for all* $x, y \in \Sigma^*$, $\mathrm{int}_n \Vdash x \rightsquigarrow y$ *if and only if there exists* $u, v \in \Sigma^*$ *such that* $x = u \bullet v$ *and* $|u| = |v| = n$ *and* $y = u \,|||\, v$.

We define by induction on $n \in \mathbb{N}$ the circuit unint_n by (1) $\mathrm{unint}_0 := \mathbf{0}$ and (2) $\mathrm{unint}_{n+1} := (\mathbf{I} \,|\, \mathbf{I} \,|\, \mathrm{unint}_n)\,;\, (\mathbf{I} \,|\, \mathrm{rol}_{n+1} \,|\, \mathbf{I}^n)$.

The next lemma states that the circuit unint_n deinterleaves a ribbon of $2n$ wires.

Lemma 21. *Let* $n \in \mathbb{N}$. *Then*

– $\mathrm{int}_n \,;\, \mathrm{unint}_n \equiv \mathrm{unint}_n \,;\, \mathrm{int}_n \equiv \mathbf{I}^{2n}$,
– $\mathrm{unint}_n : 2n \otimes 2n$, *and*
– *for all* $x, y \in \Sigma^*$, $\mathrm{unint}_n \Vdash x \rightsquigarrow y$ *if and only if there exists* $u, v \in \Sigma^*$ *such that* $y = u \bullet v$ *and* $|u| = |v| = n$ *and* $x = u \,|||\, v$.

A.3 Duplication

We define by induction on $n \in NN$ the circuit dup_n by (1) $\mathrm{dup}_0 := \mathbf{0}$, and (2) $\mathrm{dup}_{n+1} := (\mathbf{Y} \,|\, \mathrm{dup}_n)\,;\, (\mathbf{I} \,|\, \mathrm{rol}_{n+1} \,|\, \mathbf{I}^n)$.

The next lemma states that the circuit dup_n duplicates a ribbon of n wires.

Lemma 22. *Let* $n \in \mathbb{N}$. *Then*

– $\mathrm{dup}_n : n \otimes 2n$, *and*
– *for all* $x, y \in \Sigma^*$, $\mathrm{dup}_n \Vdash x \rightsquigarrow y$ *if and only if* $|x| = n$ *and* $y = x \bullet x$.

Formal verification of an implementation of CRT-RSA Vigilant's algorithm

Maria Christofi[2,3], Boutheina Chetali[1], Louis Goubin[3], David Vigilant[2]

[1]Trusted Labs SAS - 5, rue du bailliage - 78000 Versailles - France
[2]Gemalto - 6, rue de la Verrerie - 92447 Meudon sur Seine - France
[3]Versailles Saint-Quentin-en-Yvelines University
{maria.christofi, david.vigilant}@gemalto.com,
boutheina.chetali@trusted-labs.com, louis.goubin@uvsq.fr

Abstract. Cryptosystems are highly sensitive to physical attacks, which leads security developers to design more and more complex countermeasures. Nonetheless, no proof of flaw absence has been given for any implementation of these countermeasures. This paper aims to formally verify an implementation of one published countermeasure against fault injection attacks. More precisely, the formal verification concerns Vigilant's CRT-RSA countermeasure which is designed to sufficiently protect CRT-RSA implementations against fault attacks. The goal is to formally verify whether any possible fault injection threatening the pseudo-code is detected according to a predefined attack model.

Keywords: fault attacks, frama-C, countermeasure, cryptographic implementation, formal verification, RSA

1 Introduction

Cryptographic implementations may be subject to physical attacks that disturb the execution of the embedded code. These attacks aim to disclose sensitive information or to force malicious behavior of the attacked code. To protect the implementations against this kind of attacks, countermeasures are designed, implemented and tested using several attack scenarios. To increase the level of confidence in the correctness of the countermeasure implementation, specific procedures of code review and cross-review are used. This "manual" verification procedure is itself error prone, and in some case its degree of exhaustivity depends on the time-to-market of the product.

The aim of this work is to provide the crypto-developer with a verification procedure which will improve the current process of correctness of the countermeasure with more automation and confidence. An implementation verification procedure can be seen as a procedure which takes as input an implementation (or a pseudo-code) with the corresponding countermeasure and outputs a "yes" or "no" answer. A "yes" answer means that the set of countermeasures present in the code is efficient enough to detect every possible attack scenario, according to a predefined attack model for the considered implementation, while a "no"

answer means that the developer has to improve this set in order to include the missing scenarios.

To the authors' knowledge, the formal verification of implementations of countermeasures has not really been the subject of research until now. Several works have been done on the formal verification of cryptosystems, but generally focused on the correctness of the cryptographic protocol with respect to its specification (like [2]) and more recently to its implementation.

Indeed such verification increases confidence in the cryptographic implementation and excludes flaws due to the weaknesses of the countermeasures.

The goal of this work is to demonstrate the robustness of the countermeasure with respect to a given attack model. For that, a classical approach consists in proving that an abstract model of the implementation with its countermeasure verifies a set of properties. Then using one of the existing approaches (empirical method, code generation, computational method) to convince the developer that the verified abstract model is a correct abstraction of the original code. The approach we take is to follow the developer view, using the source code of the cryptosystem with its countermeasures. For that, we will use a static analysis based tool which takes the pseudo-code as an input to the formal verification. Moreover, we will focus on a well-known cryptosystem, RSA, and more precisely the algorithm associated to Vigilant's countermeasure provided in [23].

Attacks based on information gained from the physical implementation of a cryptosystem, other than brute force or theoretical weaknesses in the algorithms, are called *side channel attacks*. Attacks are typically distinguished in passive (such as timing information, power consumption and electromagnetic leaks) and active (such as fault injection) attacks.

This paper considers only fault injection attacks and more precisely single fault injection attacks, i.e. attack scenarios where only one fault is injected.

Structure of the paper

In Sect. 2, fault attacks and some of the CRT-RSA countermeasures are reminded. Then the general idea of Vigilant's countermeasure is briefly presented in Sect. 3. Section 4 describes our methodology to formally verify an implementation of a countermeasure and then Sect. 5 presents the formal verification of the pseudo-code of Vigilant's countermeas ure as well as its results.

2 Fault attacks and countermeasures on CRT-RSA

This Section is a short introduction to fault injection attacks, and especially attacks targeting the CRT-RSA algorithm.

2.1 Fault injection attacks

Fault attacks consist in tampering with a device in order to have it perform some erroneous operations, hoping that the result of that erroneous behavior will leak information about the involved secret parameters.

The fault attacks in a specific code can be seen either as modifications of a specific variable or as modifications of code instructions (including modifications on the execution flow and logical level modifications). The former one concerns attacks that aim to trouble on the value of a register, while the later one concerns attacks on the instructions of the code. In [4], Bar-El and al. present various methods to induce faults and exploit such errors, and give several examples of both attacks and countermeasures.

Modifying a variable with a fault injection can be seen as adding new instructions that assign an arbitrary value to this variable. In the same vein, modifications of code instructions are simulated by a *goto* instruction. Formalizing modifications of instructions requires the formalization of the program execution, and this will be part of a future extension of this work. However, a first attempt to modelize thin kind of modifications, and especially the jump attacks, one can find in [6].

The methodology proposed in Sect. 4 aims to guarantee the validity of a countermeasure pseudo-code where the effect of the attack is the value modification of a variable.

Therefore, the level of the details provided in the pseudo-code is relevant with respect to the formalism. For example, the result of a formal verification can be different for a pseudo-code where the smallest manipulated variables are large integers, compared to a pseudo-code where the smallest variables are arrays of bits (or words) in a lower-level implementation. Indeed, the second pseudo-code would contain more steps including all multi-precision integers operations. And these extra steps would represent more locations for fault injections. Therefore the formal verification should be applied to a pseudo-code as fine as possible, in order to give the best confidence.

A fault can then be characterized by different aspects, like the number of affected bits, but also error location, time of occurrence and persistence. The different fault models are summarized in Table 1.

	Precise Bit Fault Model	**Single Bit Fault Model**	**Byte Fault Model**	**Random Fault Model**	**Arbitrary Fault Model**
control on location	complete (chosen bit)	loose (chosen variable)	loose	loose	loose/no
control on timing	precise	no	no	no	no
number of affected bits	1	1	8	random	random
fault type	bit set or reset	bit flip	random	random	unknown
persistence	permanent and transient	permanent and transient	permanent and transient	permanent and transient	permanent and transient

Table 1. Fault models

2.2 Countermeasures on CRT-RSA

Focusing now to the CRT-RSA algorithm, as a signing procedure and some already known countermeasures used to protect it.

Let $N = p \cdot q$ be a product of two large prime numbers. To sign a message m, one first computes $S_p = m^d \mod p$ and $S_q = m^d \mod q$ and then uses the Chinese Remainder Theorem (CRT) to build the signature $S = m^d \mod N$ (this is done by computing $S = (S_p \cdot q \cdot (q^{-1} \mod p) + S_q \cdot p \cdot (p^{-1} \mod q)) \mod N$).

CRT-RSA is especially susceptible to software or hardware errors. Boneh, DeMillo and Lipton were the first to present a fault attack on RSA in both standard and CRT mode [7]. In the case of the CRT-RSA algorithm, if a fault is induced during the computation of S_p (respectively S_q), then an erroneous value S'_p (resp. S'_q) is used during the CRT-recombination leading to an erroneous signature S'. As $S \equiv S_p \mod p$ and $S \equiv S_q \mod q$, we now have $S' \equiv S \mod q$ (resp. $S' \equiv S \mod p$), but $S' \not\equiv S \mod p$ (resp. $S' \not\equiv S \mod q$). Therefore, if $p \nmid (S - S')$ then the secret parameter q can be easily obtained by computing $\gcd(S - S', N)$. The other secret parameters of the private key $p, d_p(= e^{-1} \mod (p-1)), d_q(= e^{-1} \mod (q-1)), i_q(= q^{-1} \mod p)$ can then easily be computed.

An improvement of this attack comes later on by Lenstra in [19]. He claims that if a fault is induced during the computation of S_p then $S'^e \equiv m \mod q$ but $S'^e \not\equiv m \mod p$. Therefore the secret parameter q can be obtained by computing $\gcd(S'^e - m, N)$. The advantage of this attack comparing to the previous one is that now only one execution of the cryptographic algorithm is required to recover the private key.

However, for the above attacks, the attacker needs to know the whole message. Some efforts have already been done for attacks without the need of knowing it. As for example, the one of Coron and al. in [12].

An obvious countermeasure against these attacks is to verify the signature by using the public key (e, N). Usually e is small (for example $2^{16} + 1$), but this method may be very costly when e is large as it implies a second exponentiation. Moreover, the public exponent is not always available.

Since the publication of this attack, a large variety of countermeasures have been published in the field. The first method was proposed by Shamir in [22]. Shamir suggests to choose a small integer r, then compute $S_{pr} = m^d \mod pr$ and $S_{qr} = m^d \mod qr$ and ensure the integrity of these two exponentiations by testing whether $S_{pr} \equiv S_{qr} \mod r$ before combining S_{pr} and S_{qr} with the CRT formula. However, Aumüller and al. in [3] show that this method does not protect the CRT recombination and propose an implementation that also protects the CRT recombination. As opposed to Shamir's method, only d_p and d_q (and not d) are required. This solution gives good performance, as comparing to the classical CRT-RSA implementation, only two extra exponentiations and a few modular reductions are required. The main disadvantage of this method: it requires an extra prime parameter. There are already many improvements of Shamir's method, such as the one proposed by Vigilant in [23]. After some flaws

discovered, [11] presents an improvement of this algorithm giving two possible attacks and the corresponding countermeasures. The first attack concerns a fault that changes the last "*mod* N" operation, while the second one concerns the way that $p-1$ (resp. $q-1$) is computed/stocked. The first attack does not apply to the case of our model (due to the impossibility of implementing a "*mod* 0" operation, see later on for more details about the model used). The second attack demands a different implementation than the one presented in [23]. As said in Sect. 1, the results of our method are specific to the implementation verified and can be different for different implementations of the same algorithm. As we want to verify the original implementation of [23], this paper verifies Vigilant's algorithm as described in [23] against fault attacks.

Another protection has been proposed by Giraud in [15] in which the fault detection comes from the exponentiation algorithm. Actually, by using the Montgomery powering ladder to compute $m^d \mod N$, both values $m^d \mod n$ and $m^{d-1} \mod N$ are available at the end of the computation. These values can then be used to verify the integrity of the exponentiation. In [8], Boscher and al. also proposed a countermeasure where the detection comes from another exponentiation algorithm. Finally, Rivain proposed a detection method based on addition chains in [21].

Examples of pseudo-codes for implementing the countermeasures were only provided by Aumüller and al. in [3] and by Vigilant in [23]. This paper studies the pseudo-code provided by Vigilant.

3 Vigilant's CRT-RSA countermeasure

Vigilant's countermeasure is a method to protect a modular exponentiation against fault attacks. This method can be efficiently used for protecting CRT-RSA on embedded devices, since it does not require the public exponent, neither precomputation, nor extra parameters.

Protecting an exponentiation $S = m^d \mod N$ against fault attacks consists in computing $m^d \mod N$ in $\mathbb{Z}_{N \cdot r^2}$ where r is a small random integer co-prime with N. The message m is transformed into m' such that:

$$m' \equiv \begin{cases} m & \mod N \\ 1 + r & \mod r^2 \end{cases}$$

This implies that

$$S' = m'^d \mod Nr^2 \equiv \begin{cases} m^d & \mod N \\ 1 + d \cdot r & \mod r^2 \end{cases}$$

So, a consistency check of the result S' can be performed modulo r^2 from d and r. If the verification $S' \mod r^2 = 1 + d \cdot r \mod r^2$ is successful, then the final result $S = S' \mod N$ is returned.

This secure exponentiation can be applied to RSA with CRT. The principle is to perform two exponentiations modulo $p \cdot r^2$ and $q \cdot r^2$ (so we obtain S_p and

S_q respectively) and then perform a final consistency check after recombination, guaranteeing that no error occurred during the computation of S_p or S_q and during the recombination.

Algorithm 1 presents the pseudo-code of Vigilant's implementation as provided in [23].

Algorithm 1 Vigilant's CRT-RSA implementation code

Input: message m, e, key (p, q, d_p, d_q, i_q)
32-bit random integer r
64-bit random integers R_1, R_2, R_3, R_4
Output: signature $S = m^d \mod N$

$p' = p \cdot r^2$
$m_p = m \mod p'$
$i_{pr} = p^{-1} \mod r^2$
$\beta_p = p \cdot i_{pr}$
$\alpha_p = (1 - \beta_p) \mod p'$
$\hat{m}_p = (\alpha_p \cdot m_p + \beta_p \cdot (1 + r)) \mod p'$
if $(\hat{m}_p \neq m \mod p)$ **then return** error
$d'_p = d_p + R_1 \cdot (p - 1)$
$S_{pr} = \hat{m}_p^{d'_p} \mod p'$
if $(d'_p \neq d_p \mod (p - 1))$ **then return** error
if $(\beta_p \cdot S_{pr} \neq \beta_p \cdot (1 + d'_p \cdot r) \mod p')$ **then return** error
$S'_p = S_{pr} - \beta_p \cdot (1 + d'_p \cdot r) - R_3$
$q' = q \cdot r^2$
$m_q = m \mod q'$
$i_{qr} = q^{-1} \mod r^2$
$\beta_q = q \cdot i_{qr}$
$\alpha_q = (1 - \beta_q) \mod q'$
$\hat{m}_q = (\alpha_q \cdot m_q + \beta_q \cdot (1 + r)) \mod q'$
if $(\hat{m}_q \neq m \mod q)$ **then return** error
if $(m_p \mod r^2 \neq m_q \mod r^2)$ **then return** error
$d'_q = d_q + R_2 \cdot (q - 1)$
$S_{qr} = \hat{m}_q^{d'_q} \mod q'$
if $(d'_q \neq d_q \mod (q - 1))$ **then return** error
if $(\beta_q \cdot S_{qr} \neq \beta_q \cdot (1 + d'_q \cdot r) \mod q')$ **then return** error
$S'_q = S_{qr} - \beta_q \cdot (1 + d'_q \cdot r) - R_4$
$S = S'_q + q \cdot (i_q \cdot (S'_p - S'_q) \mod p')$
$N = p \cdot q$
if $(N \cdot [S - R_4 - q \cdot i_q \cdot (R_3 - R_4)] \neq 0 \mod Nr^2)$ **then return** error
if $(q \cdot i_q \neq 1 \mod p)$ **then return** error
return $S \mod N$

This implementation has many advantages:

– no need of special hypotheses for r. However, in [23] we can find some recommendations about r, such that $i_q \neq 0 \mod r$, r should be odd, at least a 32-bit random integer and as large as possible
– no precomputation is needed
– only p, q, d_p, d_q, i_q, and m are needed for the calculation

4 Formal verification of implementations of countermeasures

The aim of this work is to formally verify the resistance of the pseudo-code described in Algorithm 1 against fault attacks. The goal is to build a formal environment that will allow the cryptographic engineer to introduce his secure code and check the validity of his countermeasure. The main steps of the verification procedure to follow are:

1. define the implementation that we want to verify with the corresponding set of countermeasures
2. choose a fault model
3. simulate every possible injected fault with respect to this fault model
4. inject this fault model to the original source code implementation
5. model the property corresponding to the verification
6. use a tool to generate some proof obligations corresponding to the property to prove
7. prove these obligations (either automatically or using a proof assistant)

For the verification part of our work, we use a static analysis based tool, named frama-C [14], that will allow to perform an analysis of the source code without executing it. The source code will then correspond to the implementation of the cryptosystem with its countermeasures along with a simulation of the chosen fault model.

4.1 Frama-C

Frama-C [14] is an open source extensible platform dedicated to source code analysis of C software. The frama-C platform gathers several static analysis techniques into a single collaborative extensible framework. The collaborative approach of frama-C allows static analyzers to build upon the results already computed by other analyzers in the framework.

In addition, frama-C verifies some "safety properties" like the division by zero or loop's termination and correctness.

One of the advantages of frama-C, against other tools of static analysis or even bug-finding tools, is that it allows its user to manipulate functional specifications, and to prove that the source code satisfies these specifications written in a dedicated language ACSL [1] (ANSI/ISO C Specification Language, a behavioral specification language for C programs). ACSL is a language of annotations, threatened as standard comments by the C compiler, that allows the user to

express the above specifications in such a way that they do not affect a normal execution of the implementation but they are verified by frama-C.

Frama-C is a plugin system. In order to perform a verification, we use Jessie [17], the deductive verification plugin of C programs annotated with ACSL. It uses internally the languages and tools of the Why platform [24] [1]. The Jessie plugin uses Hoare-style [16] weakest precondition computations to formally prove ACSL properties. The generated verification conditions can be submitted to external automatic provers such as Simplify, Alt-Ergo, Z3, CVC3.

For more complex situations, interactive theorem provers, like Coq, PVS, Isabelle/HOL, can be used to establish the validity of the verification conditions.

The aim of the work presented in this paper is the verification of a C code including a cryptographic implementation and a simulation of all possible fault attacks. For this, Jessie needs as input this transformed code and outputs the proof obligations to verify using an automatic or an interactive prover. The user is then free to exploit these results.

4.2 Fault model

As this is a first attempt to formally verify a cryptographic implementation, we have chosen a quite simple fault model which is still realistic, but different from the one described in [23]. The two models are clearly not equivalent. However, the verification procedure is still the same for other models and it will be part of our future work.

In the original fault model, the attacker can:

- inject only one fault per execution
- modify a value in memory obtaining a totally random result uncorrelated to the original value (known as permanent fault)
- modify a value when it is handled in local registers, without modifying the global value in memory. The handled value obtained is fully random from the attacker point of view and uncorrelated to the original value (known as transient fault)

but the attacker cannot:

- modify the code execution. Processor instructions cannot be replaced or removed while executing code
- inject a permanent fault in the input elements, the message m or the key (p, q, d_p, d_q, i_q)
- change the boolean result of a conditional check. An expression "if $a = b$" has a result true or false that cannot be modified.

Our fault model is based on the above with three differences. We consider that the attacker :

- can modify the value in memory but by only setting the value to 0 (in the case of the pseudo-code, this corresponds to set the whole variable to 0)

[1] WHY is a general-purpose verification condition generator, which is used as a back-end by other verification tools but which can also be used directly to verify programs. WHY produces verification conditions from annotated programs given as input.

– can inject a permanent fault in the input elements, the message m as well as the key (p, q, d_p, d_q, i_q)
– cannot inject a fault in m at the very beginning (i.e. before line 1 of the Algorithm 1) of the implementation.

4.3 Fault injection simulation

Once the fault model is chosen, it must be injected in the initial code of the implementation. This simulation consists in setting the value of the "attacked" variable to 0, for every possible fault. Obviously such a modeling creates a lot of cases to verify. The number of the cases increases according to the number of the code instructions and the variables used in it. Thus, for codes that describe real cryptographic implementations, this modeling may become very huge and so, quite inefficient.

For that we introduce an optimization by defining some equivalence classes between attacks that have the same effects. To do so, we use the notions of *read* and *write* for any variable used in the code. The general idea is to characterize every line of the original code by a *read, write, read/write,* $\emptyset$ type according to the actions occurred to the variables appeared in it. The *read* (resp. *write*) type means that the considered code line reads (resp. writes) the variable. The *read/write* type means that the code line performs a read and a write operation (as for example, for the variable *var* in the instruction $var = var + 1$). The $\emptyset$ means that no operation is performed concerning this variable. Let *Type(var,i)* define the characterization of the variable *var* on line i.

We then determine the next use of a variable *var* with the help of the following definition.

Definition 1. *Let consider that we start our analysis from the line i, NextType (var, i) is the next found typed line using var and is defined as follows:*

$$NextType\ (var,\ last) = \emptyset$$

$$NextType(var, i) = \begin{cases} Type(var, i) & , if\ Type(var, i)\ \neq\ 0 \\ NextType(var, i+1)\ , & otherwise \end{cases}$$

where last is the last line of the source code.

The different types are illustrated in a simple example in Table 2.

Attacks on code with sequential control flow. To simplify, let's first focus to a code without any loops nor conditionals. For such a code, the equivalence classes correspond to the minimal code to verify in order to ensure a security property. In fact, the class of the original source code includes all the attacked codes for which the attack is useless. Formally, we have:

Lemma 1. *If $NextType(var, i) \in \{write, \emptyset\}$, then an attack on var injected at line i is useless and equivalent to the original source code.*

1: int example(int a, int b){		
2: int x = 0;	// Type (x,2) = write	On line 2, the use of x is of type "write".
3: a = a + 1;	// Type (a,3) = read/write	On line 3, the use of a is of type "read" and "write".
	// NextType (x,3) = write	On line 3, the next use of x is of type "write" (on line 4).
	// NextType (a,3) = read	On line 3, the next use of a is of type "read" (on line 4).
4: x = a + b;	// NextType (x,4) = $\emptyset$	On line 4, there is no next use of x.
5: }		

Table 2. Code example

Obviously, if the next use of *var* is "write", the operation performed will have no effect to the value of *var* stored in memory. Contrary to the cases that the next use of *var* is "read" or "read/write" where the following lemma is applied:

Lemma 2. *If* $NextType(var, i) \in \{read, read/write\}$ *and* j *the line that represents the next use of the variable var, then an attack on var injected at the interval* $[i, j]$ *has exactly the same effect on var than an attack injected at line* j*, but has no effect between lines* i *and* $j - 1$*.*

The aim of these two lemmas is to separate the useful attacks from the useless ones, i.e. the attacks that have an effect on the code from the ones that have no effect. It reduces the number of the attacks so that only useful attacks are kept. These two lemmas are summarized to the following theorem (we recall that for the moment we have a code with no loop and no conditionals):

Theorem 1. *If there are* n *read and read/write operations on the code for one variable, the minimal number of faults with different effect for this variable is* $n + 1$ *(i.e. one attack for every read and read/write operation plus the original code-without faults injection).*

Attacks on code with conditionals and loops. Let us now consider the case of a source code with conditionals and loops. The type of any line can be defined in the same way as for any other line of a non conditional code, thus Lemma 1 remains valid.

We first deal with the conditional instructions (an if-then-else structure). This part of code can be decomposed in three parts: the condition, the then-block and the else-block (which can be empty). It is possible to inject attacks at either the condition or the then/else-block.

As in the case of code with sequential control flow, we inject an attack before any *read* operation of every variable.

However, as we want to minimize the number of injected attacks, if no *read* operation happens during the if-condition and a *read* operation happens in both the then and the else block, instead of injecting an attack at both corresponding lines, we can inject an attack before the if-condition when no operation is

performed between the if-condition and both these *reads*. An example is given in Fig. 1. This can be done only in the case of fault models where the fault is always of the same nature. An example of this kind of fault model is the one studied in this paper, which consists on setting a value to 0. An example of fault model that we cannot apply this optimization is the fault model which sets a value to a random value. This is because every fault injection can correspond to another random value.

```
1: int example_if(int x, int y){
2:         if (y > 0 )                  // condition
3:           {y = x; }                  // then-block
4:         else y = −x;                 // else-block
5:         return y;
6: }
```

Fig. 1. Code example with conditionals (considering attacks on variable x). For the fault model consisting on setting a value on 0, instead of injecting two attacks in both lines 3 and 4, we can inject one and only attack in line 2.

In the same vein, for the loop instructions, we inject an attack before any *read* operation of every variable.

4.4 Adding the fault model to the implementation

Before starting the verification, the simulated fault model will be added to the original code. For that, an additional variable is used, named f, which represents the faults. All possible attacks are finally introduced in such a way that this part of code will be executed once the corresponding simulated attack occurs.

As an example, one can see Figure 2. In this example, a fault consists on setting the value of a variable to 0. The lines 1, 6, 11 and 12 of the transformed code are equivalent to the initial code, while both the lines 3 and 8 represent attacks to the variable x, and lines 4 and 9 attacks to the variable y. Lines 2 to 5 describe all possible attacks for the instruction at the line 6, while lines 7 to 10 describe all possible attacks for the return statement at line 11.

Similarly, all possible attacks (w.r.t. the fault model) can be simulated and induced into the original code. Currently, the process of generating automatically the simulation into the original code is a work in progress.

4.5 Modeling the main property

The goal of the verification is to prove, for a given implementation, the validity of a set of countermeasures with respect to a set of attacks (for a given attack model). In other words, given an implementation and a set of countermeasures,

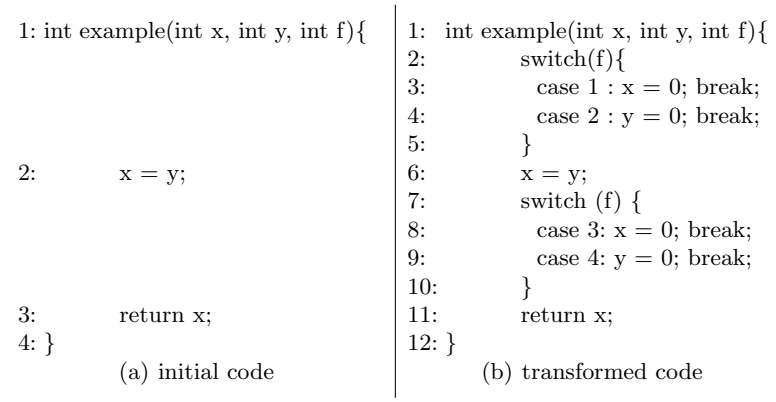


Fig. 2. An example of a fault injection in the code

we want to prove whether any attack by fault injection (w.r.t. the attack model) is detected (an error flag is raised).

For the fault model studied in this paper, this means that the output of any execution of the given code is either the expected result or the error flag. As we cannot know in advance the expected result, we have to express it in terms of a function using the entry variables. The property to prove is then summarized to the Theorem 2.

Theorem 2. *Let $f \in \{0\} \cup F$, where F is the set of faults for the current implementation and $f = 0$ the original execution of the implementation (without injected faults). Let also res be the output of the implementation, $x_1, ..., x_n$ be the n variables of the input of the implementation and g a function. Then :*

$$[(f = 0) \Rightarrow (res = g(x_1, ..., x_n))] \; AND \; [(\forall f \in F) \Rightarrow (res = ERROR)]$$

When the output is the error flag, it means that the countermeasures are robust in the sense that they detect any fault injection (according to the model).

5 Formal verification of the pseudo-code of Vigilant's countermeasure

The following section describes the use of the presented approach to the pseudo-code of Vigilant's countermeasure. The verification is based on the procedure described in Sect. 4.

As described in Sect. 4.2, the fault model we use is the following:
An attacker can:

- inject only one fault per execution
- modify the value in memory by setting the value to 0
- inject both transient and permanent faults to any variable

but (s)he cannot:

- modify the code execution
- inject a fault in m at the very beginning (that is before line 1 of the Algorithm 1) of the implementation.
- inject a fault in S at the very end (i.e. after line 31 of the Algorithm 1) of the implementation
- change the boolean result of a conditional check. An expression "if $a = b$ " has a result true or false that cannot be modified.

For the pseudo-code of Vigilant's CRT-RSA algorithm presented in [23], under the above assumptions and using the procedure described in this paper, 95 possible faults are obtained. These faults are presented in the Appendix A.

Some additional hypotheses have to be made:

- $m \mod p \not\equiv 0$ and $m \mod q \not\equiv 0$
- r is odd and $i_q \not\equiv 0 \mod r$ as it is recommended in [23]
- $\gcd(p, r^2) = 1$ and $\gcd(q, r^2) = 1$, for the efficiency of the computation of i_{pr} and i_{qr} respectively

Once every possible fault is injected using the method described in Sect. 4 and with respect to the above fault model, we call the frama-c platform with the jessie plugin to run the verification procedure of the property of Theorem 2.

The results of this verification indicate some cases of faults (the underlined cases in Algorithm 2 of Appendix A) that are not detected by the given countermeasures.

"Sensitive" cases are separated in three main categories:

- The first category contains cases with success probability one (that means that such a fault will never be detected). These cases (cases 19, 36, 60 and 77 in Algorithm 2) correspond to faults on the random values R_1, R_2, R_3 and R_4 and concern the randomization of some variables. In these cases, the output is the real signature and no information about the secret values is obtained. Hence, these cases are of a real interest as we can expect the same behavior whenever a random value appears. However, whenever we obtain the valid signature, the attacks presented in Sect. 2.2 are no more applicable. (Depending on the fault model this can give some information to the attacker about the attacked variable)
- The second (and the bigger) one contains cases with a weak success probability. (Noting $|x|$ the size of x)
  - For the cases 6, 8, 13, 27, 29, 33, 34, 41, 44, 46, 51, 68, 70, 74, 75, 79, 82, 87, 88 and 91, the probability that an injected fault is undetectable is $2^{-2|r|+1}$.
  - For the cases 22, 28 and 32, this probability is $2^{-(|p'|-1)}ln2$.
  - For the cases 63, 69 and 73, this probability is $2^{-(|q'|-1)}ln2$.

  We notice here that frama-C tool cannot manipulate probabilities. The probabilities mentioned here are manually calculated (see Appendix B for more details).
- The last category contains cases with a high success probability (in this case 1) and where the output is a faulty signature. These are the most dangerous cases as we can extract information about the secret values. These cases

are: 18 and 59 and correspond to permanent faults on d_p and d_q during the computation of d'_p and d'_q respectively. In case 18 (respectively 59), we obtain a faulty signature modulo p (resp. modulo q) and the right one modulo q (resp. modulo p). So it will be easy for the attacker to compute q (resp. p) and then the other secret parameters. Indeed as already said, our fault model allows permanent faults on d_p and d_q, contrary to the original fault model. This fault model difference is of prime importance for our results here.

6 Related work

To our knowledge, the use of frama-C for the verification of countermeasures is novel, but other uses of frama-C already exist. In [13], one can find the results of a formal verification of source code of a model of automaton in SAM language and its C language implementation, obtained using frama-C and Caveat. In [9], one can find a formal proof of correctness of the key commands of the SCHUR software, which is an interactive program for calculating with characters of Lie groups and symmetric functions. Another example of a use of frama-C is [5] which is about verification of some interval security properties for smart card C codes using value analysis.

Other verification techniques, such as model checking, are also quite common to verify temporal properties in programs. In [18], such a verification concerning safety properties can be found, while in [10], one can find the results of a verification of a real system using MOPS - a tool for software model checking security-critical applications-. Although model checking is fully automated, it is limited to simple implementations due to the exhaustive exploration of the model.

Another remarkable effort on verifying programs with the presence of faults is made in [20] (thank to the anonymous reviewer for this citation), where the authors have developed a new logic for reasoning about faults.

7 Conclusion and perspectives

Vigilant's countermeasure is a countermeasure protecting modular exponentiations against fault attacks that was applied to CRT-RSA. In this paper, we have presented the results of the formal verification of the resistance of the pseudo-code provided in [23] against fault attacks, with respect to the fault model described above.

The obtained results are very promising. The approach has been developed with a simple fault model. The goal now is first to evaluate the pertinence of this fault model with the crypto-developers. Then, we plan to extend this method to other fault models and to double fault attacks. This work will continue along with experimentations on other cryptographic countermeasures. More practical steps are also planned, such as improving the automation in order to provide crypto-developers with a full validation environment.

Acknowledgments: The authors would like to thank Pascal Paillier for his useful contribution to this work.

References

1. ACSL. `http://frama-c.com/acsl.html`.
2. Mihhail Aizatulin, François Dupressoir, Andrew D. Gordon, and Jan Jürjens. Verifying Cryptographic Code in C: Some Experience and the Csec Challenge. In *Formal Aspects of Security and Trust - 8th International Workshop, FAST 2011, Leuven, Belgium, September 12-14, 2011. Revised Selected Papers*, volume 7140 of *Lecture Notes in Computer Science*, pages 1–20. Springer, 2012.
3. Christian Aumüller, Peter Bier, Wieland Fischer, Peter Hofreiter, and Jean-Pierre Seifert. Fault Attacks on RSA with CRT: Concrete Results and Practical Countermeasures. In *CHES*, volume 2523 of *Lecture Notes in Computer Science*, pages 260–275. Springer, 2003.
4. Hagai Bar-El, Hamid Choukri, David Naccache, Michael Tunstall, and Claire Whelan. The sorcerer's apprentice guide to fault attacks. *IACR Cryptology ePrint Archive*, 2004:100, 2004.
5. P. Berthomé, K. Heydemann, X. Kauffmann-Tourkestansky, and J.-F. Lalande. Attack model for verification of interval security properties for smart card C codes. In *Proceedings of the 5th ACM SIGPLAN Workshop on Programming Languages and Analysis for Security*, PLAS '10, pages 2:1–2:12, New York, NY, USA, 2010. ACM.
6. Pascal Berthomé, Karine Heydemann, Xavier Kauffmann-Tourkestansky, and Jean-François Lalande. Simulating physical attacks in smart card C codes: the jump attack case. In *e-Smart*, 2011.
7. Dan Boneh, Richard A. DeMillo, and Richard J. Lipton. On the importance of eliminating errors in cryptographic computations. *J. Cryptology*, 14(2):101–119, 2001.
8. Arnaud Boscher, Robert Naciri, and Emmanuel Prouff. CRT RSA Algorithm Protected Against Fault Attacks. In *WISTP*, volume 4462 of *Lecture Notes in Computer Science*, pages 229–243. Springer, 2007.
9. Franck Butelle, Florent Hivert, Micaela Mayero, and Frédéric Toumazet. Formal Proof of SCHUR Conjugate Function. In *AISC/MKM/Calculemus*, volume 6167 of *Lecture Notes in Computer Science*, pages 158–171. Springer, 2010.
10. Hao Chen, Drew Dean, and David Wagner. Model Checking One Million Lines of C Code. In *NDSS*. The Internet Society, 2004.
11. Jean-Sébastien Coron, Christophe Giraud, Nicolas Morin, Gilles Piret, and David Vigilant. Fault Attacks and Countermeasures on Vigilant's RSA-CRT Algorithm. In *FDTC*, pages 89–96. IEEE Computer Society, 2010.
12. Jean-Sébastien Coron, David Naccache, and Mehdi Tibouchi. Fault attacks against emv signatures. In *CT-RSA*, volume 5985 of *Lecture Notes in Computer Science*, pages 208–220. Springer, 2010.
13. Stéphane Duprat, Pierre Gaufillet, Victoria Moya Lamiel, and Frdric Passarello. Formal verification of SAM state machine implementation. In *Embedded Real Time Software and Systems (ERTS'10)*, 2010.
14. frama-c. `http://frama-c.com/`.
15. Christophe Giraud. An RSA Implementation Resistant to Fault Attacks and to Simple Power Analysis. *IEEE Trans. Computers*, 55(9):1116–1120, 2006.

16. C. A. R. Hoare. An axiomatic basis for computer programming (reprint). *Commun. ACM*, 26(1):53–56, 1983.
17. Jessie. `http://krakatoa.lri.fr/#jessie`.
18. Orna Kupferman and Moshe Y. Vardi. Model checking of safety properties. *Formal Methods in System Design*, 19(3):291–314, 2001.
19. Arjen Lenstra. Memo on RSA signature generation in the presence of faults. manuscript, 1996.
20. Matthew L. Meola and David Walker. Faulty logic: Reasoning about fault tolerant programs. In *Programming Languages and Systems, 19th European Symposium on Programming, ESOP 2010, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2010, Paphos, Cyprus, March 20-28, 2010. Proceedings*, volume 6012 of *Lecture Notes in Computer Science*, pages 468–487. Springer, 2010.
21. Matthieu Rivain. Securing RSA against Fault Analysis by Double Addition Chain Exponentiation. *IACR Cryptology ePrint Archive*, 2009:165, 2009.
22. Adi Shamir. Improved method and apparatus for protecting public key schemes from timing and fault attacks. Patent number: WO9852319, 1998.
23. David Vigilant. RSA with CRT: A New Cost-Effective Solution to Thwart Fault Attacks. In *CHES*, volume 5154 of *Lecture Notes in Computer Science*, pages 130–145. Springer, 2008.
24. Why. `http://why.lri.fr/`.

A Vigilant's CRT-RSA implementation code with fault simulation

Algorithm 2 Vigilant's CRT-RSA implementation code with fault simulation

Input: message m, e, key (p, q, d_p, d_q, i_q)
32-bit random integer r
64-bit random integers R_1, R_2, R_3, R_4
an integer f
Output: signature $S = m^d \mod N$

switch (f) {
    case 1 : $p = 0$
    case 2 : $r = 0$
}
$p' = p \cdot r^2$
switch (f) {
    case 3 : $p' = 0$
}
$m_p = m \mod p'$
switch (f) {
    case 4 : $p = 0$
    case 5 : $r = 0$
}
$i_{pr} = p^{-1} \mod r^2$
switch (f) {
    case 6 : $i_{pr} = 0$
    case 7 : $p = 0$
}
$\beta_p = p \cdot i_{pr}$
switch (f) {
    case 8 : $\beta_p = 0$
    case 9 : $p' = 0$
}
$\alpha_p = (1 - \beta_p) \mod p'$
switch (f) {
    case 10 : $\alpha_p = 0$
    case 11 : $m_p = 0$
    case 12 : $p' = 0$

34: **case** 13 : $\beta_p = 0$
35: **case** 14 : $r = 0$
36: }
37: $\hat{m}_p = (\alpha_p \cdot m_p + \beta_p \cdot (1 + r)) \mod p'$
38: **switch** (f) {
39: **case** 15 : $\hat{m}_p = 0$
40: **case** 16 : $m = 0$
41: **case** 17 : $p = 0$
42: }
43: **if** $(\hat{m}_p \neq m \mod p)$ **then return** error
44: **switch** (f) {
45: **case** 18 : $d_p = 0$
46: **case** 19 : $R_1 = 0$
47: **case** 20 : $p = 0$
48: }
49: $d'_p = d_p + R_1 \cdot (p - 1)$
50: **switch** (f) {
51: **case** 21 : $d'_p = 0$
52: **case** 22 : $\hat{m}_p = 0$
53: **case** 23 : $p' = 0$
54: }
55: $S_{pr} = \hat{m}_p^{d'_p} \mod p'$
56: **switch** (f) {
57: **case** 24 : $d'_p = 0$
58: **case** 25 : $d_p = 0$
59: **case** 26 : $p = 0$
60: }
61: **if** $(d'_p \neq d_p \mod (p - 1))$ **then return** error
62: **case** 27 : $\beta_p = 0$
63: **case** 28 : $S_{pr} = 0$
64: **case** 29 : $d'_p = 0$
65: **case** 30 : $r = 0$
66: **case** 31 : $p' = 0$
67: **if** $(\beta_p \cdot S_{pr} \neq \beta_p \cdot (1 + d'_p \cdot r) \mod p')$ **then return** error
68: **switch** (f) {
69: **case** 32 : $S_{pr} = 0$
70: **case** 33 : $\beta_p = 0$
71: **case** 34 : $d'_p = 0$
72: **case** 35 : $r = 0$
73: **case** 36 : $R_3 = 0$
74: **case** 37 : $p' = 0$
75: }
76: $S'_p = (S_{pr} - \beta_p \cdot (1 + d'_p \cdot r) - R_3) \mod p'$
77: **switch** (f) {
78: **case** 38 : $q = 0$
79: **case** 39 : $r = 0$
80: }
81: $q' = q \cdot r^2$
82: **switch** (f) {
83: **case** 40 : $q' = 0$
84: **case** 41 : $m = 0$
85: }
86: $m_q = m \mod q'$

87: **switch** (f) {
88: **case** $42 : q = 0$
89: **case** $43 : r = 0$
90: }
91: $i_{qr} = q^{-1} \mod r^2$
92: **switch** (f) {
93: **case** $44 :$ $i_{qr} = 0$
94: **case** $45 : q = 0$
95: }s
96: $\beta_q = q \cdot i_{qr}$
97: **switch** (f) {
98: **case** $46 :$ $\beta_q = 0$
99: **case** $47 : q' = 0$
100: }
101: $\alpha_q = (1 - \beta_q) \mod q'$
102: **switch** (f) {
103: **case** $48 : \alpha_q = 0$
104: **case** $49 : m_q = 0$
105: **case** $50 : q' = 0$
106: **case** $51 :$ $\beta_q = 0$
107: **case** $52 : r = 0$
108: }
109: $\hat{m}_q = (\alpha_q \cdot m_q + \beta_q \cdot (1 + r)) \mod q'$
110: **switch** (f) {
111: **case** $53 : \hat{m}_q = 0$
112: **case** $54 : m = 0$
113: **case** $55 : q = 0$
114: }
115: **if** ($\hat{m}_q \neq m \mod q$) **then return** error
116: **case** $56 : m_p = 0$
117: **case** $57 : m_q = 0$
118: **case** $58 : r = 0$
119: }
120: **if** $(m_p \mod r^2 \neq m_q \mod r^2)$ **then return** error
121: **switch** (f) {
122: **case** $59 :$ $d_q = 0$
123: **case** $60 :$ $R_2 = 0$
124: **case** $61 : q = 0$
125: }
126: $d'_q = d_q + R_2 \cdot (q - 1)$
127: **switch** (f) {
128: **case** $62 : d'_q = 0$
129: **case** $63 :$ $\hat{m}_q = 0$
130: **case** $64 : q' = 0$
131: }
132: $S_{qr} = \hat{m}_q^{d'_q} \mod q'$
133: **switch** (f) {
134: **case** $65 : d'_q = 0$
135: **case** $66 : d_q = 0$
136: **case** $67 : q = 0$
137: }

```
138: if (d'_q ≠ d_q  mod (q − 1)) then return error
139:     case 68 : β_q = 0
140:     case 69 : S_qr = 0
141:     case 70 : d'_q = 0
142:     case 71 : r = 0
143:     case 72 : q' = 0
144: if (β_q · S_qr ≠ β_q · (1 + d'_q · r)  mod q') then return error
145: switch (f) {
146:     case 73 : S_qr = 0
147:     case 74 : β_q = 0
148:     case 75 : d'_q = 0
149:     case 76 : r = 0
150:     case 77 : R_4 = 0
151:     case 78 : q' = 0
152: }
153: S'_q = (S_qr − β_q · (1 + d'_q · r) − R_4)  mod q
154: switch (f) {
155:     case 79 : S'_q = 0
156:     case 80 : q = 0
157:     case 81 : i_q = 0
158:     case 82 : S'_p = 0
159:     case 83 : p' = 0
160: }
161: S = S'_q + q · (i_q · (S'_p − S'_q)  mod p')
162: switch (f) {
163:     case 84 : p = 0
164:     case 85 : q = 0
165: }
166: N = p · q
167: switch (f) {
168:     case 86 : N = 0
169:     case 87 : S = 0
170:     case 88 : R_4 = 0
171:     case 89 : q = 0
172:     case 90 : i_q = 0
173:     case 91 : R_3 = 0
174:     case 92 : r = 0
175: }
176: if (N · [S − R_4 − q · i_q · (R_3 − R_4)] ≠ 0  mod N · r^2) then return error
177:     case 93 : q = 0
178:     case 94 : i_q = 0
179:     case 95 : p = 0
180: if (q · i_q ≠ 1  mod p) then return error
181: return S  mod N
```

B Details concerning the success probabilities of fault attacks

In this Appendix, we would like to give more details about the computation of the probabilities presented in Sect. 5. Noting $|x|$ the size of x.

Assume that the attacker modifies value A ($A = B \mod C$) and that C is a uniform, t-bit integer. We suppose that C is odd (r is odd according to the recommendations in Sect. 5, as well as p and q) and we force $2^{t-1} < C < 2^t$. Note $S = \{C : 2^{t-1} < C < 2^t \; and \; C = 1 \mod 2\}$.

We note U the event that the fault is undetected and F the event of taking an element c in S s.t. $c = C$. So, $Pr[U|F]$ is the probability that an event is undetected assuming F. Since the final result will depend only on the initial values which are uniformly distributed (the only exception may be the message m. To avoid this case, we can assume that the message used is the message obtained after a padding - like OAEP-. So the resulted m will also be uniformly distributed), we know that:

$$Pr[U|F] = \frac{1}{C} \; and \; Pr[F] = \frac{1}{|S|}$$

and then

$$Pr[U] = \sum_{C \in S} (Pr[U|F] \cdot Pr[F]) = \frac{1}{|S|} \cdot \sum_{C \in S} \frac{1}{C}$$

Let $\overline{S} = \{C : 2^{t-1} < C < 2^t \; and \; C = 0 \mod 2\}$, then

$$\sum_{C \in S \cup \overline{S}} \frac{1}{C} = [lnC]_{2^{t-1}}^{2^t} = ln(2^t) - ln(2^{t-1}) = ln2$$

We consider approximately that $|S| = |\overline{S}|$. Then:

$$Pr[U] = \frac{1}{|S|} \cdot \sum_{C \in S} \frac{1}{C} \approx \frac{1}{|S|} \cdot \frac{1}{2} \cdot \sum_{C \in S \cup \overline{S}} \frac{1}{C} = \frac{1}{|S|} \cdot \frac{ln2}{2} = \frac{1}{2^{t-2}} \cdot \frac{ln2}{2} = 2^{-(t-1)} ln2$$

This is the obtained probability for the faults: 22, 28 and 32 with $t = |p'|$, 63, 69 and 73 with $t = |q'|$.

Supposing now, that the attacker modifies a value A ($A = B \mod C^2$). Following the same reasoning, we conclude that :

$$Pr[U] \approx 2^{-2t+1}$$

This is the obtained probability for the faults: 6, 8, 13, 27, 29, 33, 34, 41, 44, 46, 51, 68, 70, 74, 75, 79, 82, 87, 88 and 91 with $t = |r|$.

Toward a Taxonomy of Communications Security Models

Mark Brown
RedPhone Security
mark@redphonesecurity.com

Abstract. Formal specifications, models and their accompanying proofs have long been promoted as setting the highest standard for program verification. But computer security remains threatened by covert channels, subliminal channels, side channels, fault injections, bypass, protocol attacks and subversion despite rigorous application of formal methods. We advance the thesis that there exist several hierarchically ordered and adjacent sciences, notations, and security requirements analyses which must each be addressed to achieve comprehensive security. We support this thesis from a survey of relevant models of communications security. We argue that a taxonomy of security concerns consisting of levels called "Epochs" provides a comprehensive framework for identifying, locating and analyzing security requirements and assumptions, and gives sense to the effective use of formal methods.

Keywords: formal models, formal specifications, security requirements

1 Introduction: Classical Computer Security Models

Mathematical, formal models of Gödel-Kleene recursive functions and Church-Turing computability led to the birth of computer science. Formal models can express algorithms in a mathematical notation and prove the algorithms exhibit first order correctness properties such as safety or liveness. [1] For example, one might express an algorithm within an interactive theorem proof assistant, and prove that the specified algorithm has some property of interest. However, properties of algorithms provide a constructive definition of correct outputs but not a secure realization. Proofs about algorithms typically assume type correctness on inputs, that is at least the inputs have been chosen from the specified set ($x \in X$) and also assume "no other changes" within the system. Such assumptions may fail when some fundamental layer of the system is subverted, when malicious inputs are presented, when guarding conditions are bypassed, when forged facsimiles are accepted as authentic information, or in general, when some assumption required for correctness fails. In this Introduction we recount the history of suggestions and surprises concerning which requirements should be the subject of formal security models, and what proofs and assumptions address security concerns.

The classical computer and network security model arose in the early 1970's with RAND's Ware Report [2], the United States Air Force's Anderson Report [3] [4], and the Bell-LaPadula model [5]. These documents identified problems – including operating system subversion – in which some assumption was forced to fail by an attacker. Proposed solutions of this era included the Reference Monitor concept [6] and a notional precondition of nonbypassibility. [3] Researchers advanced formal models of computer security (as in [7] and [8]), which laid the foundation for secure classical operating system kernels.

The problem of design and formal verification for classical models of secure operating systems was addressed by many researchers, including Saltzer and Schroeder [9], Denning [10], Popek and Kline [11], Landwehr [12], Kemmerer [13] and others. These classical computer security efforts shared the same goal: to secure isolated systems and only after that to allow for the modular addition of communications security in the form of cryptographic hardware units – i.e. a network. This classical computer and network security model led to the formation of the US National Computer Security Center in 1981 and its development of the Trusted Computer Security Evaluation Criteria (TCSEC), followed by the first attempted evaluations of security kernels such as KSOS, SCOMP, LOCK [14] [15], and GEMSOS [16]. TCSEC was succeeded by Common Criteria but even so, few high assurance products have been successfully evaluated. In fact, researchers from that era have even questioned the meaningfulness of the quest for multi-level secure operating systems. [17]

As these models were being developed, covert channels – which are information leakages that break intended security requirements – were being discovered and studied. They were first identified in 1973 by Butler Lampson. He defined them as channels "not intended for information transfer at all" and required enforcement such that the operating system kernel supervisor "must ensure that a confined program's input to covert channels conforms to the caller's specifications." [18, p. 4] Lampson stipulated that "It is necessary to enumerate them all [leakage channels] and then to block each one," but also allowed that "there is not likely to be any rigorous way of identifying every channel in any system of even moderate complexity." [18, p. 4] A formal model used to measure covert channels was introduced by Goguen and Meseguer in 1982. [19] In following years, other covert channel analyses have been published, using various models and definitions (a partial survey is [20]). Toward a solution to the problem of covert channels, Lampson proposed two confinement conditions for a program: 1) it must be memoryless, that is, "it must not be able to preserve information within itself from one call to another," and 2) "total isolation", that is, "A confined program shall make no calls on any other program." [18, p. 3]

In 1981, John Rushby formalized Lampson's radical confinement concept: rather than starting with an operating system and attempting to secure it against covert channels, one should begin with a minimalistic formal model that could express leak-free operation and allow that model to reorganize whatever secure system might be built from it. [21] Rushby's more detailed paper explicitly puts forward information flow analysis over access control security, asserting that only the former can "establish the absence

of storage and legitimate channels." [22] The resulting system might not have all the features of an operating system, suggested Rushby, but at least it would be secure.

1.1 Problems with the Meaning of Models

Lampson's informal analysis contained the germ of an even more challenging problem, one that could threaten the meaningfulness of measuring covert channels. Lampson raised the question about what was "intended" by the system designers and what the system specifications meant with respect to covert channels.

Addressing Lampson's concern, Landwehr *et al.* articulated the problem of covert channels along the lines of meaningful specifications and intent. In contrast to a general-purpose operating system, they emphasize formally modeled security requirements, "assertions," that shall be enforced uniformly on the whole system. Specifically, they proposed to capture requirements for secure email-like communications for a system called Military Message Systems (MMS). [23] MMS functionality was similar to pre-Internet ARPANET email, with the addition of multiple levels of security.

By emphasizing the meaningfulness of requirements over mechanism or model, the authors presented an even more radical response to the classical security problem: one must start with the functional and security requirements of the application, they argued, and not with a generalized Reference Monitor mechanism (as in [3], [7], [8]) or even a generalized leak-free information flow model (as in [21], [22] [24], [19]).

We agree that the MMS approach was indeed more meaningful than the classical model of a secure computer. We suggest that they were successful not just because they developed specific "application" requirements, but because their requirements addressed the subject of secure communications – i.e. MMS – and explicitly addressed how messages were to be viewed, accessed, altered and sent by users within an identified context. Specifically, that document's ten requirement assertions addressed "shall nevers" using application-specific, explicitly defined terms and security language such as "is always," "no … can be," "can only," "can … only if", etc. By doing so the MMS approach explicitly addressed Lampson's informal concern with what was intended and what the specifications mean, a concern concurrently being taken up by evaluators [25]. Our approach is similar. We maintain that formal specifications treating the subject matter of secure communications best address the challenge of constructing meaningful models. We argue below that requirements concerning valuable communications in general cannot result in secure exchanges of information until the cryptography, protocol, parties, terms, laws and cultural forces have been formally addressed by specific requirements, at the very least by assumption.

1.2 Communicating Meaning

Both the information flow analyses by Rushby and the MMS analysis of application-specific communications that satisfy security requirements had the effect of reconnecting the field of communication security (COMSEC) with computer security

(COMPUSEC). These had previously been separated by the pioneers of TCSEC. This was one of several rediscoveries of Shannon's communication theory.

By 1981, Claude Shannon's work to provide a mathematical theory of cryptography had been rediscovered in unclassified COMSEC researches; Shannon's model is cited by "New Directions in Cryptography" [26] and by RSA asymmetric cryptosystems [27]. We read Shannon's model of "enemy" and "knowledge" as providing the "shall never" requirements to these papers, and Shannon's mathematization of cryptography were also revisited, revised, and used again to prove cryptographic security. [28] The discovery of public key cryptography led to interest in cryptographic communications protocols and their formal models. This led to the first formal models of parties to communications and, just as significantly, models of the attackers of communications systems. [29] Dolev and Yao's model of a stateful attacker, with powers and cunning far exceeding Shannon's "enemy", led to the emergence of protocol analysis, later nicknamed "Programming Satan's Computer" because of its diabolical subtlety. [30]

The rediscovery of Shannon was also of interest to others outside the world of communications security. In 1987, a paper titled "Covert Channel Capacity" [31] linked Shannon's information theory to the covert channel model proposed in [22] and [19]. Rushby and Millen referenced research conducted by SRI and Ford Aerospace dating back to 1976 that used Shannon-style information flow theory [22] [32] [33] [34]. By 1990, McLean had observed that "…[t]here is a general belief in the security community that the correct explication of security should be formulated in terms of Shannon-style information flow." [35] McLean's observation about the COMPUSEC security community's interest in Shannon proved to have predictive power. In the 1990's publications share a theme of Shannon-style information theoretic analyses and probabilistic security analyses, modeling security in terms of information flows.

In 1996 Kocher introduced the "side channel." [36] Side channel attacks may be compared to wiretap and TEMPEST attacks, since the attacker located at the perimeter of the system. [37] As cryptosystems have become increasingly popular and widely deployed, the relevance of formal models, confidentiality properties, and especially side channel attacks (both active and passive) have become more relevant to a wider audience. In recent literature, defenses against side channel fault injections commonly rely upon noise, error, entropy and coding theory introduced by Shannon in his communication theory. [38] Side channel attacks now explicitly utilize Shannon's information theoretic measures and revisit Shannon concepts such as entropy (for "guessing entropy" analyses) and knowledge (for gathering data that can be used for a template attack). [39] [40] [41] Additionally, several proposals to defend against passive side channel attacks use masking, scrambling, clock randomization and other randomization techniques used to obscure the emissions measurable by a side channel attacker. [42] [43] [44] [45]

Currently several security specialties borrow from Shannon's mathematizations. We propose that the shared "beliefs" of these "security communities" was that Shannon's mathematizations of communication were meaningful because they provided explanatory power and predictive certainty, which are typical measures of scientific progress.

1.3. Problems With the Level of Models

While Landwehr *et al.* gained meaningfulness by clearly specifying high level security requirements concerning allowable information exchanges of US classified information, they provided no explicit requirements concerning the secure construction of the system mechanism. The authors consider this a benefit: "Because the model avoids specifying implementation strategies, software developers are free to choose the most effective implementation." With the benefit of hindsight, especially in view of side channel attacks, we disagree that this approach is entirely advantageous. We suggest Landwehr *et al.* swung their corrective pendulum too far. Could an MMS-implementer satisfy the ten assertions in [23] equally well using GEMSOS or Berkeley UNIX? By allowing implementers freedom, in view of their statement, did [23] lose its grasp on the "shall-nevers"? Can sufficient conditions for a secure implementation of MMS be efficiently derived from [21]'s ten assertions and four assumptions?

With respect to covert channels, Rushby admitted that neither classical access control nor information flow formal verification techniques, as practiced, could be expected to eliminate Lampson's "confinement problem" with covert channels:

> Although the properties established by access control verification and by information flow analysis are undoubtedly important ones, it is not clear that they amount to a complete guarantee of security. *Both these verification techniques are applied to system descriptions from which certain "low level" aspects of system behavior have been abstracted away.* This is despite the fact that penetration exercises indicate that it is precisely in their handling of these low level details that many computer systems are most vulnerable to attack – and, consequently, that these are the areas where verification of appropriate behavior is to be most desired. [22]

Unfortunately, even formal specifications of operating systems had little to say about low-level covert channels or their meaning. This in turn led to the puzzles that Millen later documented concerning how to determine if a covert channel was a "real threat" – i.e. practically meaningful. [25]

Practical challenges await those who practice formal methods at a low level, i.e., for software object code or hardware netlist representations. One may begin to write an algorithm and prove a property, for example following Floyd, Hoare, Gries, etc., but the effort quickly becomes laborious for a set of algorithms comprising a system. This situation is exacerbated because verifying equivalence between formal specification and low-level realization is often nontrivial. Developing such an argument requires expertise in both the mathematical axioms and the idiosyncrasies of the target computer and toolset. The latter is subject to change every few years, leading to costly obsolescence.

But where, given the goal of a concise statement of requirements, could low-level aspects of system behavior be expressed? We suggest that this problem, which we take to be the core issue of Lampson's covert channel problem, lies in identifying

assumptions. Whereas Lampson originally thought of covert channels as "not intended for communication," Marv Schaefer, former chief scientist of the National Computer Security Center, once characterized covert channels as "system behaviors that surprise the system's developers." [46] The problem, as we see it, is lack of clarity and organization concerning what and on what occasion to specify the shalls and shall nevers more granularly, at a lower level, and when to assume that a meaning will hold as intended at a more fundamental level. This problem played out clearly in co-author McLean's subsequent criticisms of the meanings of aspects of the Bell-LaPadula model. [47] [48] McLean independently experimented on the popular Bell-LaPadula Basic Security Theorem (BST) and came to surprising conclusions:

> [BST] can be proven for security models that are obviously not secure. We conclude that the value of the BST is much overrated since there is a great deal more to security than it captures. Further, what is captured by the BST is so trivial that it is hard to imagine a realistic security model for which it doesn't hold. [47]

At the same time, we observe that the security requirements in [23] specified by Landwehr *et al.* could be understood to contain approximately the same level of ambiguity concerning system realization at a low level. Whereas the Bell-LaPadula model defines operations read, write, etc., which in their realization may be reversed in a way that violates a good definition of security, so also in [23] some assumptions might be abused. We take this to indicate that security requirements and models should be applied to low levels so as to eliminate covert channels and clarify intent. If, for example, one-way information flows called "intransitive noninterference" are critical to security, then clear requirements language (as in [49] [50]) and strong or redundant mechanism(s) should be specified at the most appropriate level of granularity so that predictive certainty may be achieved. Whether the initial assumption is "view," "read," or "*," we must complete our specification with shall-nevers to secure it at a low level.

1.4. Toward a Taxonomic Scope for Models

A 1993 paper identifies the specificity gap between general-purpose products and specific-use implementations in a fixed environment as a problem, especially as it pertains to certification. [51] In this paper, Payne *et al.* indicate that a TCSEC-styled requirements analysis resulting in a "Formal Security Policy Model" yields a focus which "is clearly too narrow." They propose a "comprehensive INFOSEC certification methodology" as a requirements analysis extended much more broadly, resulting in "INFOSEC policy," which is one that performs their novel methods of trade-off analysis over a system's development and implementation concerning requirements including location, use and cost-effectiveness, mission, threats, budget, etc. [51].

Also in 1993, Landwehr published a historical survey of computer security flaws and used it to illustrate his contributions to the topic of assertion-assumption security requirements analysis [52]. We agree with Landwehr's motivations (especially with his

observations in section 5, "Problems We Face") and many aspects of his proposed research agenda, including his description of future research:

> [We need] abstract, formal work on security modeling techniques that support composition and decomposition, methods and tools for reasoning about and finding flaws in cryptographic protocols, and concrete approaches for standardizing security function and assurance requirements…
>
> Imaginative approaches are needed to organize systems so that requirements for high assurance software are kept to a minimum. We need practical methodologies for exploiting formal methods on those portions of systems that unavoidably require high assurance, and we need methods to estimate or measure the security actually provided. [52]

Not surprisingly, Landwehr was co-author of a taxonomy of computer security flaws not long after the above-quoted paper was published. [53] This in turn was followed by attempts to reconcile a security taxonomy with a formally modeled taxonomy of concerns of dependability, [54] [55] and the more general model of the all-powerful network attacker. [56] [57] The work of Laprie [58] indicates that the scope of security concerns has been mathematized by a scientific community and can be ordered taxonomically.

The problem with prevailing security requirements lists, with or without taxonomy, is they lack explanatory and predictive power. Whereas system-specific security requirements analyses are directed at clearly identifying the "shall nevers" [59] [60] [61] [62], a majority of publications identify lists of "shalls" that may or may not apply to any one system, and may or may not adequately mitigate security threats. Numerous examples have been published, codified and maintained, including DCID6/3, Common Criteria, NIST SP800-53 and SP800-37, ISO IEC 17799 and the IEC 27000 family. While generalized lists of good practices are becoming more comprehensive, lists of vulnerabilities and successful attacks are too. As either list expands, so does the certification cost and liability for legitimate system owners. Incremental cost increases are not the root problem; the lists are characteristically descriptive and reactive, not predictive or preventative.

What is needed is an organization of what must be specified or assumed to express the comprehensive security requirements of a communications system. It should address all "those portions of systems that unavoidably require high assurance." [52] A set of non-overlapping mathematizations should organize the requirements' subject matter, each of which should be consonant with the present and future findings of a progressing scientific field. Formal models of comprehensive requirements would lead to communication security that is verifiable, validatable and meaningful; this would offer predictive certainty about system behavior even against determined attacks.

2 A Taxonomy of Epochs

We propose a taxonomy of securable subject matter, divided into levels of granularity we mark by "Epochs." Our taxonomy provides a comprehensive framework for identifying and analyzing meaningful security requirements and, in the complementary analysis, attacks. Following Rushby's observations about covert channels [22], we include low level concerns (algorithms, semiconductors and the physics of emissions at the system perimeter) at low-numbered levels. Following [51], we expand the scope to high levels including users and malicious users, exchanged communications, costs, and organizational security policies concerning future threats, mission and budget. In particular, this attack model shows how all attacks can be conceived of as communication attacks, but across a range of "Epochs." As all attack models directly or indirectly specify a security model, our attack model also identifies an abstract means for achieving comprehensive security. Lastly, our attack model sheds light on the proper use of formal models in information assurance – that is, it provides a framework for the construction of meaningful models of security.

Each Epoch circumscribes the limits of information available to some particular algorithm's run within a concretely realized system. In the table below we define nine Epochs by a scientific field, by runtime threats arising from information the Epoch accepts, and by information it provides to Epochs below (providing an "oracle" and/or state memory) and to Epochs above (providing a list of assumed-correct "assertions").

Table 1. Security Epoch Taxonomy

Epoch, Science and Range	I/O and Realization Threats (from Epoch below)	Oracle/State (govern below) Assertions (assumed above)
0. Perimeter Physics, from in-contact to nearby	**Environment**: particles and interactions; package, board, wires, case; network, connectors, I/O devices; site walls, location, geography.	**Data assertions**: valid; live; expected location; environmental conditions within specification
1. Timed Logic (TC) Semiconductors, from manufacturing processes to programming and microcode	**I/O Data**: invalid, replayed, simulated; not live, not functioning; relocated; harsh conditions; destruction **Program**: optimization, mistranslation; subverted synthesizer; program replacement, forgery, addition, trigger	**State:** Data samples, registered variables, memory cache **Architecture/address assertions**: wiring, arithmetic and logic operations, complex instructions, I/O analog to digital, codings, encodings, link protocol, errors, memories
2. Algorithm (P) Programming, State machines that run for-all inputs, from stateful circuits to operating system software	**I/O Data**: leaked, rerouted, forged, errors, buffer corruption; observed human interactions **Program addresses**: bypassed or subverted operations, instructions, memories, I/O	**State:** state variables, transition conditions, shared memories **Correctness assertions**: proven properties, big O notation, parallelization, first order type correctness, testing and verification coverage, performance metrics

Epoch, Science and Range	I/O and Realization Threats (from Epoch below)	Oracle/State (govern below) Assertions (assumed above)
3. Separation (NP) Program Analysis, State memories and outputs, from hard-to-guess to hard-to-corrupt	**I/O**: direction fault, data rerouted, key leaked, type confusion **Program**: cryptanalysis, predicate failure or confusion; no-operation, weak method, single point of failure, bypass, subversion, trigger	**Oracle:** state invariant, trace hyperproperty **Boundary assertions**: one-way function, statistical independence, encryption, randomness
4. Protocol (DY) Protocol Design, Input/output memories, sourced from unknown network-connected locations	**I/O**: Dolev-Yao stateful attacker, replay, rollback, MITM, state corruption **Program**: protocol failure or confusion	**Random Oracle:** pseudorandom keys, nonces, hashes **Session assertions**: data authentication, data integrity, cryptographic strength, subject authentication, key generation process
5. Subject (TT) Varies by Employment, Decision oracles, from sensors to data entry to purchase, command and control, and remote administration	**Input**: protocol bypass, subversion, trigger, storage fault **Output**: covert channel stream, malware, deception/ denial/degradation attempt, false alarm, attempt to exceed authority, audit evasion	**Subject identity oracle**: name, id, employment, criminal background check, credit report, membership, duties, education, skill, certification, legal powers **Subject intent assertions:** informed consent, skilled operation, intelligent decision, messages
6. Resource Requirements Analysis, for mission-relevant information from assets, plans and processes to legal terms	**Deception concerning**: assets, principals, partners, contracts, monetary instruments, possession, treaties, borders, weapons, classified data, legally protected data categories **Program**: disorganization, degradation, deletion or fraudulent insertion of resources	**Knowledge oracle:** set operations and set relational queries for all stored knowledge (e.g. a database) **Meaning assertions:** interpretable under law and policy, using: legal agreements, defined terms and categories, accurate records and measures, applicable conditions, standards and laws
7. Cost / Efficiency Varies by Principal, ownership concerns and negotiations.	**Deception concerning**: economic constraints such as: law and policy prohibitions, supply and demand, tax, tariff, policy, competition, risk assessment, strategy, financial expectations **Program**: same as for Epoch 6	**Accessibility oracle:** legal acceptability, policy allowance, authorization, purchase approval **Cost assertions**: accounting and audit reports (GAAS-GAAP), insurance/ actuarial rates, market rates
8. Culture Values, norms and human influences	**Estimates** for cultural outputs and value changes	**Value assertions**: Asset value, cost of ownership, and associated liabilities, risk and cost of loss. Risk analysis.

2.1 Justification for Taxonomy Features

Limits on the information available within an Epoch are derived from Alan Turing's notion of computability (the "Turing Machine") and his supposition that an algorithm could be supplied with an "oracle." Turing defines an oracle as the assumption that there exists some set of answers that can correctly answer a hard question, even a question which is noncomputable within a given logical system. Within our taxonomy

we say each security Epoch realizes a "Turing Degree" because it realizes one or more algorithms, and for any given run it takes inputs, gives outputs, and has access to the oracle at that Turing Degree. A system run accepts input from its physical environment located at the system's Perimeter Epoch, executes a first algorithm at the first Epoch which may in turn cause inter-Epoch interactions, and finally returns output to the physical environment.

Following Lampson's suggestion, we require that each Epoch must be confined during its algorithm run, meeting the two conditions of memorylessness and total isolation for function calls. [18] This affects the duration of a run of any Epoch's algorithm: an algorithm that can retain no state between runs will be bounded by its Epoch's complexity measure. Unfortunately, the field of complexity theory still retains a number of important unanswered questions, so our choice of specific complexity measures upon which we attempt to base claims for shall-nevers is not fully determined. At this time, we take that view that assertions of noncomputability may be justified by engineering solutions that combine the computationally infeasible with the physically infeasible, cost prohibitive, illegal, etc. So we selected the following complexity measures as Epoch-run-constraints in our taxonomy: Boolean circuits (approximately TC), polynomial time (P), nondeterministic/guessing polynomial time (NP), stateful attacker-secure (e.g. secure for all protocol sessions granted the Dolev-Yao stateful attacker), an intelligent oracle (passes the Turing Test), and the set of all oracles that store validated relational knowledge tuples. We note that a recursive function (or state machine) deals with a state tuple twice: accepting it as input and returning it as output. We require that for any Epoch's nontrivial recursive function, its state memory be retained across multiple function calls by the next higher Epoch, and that the higher Epoch must contain any security-relevant state transition guard condition evaluator.

We augment each Epoch's first order Peano Arithmetic axiom set with some axioms from a specific scientific field. These additional axioms may be identified with reference to some pioneering researchers, some theory and accepted mathematization, published expressions in some notation, and giving rise to some conjectures, theorems, etc. An Epoch's algorithm must be mathematized under the axioms of some notation (a language), fundamental constants (a kernel), equality (a relation), closure and functions (a universe). As Turing [63] and others have showed, these can be subjected to ordinal numbering schemes. To run the algorithm, each Epoch receives its respective portion of Turing's Program Tape, which may be treated as an assumption or oracle. Our Epochs' algorithms are expressed using first order arithmetic, and our hierarchy arises out of this limitation.

The taxonomy, but not any one Epoch, can express higher order predicates. McLean asserted that first order predicates, or properties, simply cannot express notions of security [64], observing that Alpern-Schneider properties are "trace sets" and known "possibilistic" security properties are "sets of trace sets". Improving upon McLean's periphrasis, Clarkson and Schneider call these "hyperproperties." [65] [66] Others confirm that "a *hyperproperty* is a second-order predicate over system execution traces," [67] which like the halting problem simply cannot be expressed within a first

order system (e.g. Peano Arithmetic). Regarding higher-order expressions in our proposed hierarchy, we recall that Turing first proved the Entscheidungsproblem was not computable by a Turing Machine [68], and that next he intended to overcome Gödel's incompleteness (noncomputability) by introducing an Oracle Machine, which defined relative computability. Turing set up the problem this way:

> The well-known theorem of Gödel (1931) shows that every system of logic is in a certain sense incomplete, but at the same time it indicates means whereby from a system *L* of logic a more complete system *L'* may be obtained. By repeating the process we get a sequence *L, L1 = L', L2 = L1'* … each more complete than the preceding. [63, p. 161]

Turing went on to postulate his oracles as that which must be added to any axiomatic system *L* in order to form *L',* now called a Turing jump for the halting problem. [69] Our taxonomy provides a sequence of nine logical systems of increasing Turing degree, each of which is an oracle machine or "hypercomputer" [70].

We locate security assertions at the next-higher Epoch for that which cannot be computed or output by some relatively-lower Epoch. At the higher Epoch, one may interactively specify a state space such that all defined state values are validatable and can be constructively mapped to and from intended inputs. Next one may specify the state transition decision function such that all allowed state transitions refine a security hyperproperty meaningful for the state space at the higher Epoch. We consider any such state space *S* with its state transition relation *S x S* "secure by design", and the relation is a security oracle. Recall that, practically speaking, system state *S* is a collection of memories that persists between inputs having accepted some prior input. We observe that a higher-Epoch state stores, albeit in a *compressed* format determined by lower-Epoch input processing, prior inputs to the run at this Epoch observed up to the present. The lower Epoch's effects on a state tuple can express a first order property on its inputs. The higher Epoch decides if the state transition is allowed. Thus any two adjacent Epochs can define a security property as a selection among "sets of trace sets." When an Epoch's state transitions and/or security oracle is located in a higher Epoch we say that Epoch is "governed" with respect to that state relation or oracle. The rightmost column of Table 1 provides examples.

Regarding the development lifecycle, each Epoch may be analyzed, specified, designed, implemented and tested as a module. Such a module may be developed over time, and may assume inputs and oracles. Treating an Epoch algorithm as a module can in turn facilitate the use of simulated oracles to test the algorithms. During the development timeframe, one may interactively define the intent and validation procedures for the specification, even to the point of adjusting the state space and state transition relation specification to better fit the model to the meanings the system shall, and shall never, realize. We advocate tuning an algorithm's governance thus.

After governing an Epoch and implementing it, one must examine its assumptions and inputs for threats. For examples, see the center column in Table 1. Unlike algorithm verification, the threat analysis of assumptions presents the challenge not only of

enumerating predicative assumptions or intuitions, but also of engineering the system to reduce the size of this set. We advocate fitting, shrinking and constraining the state space so that it only includes those tuples at the intersection of what can be output from the lower Epoch, what can be validated, and what the intended security property means. While justifiable validation methods may depend upon a small number of assumptions, efforts to find improved validation methods that use fewer assumptions lead to more direct and certain results. While we know of no codified basis set of information security assumptions, in our Introduction we have presented one effective basis of assumptions from our survey of historical proposals such as Turing's, Anderson's, Lampson's, and now presented as our taxonomy of Epochs. Based on that analysis, we suggest the following methods to engineer a system realization for which such assumptions may be valid.

Implementers must introduce some method of separation [21] or confinement [18] that can practically prevent information leakage between Epochs. Otherwise, the hierarchy will collapse and assertions about the limits of what is computable will not hold. This is an essential point, especially for COMSEC key secrecy concerns, and it becomes increasingly difficult to achieve as the scope of the system is enlarged to include larger spheres of compatibility and connectivity. We use the statements "No Other Changes" and "Type Correctness of Realization" in our threat analysis of every input and assumption required by security-critical Epoch algorithms. "No other changes" is straightforward but challenging: while the system stores or moves some secret value, how can I know that there is not some background process that discloses the secret? Such concerns are especially relevant to side channels. Regarding "type correctness," our concern is primarily that when we invoke an algorithm at a lower Epoch, that invocation should return valid results. Our threat analysis considers possibilities such as: has it been subverted, bypassed, disabled, or renamed? We consider the possibility of triggered type correctness failures, where only magic values known only to the attacker can trigger bad behaviors.

The latter concerns led us to want to mathematically derive the shall-nevers of our system from the shalls. We attribute this intuition to John von Neumann, who asserted: "If one has really technically penetrated a subject, things that previously seemed in complete contrast, might be purely mathematical transformations of each other." [71] In our taxonomy, we propose eight such subjects, discounting a ninth that cannot be adequately mathematized. We assume these Epochs will progress toward a fully-developed theory, mathematization, and notation to the point where a set of "shalls" can be mathematically transformed into a "shall-never" security requirement.

2.2 Assumptions of the Taxonomy

With respect to framing philosophical assumptions, especially the justifications of mathematizations with phenomena, our approach is indebted to Edmond Husserl's *Crisis of the European Sciences* [72]. Husserl used the term "epoche" (from the Greek word εποχή, meaning "suspension of judgment; cessation") to denote the sense of being a prisoner of one's own experience. We use the term "Epoch" in a similar way, though as a noun, such that upon entering an Epoch, one becomes a prisoner to that

Epoch. One can hardly see beyond the Epoch, except to make assertions or assumptions. As a prisoner, one's power is reduced to only make assertions at that Epoch using the notation and system of that Epoch. Nevertheless, one must look outside the Epoch for its meaning (above) and its realization and inputs (below). The table below lists our assumptions concerning Epochs in our taxonomy.

Table 2. Taxonomy Assumptions

1. Within an Epoch
   - 1.1. Each Epoch is located within exactly one system perimeter; this defines communication between systems.
   - 1.2. Each Epoch is separated from other Epochs; this defines internal communications such that information cannot flow except through oracles.
   - 1.3. The science at each Epoch must provide some notation to encode an algorithm and variables, e.g. by an ordinal numbering or properties, which must admit to a measure of certainty.
2. Between Epochs
   - 2.1. Beyond the expressive power of any one Epoch's notation and mathematization lies a different Epoch.
   - 2.2. Realization assumptions to any one Epoch arise from security requirements governing the selection of Objects contained within a lower Epoch.
   - 2.3. Semantic assumptions made at any one Epoch arise from security requirements governing the selection of Objects contained within a higher Epoch.
3. Beyond Epoch Specifications
   - 3.1. Epoch 8 at culture does not admit to security requirements, for one cannot select which Objects of culture will take root, become obsolete, or meaningfully change the universe within a lower Epoch.
   - 3.2. Uncertainty from Epoch 8 at culture limits the certainty of all lower Epochs. Acts of culture may only be estimated, e.g. by Moore's Law, Robert's Law [94] Amdahl's Law, etc.[1]
   - 3.3. Epoch 0 at the system perimeter may be partially secured by cultural acts, e.g. architecture, military science, user training, motivations, etc., but the science of Physics does not admit to constraint by security requirements.
4. Fundamental Philosophy of Science
   - 4.1. Scientific advance, or progress, will yield explanatory power over historical observations and theories.
   - 4.2. Progress will yield predictive power approaching the limit of certainty within the Epoch.
   - 4.3. Every scientific field of study may be advanced or repressed by culture, and we assume that such cultural influences cannot be secured.

2.3 The Scope of a Secure System

We have shown above that, in this taxonomy, no Epoch can express its own security property, compute its own security condition, or assure its realization assumptions. Next, we explain why we resist declaring any algorithm realization below Epoch 7 "secure." To illustrate why, we consider some hypothetical systems whose security analysis stops short at some maximum Epoch less than 7, while granting their perfect realization, that is, every assumption of a higher Epoch has been satisfied in system assertions at lower Epochs. In each case, we consider how the attacker may "wrap" the short-maximum Epoch's implementation and defeat the meaning of security for the lower Epochs.

For example, can one meaningfully declare an Epoch 3 implementation of the Advanced Encryption Standard (AES) algorithm secure? For AES the question of key secrecy and key entropy (an oracle provided by Epoch 4) remains unexpressed and unsolved. We believe an Epoch 3 AES algorithm cannot be secure, since that algorithm may be run millions of times under side channel attack or characterization. Additionally, it cannot be secure since that implementation may be used to send garbage data under infinite timing schemes, thereby emitting messages in some on/off coding instead of using the algorithm as intended, to emit ciphertext that may be decrypted by some intended party. Implementing an AES round function in a semiconductor as a complex operation available for use at Epoch 1 only exacerbates the problems.

Moving up the taxonomy, but only to Epoch 4, is not sufficient. Suppose at Epoch 4 we implement a cryptographic key exchange protocol such as Needham-Schroeder-Lowe or the Transport Layer Security standard. Lacking Epoch 5, this begs the question: who is the attacker or who is the subject? Considering that the primary goal of Epoch 4 is to eliminate opportunities for the Dolev-Yao attacker (DY) to interfere with the valuable communication, but still omitting Epoch 5, we allow DY to simply connect anonymously, which allows a man-in-the-middle attack to render all of Epoch 4 irrelevant.

Stopping at Epoch 5 has a similar, taxonomy-predicted effect: at Epoch 5 and below the runs of the system are initiated by a subject's decision. These runs may be short and subject to timing attacks, even if human subjects are assumed to be benevolent. We assert that the meaning-assumption of benevolence implies a deep and thorough understanding, on the part of the subject, of the nature and workings of the enterprise and also the ability to identify suspicious activities that are not a part of the proper functioning of any part of the enterprise. Even if such an assumption held for all human subjects, the taxonomy helps us understand and model this benevolence requirement as an up-Epoch assumption, subsuming Epoch 6, the enterprise information and processes, Epoch 7, the efficient and lawful operation of the enterprise, and Epoch 8, the good functioning of the enterprise within its cultural milieu. Without resorting to assuming the security of all higher Epochs, we assert that a malicious, misguided or malware subject at Epoch 5 can encode the totality of all Epoch 6 knowledge using garbage messages that only serve to provide on/off signals to an accomplice; there may be an infinite variety of timing schemes to do so. Because these garbage on/off signals may be observed and decoded by any wiretap accomplice at any location on

the network, this attack degrades all of the value contained in Epoch 6 at the trivial cost to the attacker of placing a wiretap containing a simple decoder.

In this taxonomy each governed Epoch requires some oracle at a higher Epoch to answer a meaning-question such as, "can I do this and remain secure?" The Epoch above 6, Epoch 7 Cost, governs by securing some information as costly to leak for unauthorized use, or costly for unauthorized users to contaminate. Obviously it would be meaningless to "secure" some information that was readily available in the public domain. But even this conclusion has been suggested by the taxonomy, where the oracle at Epoch 8 Culture informs system designers what is cheap or valuable.

If the answer to any meaning-question is ignored, or a governing oracle is subverted, security failures can be expected occur. So we should be motivated to engineer the oracles using proofs of correctness and reduced-assumption refinements in these higher Epochs. At the same time, the oracle engineer must make realization assumptions regarding the correct implementation and properties of algorithms in a lower Epoch. But the threat here is that these realizations may be subverted. Considering threats both above and below, a finite taxonomy helps make the problems tractable.

3 An Example of a Meaningfully-Secure Formal Model

Rather than securing an algorithm, or any one Epoch, our taxonomy is useful to secure valuable communications. Examples include: conducting business over the Internet, communicating arms reduction treaty test results [73], military command and control messages, control of remote-piloted systems, control of infrastructure systems, or remote surgery. As an example of how to use the taxonomy, we illustrate how to create a more meaningful formal model within each Epoch for the classic MMS. [74] In the table below, we summarize the terms used in the MMS formal model based on the section "Correspondence to the Informal Model" of [23] as follows:

Table 3. Requirements Analysis for Security Epochs

Epoch	**Informal Definition** [23]	**Formal Notation** [23]	**Formal Security Specification** [23]
7	*Inefficiencies, user efficiency and problems discussed in sections 1-3.*	--	--
6	Object,Container,Entity (ID, Classification) Operation (inputEntityID, outputEntityID) Message (fields, relUserID, DRAFT/REL)	ES OP VS, TY	*State Set Existence Assumptions*
5	User,UserID (Role, Clearance) Access (User, Operation)	US, UI, RL, L AS(ES)	*User Assumptions {1,2,3,4}*
4	None	--	--
3	None	--	--
2	Assertion {2, 4, 7} Assertion {1} Assertion {6} Assertion {8,9,10} Assertion {3,5}	Def {1, 2} Def {5} Def {8} Def {9,10,11} Def {6,7}	*secure state* *access secure* *translation secure* *set,dg,rel-secure* *copy,CCR-secure*
1	None	--	--
0	None	--	--

Landwehr *et al*. present their security requirements in roughly the order of Table 3, and we follow their order in our illustration below. They provide commentary on this model in a retrospective, and confirm our suspicious that the preferable notation for Epoch 6 are Codd's normalized database relations [74], which provide a formal model intended to provide semantic organization to properties [75].

3.1 Eliminating the Existence Problem for Modeled Sets

Epoch 6: Following [23], we begin at Epoch 6, Resource, noting that Epoch 7, Cost, was discussed informally in the opening sections of the MMS paper. In our taxonomy at Epoch 6, algorithms convert data into culturally meaningful information terms, granted some integrity property or validation test. As late as 1989, there has been little agreement about how to measure data integrity as can be seen from records of NIST workshop SP500-168. More recent work proposed defining integrity with respect to "leakage" and "contamination." [76]. Once the system algorithmically validates and meaningfully converts data into terms as required at Epoch 6, then independently functioning legal systems may apply their laws, terms and conditions to the information. So, the security notation at this Epoch is law governing the Exchange.

Analyzing the MMS specification taxonomically, we find its modeled Epoch 6 terms were only assumed to exist and not specified constructively enough to be realized. One may contrast the MMS specification's assumptions with how the Peano Axioms construct the set of natural numbers. The MMS specification merely assumes a set of Users or a powerset of classified sequences of bits as system inputs, which fails to provide a verifiable or validatable definition. While formal models often assume set existence, demonstrating one-to-one correspondence between a modeled set's elements and some attacker's choice of input data can pose an intractable challenge. We believe that existence assumptions, when not worked out as constructive, Epoch-specific security requirements, give rise to opportunities for the attacker to introduce type errors and other confusions such that the system misinterprets the attacker's choice of data as the wrong type of information. We propose relocating these existence assumptions into the definition, science and notation of Epochs, specifically by relying upon the kernel of that science wherever possible to supply sets and elements. This relieves the system specification of the burden of creating and maintaining these constants.

By conducting taxonomic requirements analysis at Epoch 6, we receive from the science of law a large and living kernel of defined terms. For MMS, the relevant law governing the contemplated exchange of classified information is US law, e.g. espionage laws and the most recently issued executive order. Terms given in law, i.e. Section 6.1. Definitions [77], must not be redefined because law supersedes. In our analysis, the MMS model's distinction between modeled Message, Object, Container and Entity [74] is not required as the legal term "information" applies the same to all three. On the other hand, the MMS term "Classification" should be specified more granularly, by the following attributes from Section 6.1. of [77]:

- Category (a-g),

- Duration (a-c),
- Identification and Markings (a-g), (various)
- Prohibitions and Limitations (a-e) (an attestation)
- Derivative Classification rules may be required, and therefore follow process (a-d) and add attributes including listing of source materials, etc. [77]

Additionally, a conservative read of the legal definitions indicates MMS should allow the user to identify the security mechanisms, e.g. encryptions and keys, used to the source information and draft messages.

Epoch 7: Taxonomic analysis at Epoch 7, Cost, for the MMS might begin by asking which oracle must govern Epoch 6 resources. Since Epoch 6 analysis revealed defined terms in the notation of law governing the exchange of these defined terms, Epoch 7 must realize an oracle for the prohibition of the disclosure of classified information. We take the Bell-LaPadula confidentiality property to be an adequate model of this law.

In the taxonomy at Epoch 7, data can be correctly associated with monetary charges and liabilities accruing because of the timeliness, production, use, misuse, sale, and loss of information. The information might directly represent monetary units or legal actions or government papers, or might indirectly represent counts, measures, invoice, audit and/or accounting information. Analysis at Epoch 7 reveals efficiency, waste, or opportunity, and may be used for computer-aided decision systems. Its mathematization depends heavily upon the terms afforded it by Epoch 6, and is sensitive primarily to relevant inputs from higher Epochs. Applying taxonomic analysis at this level to the MMS might result in reducing correct message classification assumptions on human subjects.

Epoch 5: At Epoch 5, Subject, we find MMS assumptions regarding human Users not specifications. Taxonomic analysis indicates that the system must include algorithms to control what decisions and actions Subjects may request of the system. There exists an Epoch 5 science of authentication pertaining to public key infrastructure (PKI), multiple factors of authentication, location, etc., more generally referred to as Identity Management. This science not only addresses the initial identification, enrollment and administration infrastructure, but also challenges such as key lifecycle and proof of cryptographic integration with respect to some Epoch 4 protocol. Requirements analysis at Epoch 5 may identify an existing centralized database repository acting as a single source of truth for all authenticatable Parties, which is a best practice (axiom) of Identity Management. If a centralized repository already exists, or if requirements analysis now indicates one should be stood up, that repository system would eliminate Epoch 5 set existence assumptions, i.e. US, UI, RL, L and possibly even the modeled Access function AE(ES). If so, then security specifications at this Epoch must include an argument for the secure integration of that repository. For MMS, Epoch 5 requirements must specify an authentication oracle for a cryptographic protocol but delegate information confidentiality and integrity assumptions to Epoch 4, Protocol. At Epoch 5 in the MMS, different Subjects have access to differing classified Resources at Epoch 6. To better govern this multi-level secure feature, the security engineer could

suggest use cases and workflows that also reduce the cost in terms of time and convenience to the User, adding these to Epoch 7 specifications.

3.2 Completing a Security Specification

Without a taxonomy that describes adjacent Epochs it is difficult to measure omissions from a security specification. For example, Table 2 above shows that the MMS security specification lacks detail concerning Epochs 0, 1, 3, and 4 and does not explicitly mention any assumptions concerning these Epochs. We believe a security specification remains ambiguous at best when it is silent concerning any one Epoch. Indeed, granted a formal specification, one is tempted to rely on a formal theorem prover or other formal methods tool to point out any unsupportable assumptions. However, in our experience formal methods tools cannot admit to the possibility of an attacker – or even a simple bug – within any omitted Epoch.

Epoch 4: In taxonomic analysis of the MMS at Epoch 4, Protocol, we receive from the science a model of the all-powerful Dolev-Yao attacker [56] [57]; and that it operates as a wrapper function. Shannon's model applies: "this cryptogram is transmitted to the receiving point by a channel and may be intercepted by the 'enemy'" [78]. Since Shannon, the science has advanced. The Dolev-Yao attacker is stronger and more effective at breaking protocols because it can remember prior good messages and can inject these and other malicious messages. For a survey on notation at this Epoch, see "Ordering from Satan's Menu…" [79] Fulfilling our expectation for the science at Epoch 4, researchers using formal models of protocols have demonstrated that the Dolev-Yao network attacker can be mathematically derived from any MSR 2.0 protocol specification. [80]

Epoch 3: MMS does not specify any requirements at Epoch 3, Separation. Under taxonomic analysis at Epoch 3, the science provides Shannon's model of perfect cryptography. Cryptographic separation aims to force attackers located outside the perimeter of the system into a state of zero mutual information (i.e. perfect statistical independence from the sender and the receiver). The science proposes algorithms, and cryptanalysis identifies weaknesses compared to algorithms with ideal cryptographic functions. Cryptographic algorithms at Epoch 3 typically give outputs believed to be reducible to "hard" problems – including those under the broader conjecture that P≠NP. Other Epoch 3 measures can be effective to achieve intra-system separation, e.g. using a hook-up theorem proven by [81] [82] [83], or composition suggested in [64], etc., granted these authors' axioms, assumptions and the strength of the theorem proved. A system debugger, capable of reading and writing with respect to the system's unencrypted state, provides a good approximation of this Epoch's attacker.

Due to space constraints, our MMS example concludes here. For a general description of Epoch analysis at Epochs 2, 1 an d 0, please see Appendix A.

As seen by our taxonomy, MMS omits specification at Epochs 4 and 3, so one may expect that no formal model of Epoch 2 can secure the system against attacks at 3 or 4. Moreover, because Epochs 5, 6 and 7 are not taxonomically connected down to

Epoch 2, we assert that an attacker at either Epoch 3 or 4 has unmitigated ability to manipulate the system. Similarly, because the specification in our illustration does not address Epochs 0 and 1, all manner of hardware- or wire-level attacks can succeed against the system, including manipulating the classification level Message attribute.

Of course this analysis benefits from hindsight; in 1984 the fields of side channel analysis and cryptographic protocol analysis were much less developed than at present. Nevertheless, one may now reasonably expect attackers to act as a wrapper function around every Epoch, and worse, inside every Epoch if they are not by force of system assertion shut out from penetrating the Epoch and disrupting the specified Object selections.

4 Conclusion

Our history chronicles, to some extent, the tremendous amount of high-quality work in computer and communications security over the past 40 years. It has also shown that, despite the undoubted importance of formal models to certain areas of computer and communications security, there is no consensus about what subject matter must be specified to achieve comprehensive security. Our contribution involves ordering specifications' subject matter, and articulating the meaningful limits of the formal models of communications security.

A challenge in building a secure system is to identify, locate and justify assumptions and theorems in a system that secures some valuable communications, even against an attacker who tries to force security failures. The value of applying this taxonomy is that by emphasizing information flows across Turing degrees it reveals to the security engineer the assumptions, threats and oracle-governed-hyperproperties in the system design. Security engineers that leverage this taxonomy can eliminate vulnerable assumptions and better utilize noncomputability and nonbypassibility to assure targeted shall-nevers.

Our taxonomy of Epochs is an attempt to clearly demarcate the different layers of computer and communication security and, as such, the boundaries and interdependencies by which the various formal models may be used to specify security. Each Epoch in our taxonomy aids the security requirements analysis effort by identifying a science which progresses, a formal notation with its kernel and expressive scope, and a range of security assertions. We note in closing that if our concept of taxonomy and our observations about it is correct, we will have articulated a framework upon which a multi-level secure communications system can be built and attain to certainties measured by a taxonomy of living sciences.

This material is based upon work supported by the US Navy-SPAWAR under Contract No. N00039-11-C-0006. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the US Navy-SPAWAR.

Works Cited

[1] L. Lamport, "Proving the Correctness of Multiprocess Programs," *IEEE Trans. on Software Eng.*, Vols. SE-3, no. 2, pp. 125-143, 1977.

[2] The Rand Corporation, "Security Controls For Computer Systems," Washington, DC, 1970.

[3] J. P. Anderson, "Computer Security Technology Planning Study," Bedford, Massachusetts, 1972.

[4] G. Pottinger, "Proof Requirements in the Orange Book: Origins, Implementation, and Implications," Washington, DC, 1994.

[5] D. E. Bell, "Looking Back at the Bell-La Padula Model," in *Proceedings of the 21st Annual Computer Security Applications Conference*, Tuscon, Arizona, 2005.

[6] C. E. Irvine, "The Reference Monitor Concept as a Unifying Principle in Computer Security Education," in *Proceedings of the IFIP TC11 WG 11.8 First World Conference on Information Security Education*, 1999.

[7] D. E. Bell and L. J. La Padula, "Secure Computer Systems: Mathematical Foundations," Springfield, Virginia, 1973, reconstructed 1996.

[8] D. E. Bell and L. J. La Padula, "Secure Computer System: Unified Exposition and Multics Interpretation," *Technical Report*, vol. 44, no. 5, p. 134, March 1976.

[9] J. Saltzer and M. Schroeder, "The Protection of Information in Computer Systems," in *Communications of the ACM*, 1974.

[10] D. E. Denning, "A Lattice Model of Secure Information Flow," in *Communications of the ACM*, 1976.

[11] G. Popek and C. Kline, "Issues in Kernel Design," in *Proceedings of the National Computer Conference*, 1978.

[12] C. E. Landwehr, "Formal Models for Computer Security," *Computing Surveys*, vol. 13, no. 3, pp. 247-276, September 1981.

[13] R. Kemmerer, "Testing Formal Specifications to Detect Design Errors," in *IEEE TRANSACTIONS ON SOFTWARE ENGINEERING*.

[14] O. Saydjari, "LOCK: An Historical Perspective," in *18th Annual Computer Security Applications Conference*, 2002.

[15] R. E. Smith, "Cost Profile of a Highly Assured, Secure Operating System," *ACM Trans. Inf. Syst. Secur.*, vol. 4, no. 1, pp. 72-101, 2001.

[16] National Computer Security Center, "Final Evaluation Report for the Gemini Trusted Network Processor," 1995.

[17] R. Smith, Multilevel Security, http://www.cryptosmith.com/book/export/html/42, 2006.

[18] B. W. Lampson, "A Note on the Confinement Problem," in *Communications of the ACM*, 1973.

[19] J. Goguen and J. Meseguer, "Security Policies and Security Models," in *Proceedings 1982 IEEE Symposium on Security & Privacy*, Oakland, CA, April 1982.

[20] S. Zander, P. Branch and G. Armitage, "A survey of Covert Channels and Countermeasures in Computer Network Protocols," *IEEE Communications Surveys*, vol. 9, no. 3, pp. 44-57, 2007.

[21] J. Rushby, "Design and Verification of Secure Systems," in *8th ACM Symposium on Operating System Principles*, Pacific Grove, California, 1981.

[22] J. Rushby, "Proof of Separability: A Verification Technique for a Class of Security Kernels," in *International Symposium on Programming*, Turin, Italy, 1982.

[23] C. E. Landwehr, C. L. Heitmeyer and J. McLean, "A Security Model for Military Message Systems," in *ACM Trans. on Computer Systems Vol. 9*, 1984.

[24] J. Rushby and B. Randell, "A Distributed Secure System," *IEEE Computer*, pp. 55-67, 1983.

[25] J. Millen, "20 Years of Covert Channel Modeling and Analysis," in *Proceeedings of the 1999 IEEE Symposium on Security and Privacy*, 1999.

[26] W. Diffie and M. E. Hellman, "New Directions in Cryptography," *IEEE Transactions on Information Theory,* Vols. IT-22, no. 6, November 1976.

[27] R. Rivest, A. Shamir and L. Adelman, "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems," in *Communications of the ACM* , 1978.

[28] Goldwasser and Micali, "Probabilistic Encryption," *J. Comput. Syst. Sci. 28(2),* pp. 270-299 , 1984.

[29] D. Dolev and A. C. Yao, "On the Security of Public Key Protocols," in *IEEE Transactions on Information Theory*, 1983.

[30] R. Anderson and R. Needham, "Programming Satan's Computer," *Computer Science Today,* pp. 426-440, 1995.

[31] J. K. Millen, "Covert Channel Capacity," in *1987 IEEE Symposium on Security and Privacy*, 1987.

[32] J. K. Millen, "Security Kernel Validation in Practice," vol. 19, no. 5, pp. 243-250, 1976.

[33] Ford Areospace and Communicatrions Corporation, "KSOS Verification Plan," Palo Alto, California, 1978.

[34] R. Feiertag, "A techniquie for Proving Specifications are Multilevel Secure," Menlo Park, California, 1980.

[35] J. McLean, "Security Models and Information Flow," in *Proceedings of the IEEE Symposium on Security and Privacy*, 1990.

[36] P. C. Kocher, "Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems," in *Proceedings of the 16th Annual International Cryptology Conference on Advances in Cryptology* , 1996.

[37] "NSA Tempest Certification Program," [Online]. Available:

http://www.nsa.gov/applications/ia/tempest/index.cfm. [Accessed 25 May 2012].

[38] M. Medwed, "Protecting Security-Aware Devices Against Implementation Attacks," Graz, Austria, 2010.

[39] F. Mace, F.-X. Standaert and J. Quisquater, "Information Theoretic Evaluation of Logic Styles to Counteract Side-Channel Attacks," in *CHES 2007 Lecture Notes in Computer Science*, Vienna, Austria, 2007.

[40] F.-X. Standaert, E. Peeters, C. Archambeau and J. Quisquater, "Towards Security Limits in Side-Channel Attacks," in *CHES 2006 Lecture Notes in Computer Science*, Yokohama, Japan, 2006.

[41] F.-X. Standaert, T. Malkin and M. Yung, "A Unified Framework for the Analysis of Side-Channel Key Recovery Attacks," in *Proceedings of Eurocrypt 2009*, Cologne, Germany, 2009.

[42] J. Golic and C. Tymen, "Multiplicative Masking and Power Analysis of AES," in *CHES 2002 Lecture Notes in Computer Science*, 2002.

[43] J. Coron and I. Kizhvatov, "Analysis and Improvement of the Random Delay Countermeasure of CHES 2009," in *Proceedings of CHES 2010, Lecture Notes in Computer Science*, 2010.

[44] J. Schmidt, T. Plos, M. Kirschbaum, M. Hutter, M. Medwed and C. Herbst, "Side-Channel Leakage Across Borders," in *CARDIS, Lecture Notes in Computer Science*, 2010.

[45] T. Guneysu and A. Moradi, "Generic Side-Channel Countermeasures for Reconfigurable Devices," in *CHES 2011*, Nara, Japan, 2011.

[46] J. McHugh, "Covert Channel Analysis," in *Handbook for the Computer Security Certification of Trusted Systems*, Naval Research Laboratory, 1995.

[47] J. McLean, "A Comment on the "Basic Security Theorem" of Bell and LaPadula," *Information Processing Letters*, vol. 20, pp. 67-70, 1985.

[48] J. McLean, "Reasoning about Security Models," in *Proceedings of the Symposium on Security and Privacy* , 1987.

[49] J. Rushby, "Noninterference, Transitivity, and Channel-Control Security Policies," SRI International, Menlo Park, California, 1992.

[50] A. W. Roscoe and M. H. Goldsmith, "What is Intransitive Noninterference?," *In Proceedings of the 12th IEEE workshop on Computer Security Foundations (CSFW '99).*, pp. 228-, 1999.

[51] C. N. Payne, J. N. Froscher and C. E. Landwehr, "Toward a Comprehensive Infosec Certification Methodology," in *Proceedings of the 16th National Computer Security Conference*, Baltimore, Maryland, 1993.

[52] C. E. Landwehr, "How Far Can You Trust A Computer?," in *Proceedings 12th International conference on Computer Safety, Reliability, and Security*, 1993.

[53] C. E. Landwehr, A. R. Bull, J. P. McDermott and W. S. Choi, "A Taxonomy of Computer Program Security Flaws, with Examples," 1993.

[54] C. Meadows and J. McLean, "Security and Dependability: Then and Now," in *Proceedings of Computer Security, Dependability, and Assurance*, Washington,

DC, 1999, pp. 166-170.

[55] C. Meadows, "Applying the Dependability Paradigm to Computer Security," *In Proceedings of the 1995 New Security Paradigms Workshop,* 1996.

[56] C. Meadows, I. Cervesato and P. Syverson, "Dolev-Yao is no better than Machiavelli," in *First Workshop on Issues in the Theory of Security*, 2000.

[57] I. Cervesato, "The Dolev-Yao Intruder is the Most Powerful Attacker," in *Proceedings of the Sixteenth Annual Symposium on Logic in Computer Science*, 2001.

[58] J.-C. Laprie, Dependability: Basic Concepts and Terminology, Springer-Verlag, 1992.

[59] B. Schneier, Secrets and Lies: Digital Security in a Networked World, New York, NY: John Wiley & Sons, 2000.

[60] C. Heitmeyer, M. Archer, E. Leonard and J. McLean, "Formal Specification and Verification of Data Separation in a Separation Kernel for an Embedded System," in *Proceedings of the 13th ACM conference on Computer and communications security*, 2006.

[61] M. Bishop, Computer Security: Art and Science, Boston, MA: Addison-Wesley Professional, 2002.

[62] N. R. Mead, "Security Requirements Engineering," 2010.

[63] A. Turing, "Systems of Logic Based on Ordinals," *Proceedings London Mathematical Society,* Vols. 2-45, no. 1, p. 161–228, 1939.

[64] J. McLean, "A General Theory of Composition for Trace Sets Closed Under Selective Interleaving Functions," in *Proceedings of the 1994 IEEE Symposium on Research in Security and Privacy*, 1994.

[65] M. R. Clarkson and F. B. Schneider, "Hyperproperties," *J. Comput. Secur.,* vol. 18, no. 6, pp. 1157-1210, 2010.

[66] M. R. Clarkson, Quantification and Formalization of Security, New York, 2010.

[67] D. Clarke and D. Milushev, "Towards incrementalization of holistic hyperproperties," in *Proceedings of the First international conference on Principles of Security and Trust (POST'12)*, Berlin, Heidelberg, Springer-Verlag, 2012, pp. 329-348.

[68] A. Turing, "On Computable Numbers, with an Application to the Entscheidungsproblem," *Proceedings of the London Mathematical Society,* vol. 42, pp. 230-65., 1936.

[69] R. Shore and T. Slaman, "Defining the Turing jump," *Mathematical Research Letters,* vol. 6, no. 5-6, pp. 711-722, 1999.

[70] Proudfoot and Copeland, "Alan Turing's forgotten ideas in computer science," *Scientific American,* April 1999.

[71] A. Bródy, Proportions, Prices, and Planning, 1970.

[72] E. Husserl, Crisis of the European Sciences, Evanston, Illinois: Northwestern University Press, 1970.

[73] G. J. Simmons, "The Prisoners' Problem and the Subliminal Channel," in

Proceedings of Crypto 1983, 1983.

[74] C. Landwehr, C. Heitmeyer and J. McLean, "A Security Model for Military Message Systems: Retrospective," in *ACSAC*, 2001.

[75] E. Codd, "A Relational Model of Data for Large Shared Data Banks," *Information Retrieval*, vol. 13, no. 6, pp. 377-387, June 1970.

[76] M. R. Clarkson and F. B. Schneider, "Quantification of Integrity," *Proc. IEEE Computer Security Foundations Symposium*, pp. 28-43, 2010.

[77] B. Obama, "Executive Order 13526- Classified National Security Information," December 29, 2009.

[78] C. Shannon, Communication Theory of Secrecy Systems, 1946.

[79] C. Meadows, "Ordering from Satan's Menu," *Science of Computer Programming*, vol. 50, pp. 3-22, 2004.

[80] I. Cerevesato, "The Wolf Within," in *Second Workshop on Issues in the Theory of Security*, 2002.

[81] D. Weber and B. Lubarsky, "The SDOS project – Verifying Hook-up Security," in *Proceedings of the 1987 Aerospece Computer Security Conference*, Orlando, Florida, 1987.

[82] D. Weber, "Quantitative Hook-Up Security for Covert Channel Analysis," in *Computer Security FoundationsWorkshop*, 1988.

[83] D. McCullough, "A Hookup Theorem for Multilevel Security," in *IEEE Transactions on Software Engineering*, 1990.

[84] C. Shannon, "A Symbolic Analysis of Relay and Switching Circuits," 1937.

[85] Information Assurance Directorate, "U.S. Government Protection Profile for Separation Kernels in Environments Requiring High Robustness," 2007.

[86] S. Escobar, C. Meadows and J. Meseguer, "Maude-NPA, Version 1.0," 2007.

[87] NIST, "Cryptographic Hash Algorithm Competition," 2012.

[88] J. W. I. Gray and P. F. Syverson, "A Logical Approach to Multilevel Security of Probabilistic Systems," *Distributed Computing*, vol. 11, pp. 73-90, 1998.

[89] J. Gray, "Toward a Mathematical Foundation for Information Flow Security," *Journal of Computer Security*, vol. 1, no. 3-4, pp. 255-294, 1992.

[90] I. S. Moskowitz, R. E. Newman and P. S. Syverson, "Quasi-anonymous channels," *IASTED CNIS*, pp. 126-131, 2003.

[91] A. Machanavajjhala, D. Kifer, J. Gehrke and M. Venkitasubramaniam, "Privacy beyond k-anonymity," *ACM Trans. Knowl. Discov. Data*, 2007.

[92] K. Chatzikokolakis, C. Palamidessi and P. Panangaden, "Anonymity Protocols as Noisy Channels," *Information and Computation* , vol. 206, pp. 378-401, 2008.

[93] D. Chaum, "The Dining Cryptographers Problem: Unconditional Sender and Recipient Untraceability," *J. Cryptology*, pp. 65-75 , 1988.

[94] A. Odlyzko, "Comments on the Larry Roberts and Caspian Networks study of Internet traffic growth," in *The Cook Report on the Internet*, 2001, pp. 12-15.

[95] S. G. Stubblebine and P. F. Syverson, "Group Principals and the Formalization

of Anonymity," *World Congress on Formal Methods*, pp. 814-833, 1999.

[96] I. Moskowitz, R. E. Newman, D. P. Crepeau and A. R. Miller, "Covert Channels and Anonymizing Networks," *WPES*, pp. 79-88, 2003.

Appendix A: Analysis at Lower Epochs

At Epoch 2, one might specify an algorithm and prove that it enforces specified safety properties on outputs, granted assumptions concerning Epoch 3 separation and Epoch 1 realization. The algorithm might be imbued with knowledge of some axioms, definitions, constants and theorems pertaining to the subject matter of the specific algorithm as described in our Introduction. [19] While introductory formal methods are typically taught at Epoch 2, moving from a correct algorithm model to a verified refinement is a challenging and increasing problem. For example, present-day software compilers and hardware synthesis tools regularly perform not only language translation but also significant analysis and optimization including inferring memories and stacks for function parameters; inferring registers, caches and delays for variables; eliminating and reordering instructions; inferring ordinal numberings; selecting primitive operators; flattening recursions; translating, resizing and encoding numbers; etc. In these situations, one must develop extensive arguments for what is taken as axiom in math: function notation, integers, operators, variables, (prenex) normal form, ordinal numbers, and computation time. Add to this the complex and proprietary hypervisor and operating system memory management schemes, the complexity of I/O device interrupts and direct memory access (DMA), and heterogeneous hardware memory management and debug features that affect memory validation techniques.

At Epoch 1, a semiconductor manufacturer converts a notation such as a Boolean algebra or hardware description language to circuits [84], a task of approximately class TC computational complexity. An attacker at Epoch 1 may modify the Epoch's notation at design time, abuse the mapping from notation to Epoch 1 (semiconductor) realization, or alter the circuit after it has been manufactured. Regarding inputs and outputs, Shannon's model of the communication of bits assumes a *noisy medium* attacking at Epoch 0, mathematizes at Epoch 1, and proves the existence of a set of perfect codings at Epoch 2. At Epoch 1 the receiver is imprisoned in a state of uncertainty (entropy) concerning the sender's input message, and the same is true for the inputs and outputs of Epoch 1 circuit-based operators, especially when experiencing fault injection attacks. Similarly, some circuit designs exhibit power draw, electromagnetic emanations, etc., which may be considered extra outputs and may be analyzable by an attacker to reveal system secret data values.

At Epoch 0, the Standard Model of Physics expresses particles and interactions, any of which can be used to communicate. Security requirements at this Epoch may be expressed as assumptions on the environment. Assertions of system input or output devices, for example a radio frequency antenna or analog sensor, may also be appropriate. Threats against inputs and outputs are broad-ranging, but are generally more

effective when the attack is launched from the physical perimeter of the system, or if the system perimeter is penetrated or disassembled. An Epoch 0 sensor input device such as a Global Positioning System (GPS) antenna might provide inputs useful to the system until it is attacked, at which time it may provide inputs useful to the attacker. An output device such as an LED located near a user-facing label of "alarm" might provide security information a system user until it is attacked, at which time it may be disabled.

Understanding the Reasons for the Side-Channel Leakage is Indispensable for Secure Design

Extended Abstract

Werner Schindler

Bundesamt für Sicherheit in der Informationstechnik (BSI)
Godesberger Allee 185–189
53175 Bonn, Germany
Werner.Schindler@bsi.bund.de

Abstract. In the last one and a half decades side-channel analysis, and in particular power analysis, has become a very important topic in both academia and industry.
Hardening security implementations against power attacks has become a matter of course. The strength of an implementation is usually rated on basis of its resistance against particular types of power attacks (e.g., DPA, CPA, template attacks).
The stochastic approach combines engineering's expertise with advanced stochastic methods. Compared to template attacks this reduces the profiling workload drastically. Apart from (possibly) providing a successful attack the stochastic approach quantifies the side-channel leakage with regard to a vector space basis, and it allows to verify leakage model assumptions. This provides useful information, which supports target-oriented (re-)design.

Keywords: Power analysis, stochastic approach, secure design, constructive side-channel analysis, multivariate statistics.

1 Introduction

In 1996 Paul Kocher introduced timing attacks [16] and in 1999 power analysis [17]. Both publications have had enourmous impact on the crypto community. In particular, one can hardly find hardware-related conferences without a session on power analysis. Unlike many other scientific areas power analysis is important for academia and industry as well. Smart cards and many other security implementations have to be made secure against power attacks.

In the pioneer paper [17] SPA and DPA were introduced. In the first years power analysis was a domain of electrical engineers. As a natural consequence, the applied mathematical methods usually were elementary. Later, cooperations between engineers and mathematicians led to more sophisticated mathematical

methods [25] etc. In particular, CPA, template attacks and MIA were discovered. In 2005 the stochastic approach was introduced [23].

A common approach to rate the strength of a security implementation is to apply several power attacks (e.g., different DPA or CPA attacks), which often gives a 'one bit decision' (successful attack / no successful attack).

The stochastic approach combines engineering's expertise with advanced stochastic methods. Compared to template attacks this reduces the profiling workload drastically. Even more interesting, the profiling phase of the stochastic approach quantifies the side-channel leakage with regard to a vector space basis. Moreover, it allows to verify leakage model assumptions. Altogether, this yields information on the source of side-channel leakage, which in turn can be used for target-oriented design decisions.

Section 2 and Section 3 give a brief survey on different classes of power attacks and of the stochastic approach. Section 4 considers the design information, which can be gained from the stochastic approach. Final remarks conclude this paper.

2 Different Types of Power Attacks: A Brief Survey

Since [17] lots of papers on power analysis have been published. Most of them consider DPA (Differential Power Analysis) or CPA (Correlation Power Analysis) techniques [2, 13, 20, 22], and a large number of countermeasures have been proposed ([3, 5, 26] etc.). Both DPA and CPA exploit the correlation between a selection function and the electrical current. They are easy to apply but exploit only a fraction of the available information. In particular, it is not clear how to combine power measurements from different time instants effectively.

Template attacks ([4, 1, 21] etc.) overcome this problem as they combine the power information from several time instants $t_1 < \cdots < t_m$. The measured power consumption at $\boldsymbol{t} := (t_1, \ldots, t_m)$ is interpreted as a realization of an m-dimensional random vectors $\boldsymbol{I}_{\boldsymbol{t}}(x, k)$ whose unknown distribution depends on the targeted subkey k, and on some part of the plaintext, ciphertext or a function thereof, denoted by x. If the attacked device applies masking techniques, the random vector $\boldsymbol{I}_{\boldsymbol{t}}(x, z, k)$ also depends on a masking value z (random number).

In the profiling phase the adversary uses an identical training device to estimate the unknown densities (obtaining the so-called *templates*) $f_{x,k}(\cdot)$, resp. $f_{x,z,k}(\cdot)$ if masking techniques are applied. In the attack phase the adversary (attacker, designer or evaluator) performs measurements at the target device. With the usual assumption that the unknown densities are (at least approximately) multidimensional normally distributed profiling simplifies to the estimation of mean values and covariance matrices. In the attack phase the measurement values are substituted into the estimated densities (maximum likelihood principle).

In a 'full' template attack on a block cipher the adversary estimates the densities for all pairs (x, k) or for all triplets (x, z, k) (masking case) where x represents a part of the plaintext or ciphertext. For given time instants $t_1, \ldots, t_m$ a full template attack should have maximum attack efficiency among all attacks

that focus on the power consumption at $\boldsymbol{t} = (t_1, \ldots, t_m)$, at least if the sample size for the profiling series is sufficiently large. A clear disadvantage of this approach is the gigantic workload in the profiling phase, especially if masking techniques are applied. In particular for strong implementations (with similar densities) this should be infeasible.

A feasible alternative to full template attacks is to estimate the densities only for selected pairs (x, k) or triplets (x, z, k), respectively. For a security evaluation the problem consists in selecting a representative subset. Alternatively, a leakage model assumption may be introduced, e.g. that the electrical current depends only on $\mathrm{ham}(x, z) := x \oplus k$, reducing the number of templates tremendously but usually losing the optimality property of full template attacks.

Other approaches apply formal methods or ideas from information theory [28, 9].

3 The Stochastic Approach: A Short Summary

In this section we briefly summarize the central steps of the stochastic approach where we distinguish between three variants. The target-algorithm of the stochastic approach is a block cipher. For details the interested reeader is referred to [23, 8, 24, 27, 12]. In Section 4 we show how the stochastic approach can be used to support (re-)design. We mention that [6] investigates a non-profiling-based variant of the stochastic approach.

Notation. We denote subkeys by $k \in \{0,1\}^s$ while $x \in \{0,1\}^p$ stands for (the relevant part of) the plaintext or ciphertext, respectively (typically, 8 or 16 bits). Random variables are denoted by capital letters, realizations thereof, i.e. values taken on by these random variables, by the corresponding small letters. Vectors are written in bold, e.g., $\boldsymbol{t}$ stands for $(t_1, \ldots, t_m)$, and $\boldsymbol{R_t}$ denotes the random vector $(R_{t_1}, \ldots, R_{t_m})$. Accordingly, $\boldsymbol{I_t}(x, k)$, $\boldsymbol{i_t}(x, k)$, $\boldsymbol{h}^*_{\boldsymbol{t};k}(x, k)$ etc. while $\sim$ indicates estimates. We write $\mathrm{diag}_n(d_1, \ldots, d_n)$ for a diagonal $n \times n$ square matrix with diagonal elements $d_1, \ldots, d_n$, and $\mathcal{N}_n(\mu, F)$ denotes an n-dimensional normal distribution with mean vector μ and covariance matrix F.

3.1 The 'Classical' Stochastic Approach (No Masking)

We follow the brief description from [12], Subsect. 4.1. The stochastic approach is based on the mathematical model

$$I_t(x, k) = h_t(x, k) + R_t \tag{1}$$

where t denotes a time instant. The power consumption $i_t(x, k)$ is interpreted as a realization of a random variable $I_t(x, k)$ whose (unknown) distribution depends on the pair (x, k). The leakage function $h_t(x, k)$ quantifies its deterministic part, which depends on x and k, while R_t denotes the noise. W.l.o.g. we may assume $E(R_t) = 0$. Both the leakage function $h_t(\cdot, \cdot)$ and the distribution of the noise are unknown and thus have to be estimated.

Profiling Let $t \in \{t_1, \ldots, t_m\}$ and $k \in \{0,1\}^s$ be fixed for the moment. We view the restricted function $h_{t;k}\colon \{0,1\}^p \times \{k\} \to \mathbb{R}$, $h_{t;k}(x,k) := h_t(x,k)$ as an element of the 2^p-dimensional real vector space $\mathcal{F}_k := \{h'\colon \{0,1\}^p \times \{k\} \to \mathbb{R}\}$. Basis functions $g_{0,j;k}(\cdot,k) = 1$ (constant function), $\ldots, g_{u-1,t;k}(\cdot,k)$ shall be selected under consideration of the concrete implementation, since they shall capture the relevant source of side-channel leakage (cf. e.g. [15, 10] and Sect. 4). Note that we do not aim at the exact function $h_{t;k}(\cdot,k)$ itself but at its best approximator $h^*_{t;k}(\cdot,k)$ in $\mathcal{F}_{u,t;k}$, the subspace which is spanned by $g_{0,j;k}(\cdot,k), \ldots, g_{u-1,t;k}(\cdot,k)$. From the power measurements $i_t(x_1,k), \ldots, i_t(x_{N_1},k) \in \mathbb{R}$ the least square estimate $\widetilde{h}^*_{t;k}(\cdot,k)$ of $h_{t;k}(\cdot,k)$ is determined. Let

$$A := \begin{pmatrix} g_{0,t;k}(x_1,k) & \ldots & g_{u-1,t;k}(x_1,k) \\ \vdots & \ddots & \vdots \\ g_{0,t;k}(x_{N_1},k) & \ldots & g_{u-1,t;k}(x_{N_1},k) \end{pmatrix}. \tag{2}$$

If A^TA is regular (usual case) the normal equation $A^TA\boldsymbol{b} = A^T\boldsymbol{i}_t$ has the unique solution

$$\widetilde{\boldsymbol{b}}^{\,*} = (A^TA)^{-1}A^T\boldsymbol{i}_t, \qquad \text{with} \quad \widetilde{\boldsymbol{b}}^{\,*} := (\widetilde{\beta}^*_0, ..., \widetilde{\beta}^*_{u-1}), \quad \text{and} \tag{3}$$

$$\widetilde{h}^*_{t;k}(\cdot,k) = \sum_{j=0}^{u-1} \widetilde{\beta}^*_{j,t;k} g_{j,t;k}(\cdot,k) \quad \text{(least square estimate of } h^*_{t;k}(\cdot,k)). \tag{4}$$

The coefficients $\beta^*_{0,t;k}, \ldots, \beta^*_{u-1,t;k}$ are called β-characteristic, and the values $\widetilde{\beta}^*_{0,t;k}, \ldots, \widetilde{\beta}^*_{u-1,t;k}$ are their estimates. Note that the leakage functions $h_{t_1,k}, \ldots, h_{t_m,k}$ are estimated separately.

In the second profiling step the covariance matrix C of the noise vector $\boldsymbol{R_t}$ has to estimated, finally yielding a density for the random vector $\boldsymbol{I_t}(x,k)$. From an information theoretical point of view it seems to be advisable to consider as many time instants $t_1 < \cdots < t_m$ as possible. Unfortunately, then the covariance matrix C is often 'almost' singular so that matrix inversion becomes an ill-posed numerical problem. Consequently, moderate estimation errors in $\widetilde{C}$ might amplify to large estimation errors of $\widetilde{C}^{-1}$, which is needed to calculate the density of $\boldsymbol{I_t}(x,t)$. Let $\widetilde{\lambda}_1 \geq \cdots \geq \widetilde{\lambda}_m \geq 0$ denote the eigenvalues of the positive semidefinite matrix $\widetilde{C}$, and let v_j denote the normalized eigenvector to $\widetilde{\lambda}_j$. Assume further that the first s eigenvalues are considerably larger than the others, i.e. $\widetilde{\lambda}_{s+1} \ll \widetilde{\lambda}_s$. Then we concentrate on the subspace of $\mathbb{R}^m$, which is spanned by the eigenvectors $v_1, \ldots, v_s$. More precisely, if P_s denotes the $(m \times s)$-matrix with columns $v_1, \ldots, v_s$ then

$$P_s^T\widetilde{C}P_s = \widetilde{D}_s \quad \text{ with } \widetilde{D}_s = \mathrm{diag}_s(\widetilde{\lambda}_1, \ldots, \widetilde{\lambda}_s) \quad \text{(PCA)}. \tag{5}$$

where D_s is diagonal with diagonal entries $\widetilde{\lambda}_1, \ldots, \widetilde{\lambda}_s$. Typically, $s \leq 3$, and often $s = 1$. Note that if the random vector Y is $\mathcal{N}_m(0,C)$-distributed then P_s^TY is $\mathcal{N}_s(0, P_s^TCP_s)$-distributed [14]. Formula (5) adjusts principle component analysis (PCA) to the stochastic approach; cf. [1] (template attacks). We point out that it is numerically more convenient to calculate the transformation matrix P_s by means of a singular value decomposition than 'directly' from (5).

Attack Phase In the attack phase the adversary performs N_3 measurements at the target device and obtains power vectors $\boldsymbol{i_t}(x_1, k^\dagger), \ldots, \boldsymbol{i_t}(x_{N_3}, k^\dagger)$ with the unknown subkey $k^\dagger$ while $x_1, \ldots, x_{N_3}$ are known. The adversary decides for that subkey candidate $k^* \in \{0,1\}^s$ that

$$\text{maximizes } \prod_{l=1}^{N_3} f_{\widetilde{D}_s}\left(P_s^T\left(\boldsymbol{i_t}(x_l, k^\dagger) - \widetilde{\boldsymbol{h}}^*_{\boldsymbol{t};k}(x_l, k^*)\right)\right) \text{ resp., minimizes}$$

$$\sum_{j=1}^{N_3}\left(P_s^T\left(\boldsymbol{i_t}(x_j, k^\dagger) - \boldsymbol{h_t}(x_j, k^*)\right)\right)^t \widetilde{D}_s^{-1}\left(P_s^t\left(\boldsymbol{i_t}(x_j, k^\dagger) - \boldsymbol{h_t}(x_j, k^*)\right)\right) . \quad (6)$$

Here $f_{\widetilde{D}_s}$ denotes the density of the centered s-dimensional normal distribution with covariance matrix $\widetilde{D}_s$

3.2 The 'Classical' Stochastic Approach (With Masking)

If the implementation is protected by masking techniques the mathematical model (1) generalizes to

$$I_t(x,k) = h_t(x,z,k) + R_t \quad (7)$$

where z denotes a masking value. Analogously to the non-masking case one considers the restricted functions $h_{t;k}\colon \{0,1\}^p \times \{0,1\}^s \times \{k\} \to \mathbb{R}$, $h_{t;k}(x,z,k) := h_t(x,z,k)$ and basis functions $g_{j,t;k}(x,z,k)$. It is one of the pleasant properties of the stochastic approach that masking neither makes profiling more difficult nor requires larger profiling workload, which is very different to (at least) full template attacks. As for template attacks in the attack phase the maximum likelihood principle is applied to a convex combination of normal distributions [24]. This convex combination of densities expresses the effect of the applied masking techniques. Since we only consider non-masked implementations in the remainder we do not deepen these aspects here.

3.3 The Stochastic Approach: The OTM-Variant

In Sect. 3.1 we have tacitly assumed (as usual in literature) that the environmental conditions remain unchanged during the experiments, implying identical leakage functions and identical distribution of the noise during the whole profiling phase and during the attack phase. However, this may not always be the case. A prominent example are the power traces of the DPA contest v2 [7]. The organizers of the contest needed almost 4 days to record the so-called template base, which we used to estimate the leakage function and the distribution of the noise. Figure 1 shows that the average power consumption is diurnally periodic, presumably due to the variation of the temperature in the lab during these days (cf. [12], Sect. 3 for details).

This suggests the following extension of the mathematical model (1)

$$I_t(x_\ell, k) = h_t(x_\ell, k) + \tau_{t;\ell} + R_t\,. \quad (8)$$

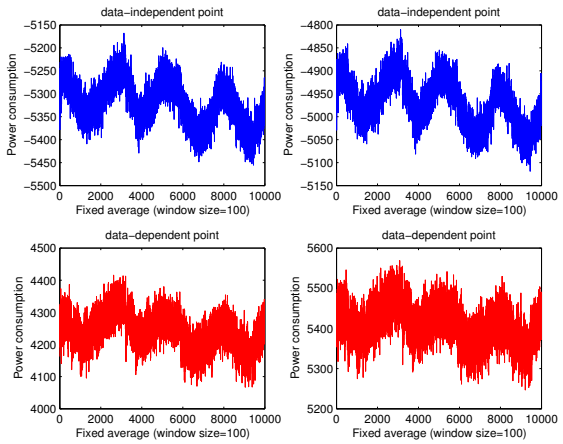


Fig. 1. Electrical current: Average value of 100 subsequent non-overlapping power traces [12]

The letter 'ℓ' denotes the label of the power trace, pointing to the time when the trace was recorded, while 't' (as before) stands for the time instant within the particular power trace. We call $\tau_{t;\ell}$ the 'drifting offset' fotr power trace ℓ. In particular, $I_t(x_\ell, k) \sim N(h_{t,k}(x_\ell, k) + \tau_{t;\ell}, \sigma^2)$.

By subtracting a suitable multiple of $g_{0,t;k}$ one can enforce that all basis vectors $g_{1,t;k}, \ldots, g_{u-1,t;k}$ have expectation 0. Then $\beta_{0,t;k} = E(I_t(X, k))$, and variations of the average current consumption influence the coefficient $\beta_{0,t;k}$. The coefficient $\beta_{0,t;k}$ is data-independent and its absolute value is usually significantly larger than the others (about factor ≈ 70 for the DPA contest v2 power traces, cf. [12]).

The drifting offsets $\tau_{t;l}$ are unknown but since the environmental conditions vary slowly $\tau_{t;\ell+1} - \tau_{t;\ell} \approx 0$, and thus

$$\begin{aligned} I_t(x_{\ell+1}, k) - I_t(x_\ell, k) &\approx h_t(x_{\ell+1}, k) - h_t(x_\ell, k) + R_t - R'_t \\ &\sim N(h_{t,k}(x_{\ell+1}, k) - h_{t,k}(x_\ell, k), 2\sigma^2)\,. \end{aligned} \tag{9}$$

In (9) the drifting offset does not appear. Based on this observation in [12] a new variant of the stochastic approach was developed, omitting the constant basis vector $g_{0,t;k}$, and the basis vectors $g_{1,t;k}, \ldots, g_{u-1,t;k}$ span the subspace $\mathcal{F}^\circ_{u,t;k}$. This variant is suitable for scenarios where a drifting offset occurs. The general concept is analogous to Subsect. 3.1.

- Profiling phase 1: Estimation of the reduced leakage function $h^{*\circ}_{t,k}(x, k) := \sum_{j=1}^{u-1} g_{j,t;k}(x, k) = h^*_{t,k}(x, k) - \beta_{0,t;k}$
- Profiling phase 2: Estimation of the covariance matrix, applying PCA.
- Attack phase: Maximum-Likelihood principle

Since the drifting offsets are unknown one exploits (9). Consequently, the random variables $\boldsymbol{I_t}(x_1, k), \ldots, \boldsymbol{I_t}(x_{N_1}, k)$ and the measurement vectors $\boldsymbol{i_t}(x_1, k), \ldots, \boldsymbol{i_t}(x_{N_1}, k)$ are not treated independently but overlapping pairs $\boldsymbol{I_t}(x_2, k) - \boldsymbol{I_t}(x_1, k)$, $\boldsymbol{I_t}(x_3, k) - \boldsymbol{I_t}(x_2, k), \ldots, \boldsymbol{I_t}(x_{N_1}, k) - \boldsymbol{I_t}(x_{N_1-1}, k)$ and $\boldsymbol{i_t}(x_2, k) - \boldsymbol{i_t}(x_1, k), \boldsymbol{i_t}(x_3, k) - \boldsymbol{i_t}(x_2, k), \ldots, \boldsymbol{i_t}(x_{N_1}, k) - \boldsymbol{i_t}(x_{N_1-1}, k)$ have to be considered. This implies additional mathematical difficulties, in particular in the attack phase, see [12], Subsect. 4.2, for details.

4 The Stochastic Approach Supports (Re-)Design

An outstanding feature of the stochastic approach is that it quantifies the leakage with regard to some vector space basis $g_{0,t;k}, \ldots, g_{u-1,t;k}$. In this section we explain how this property can be used to support target-oriented (re-)design. For design purposes only the first part of the profiling phase (estimation of the leakage function) is relevant.

4.1 How to Select Appropriate Basis Vectors (Example)

Figure 2 shows the final round of an AES implementation on an FPGA where all bytes are processed in parallel. From a logical point of view the registers $\mathrm{R1}, \ldots, \mathrm{R16}$ represent the intermediate results after Round 9 (upper row; denoted by $\widehat{\mathrm{Rj}}$ in the following) and the results after Round 10 (cipher text bytes; lower row). Physically, $\widehat{\mathrm{Rj}}$ and Rj address the same byte register. To $\widehat{\mathrm{Rj}}$ a byte is XORed, which comes from register value $\widehat{\mathrm{Ri}}$ (S-box, AddRoundKey) for a suitable index i. The CMOS technology suggests a distance model, which means that

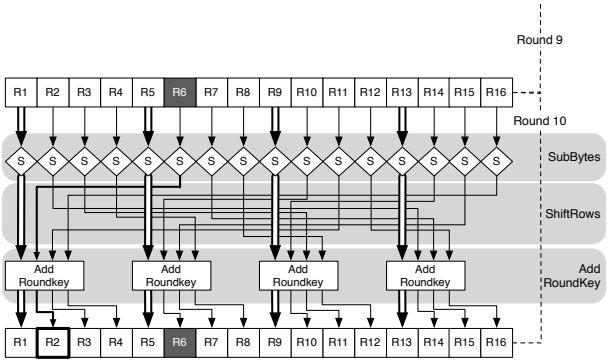


Fig. 2. FPGA implementation of AES: last round [15]

the power consumption to update the register value Rj depends on $k_{(j)} \oplus S(\widehat{\mathrm{Ri}})$,

resp. on $\mathrm{Rj} \oplus \widehat{\mathrm{Rj}}$. This motivates the choice of the following 9-dimensional basis:

$$g_{0,t;k_{(y)}}((x_{(z)}, x_{(y)}), k_{(y)}) = 1 \tag{10}$$
$$g_{j,t;k_{(y)}}((x_{(z)}, x_{(y)}), k_{(y)}) = (x_{(z)} \oplus S^{-1}(x_{(y)} \oplus k_{(y)}))_j - 0.5 \quad \text{for } j = 1, \ldots, 8$$

for suitable pairs (y, z) (cf.[15], Appendix, for instance). The index j denotes the j^{th} bit of the byte. This subspace aims at the subkey $k_{(y)}$. Note that the term $-0.5 = -0.5 g_{0,t;k}((x_{(z)}, x_{(y)}), k_{(y)})$ is not mandatory as these basis vectors span the same vector subspace as without this term, and even the coefficients $\beta_{j,t;k}$ are identical for $j > 0$. The reason to introduce the term -0.5 is that for $y \notin \{1, 5, 9, 13\}$ it is $E(g_{j,t;k_{(y)}}((x_{(z)}, x_{(y)}), k_{(y)})) = 0$ for $j = 1, \ldots, 8$ and uniformly distributed cipher text bytes $(x_{(y)}, x_{(z)})$; Example: $(y, z) = (2, 6)$, see [15]. In particular, $\beta_{0,t;k} = E(I_t(x, k))$. Note that for $y \in \{1, 5, 9, 13\}$ we have $y = z$, and $E(g_{j,t;k_{(y)}}((x_{(z)}, x_{(y)}), k_{(y)})) \approx 0$. (To obtain '$= 0$' for these y's one simply has to adjust the term -0.5.) Subsection 4.3 treats high-dimensional subspaces.

4.2 Information Gained from the β-Coefficients

Subsection 4.2 continues Subsection 4.1. Since the S-box defines a bijection it may be natural to expect $|\beta_{1,t;k(y)}| \approx \cdots \approx |\beta_{8,t;k(y)}|$. In fact, why should some bit lines behave significantly different than others? Figure 3 shows the β-coefficients $|\beta_{1,t_i,k_{(1)}}|, \ldots, |\beta_{8,t_i,k_{(1)}}|$ for 20 time instants $t_1 < \cdots < t_{20}$ for the key byte value $209 = (11010001)_2$ (other key values behave similarly).

These β-coefficients stem from an AES implementation on a XILINX FPGA. The S-box permutation was realised by a lookup-table, and the design was finally synthesised for the Virtex-II pro family using the automatic place & route algorithm. The β-coefficients indicate that the automatic place & route process treats the particular bits very differently. In fact, a closer look at the design (Figure 4) shows that bit 5 switches several first stage multiplexers etc. Of course, this is an important design information, which allows to identify the origin of this imbalancedness. We refer the interested reader to [15], Sect. 4, for details.

4.3 High-Dimensional Subspaces

In Subsection 4.1 and Subsection 4.2 we focused on the last round of an AES implementation on an FPGA. The subspace $\mathcal{F}_{9,t;k}$ revealed a significant weakness of this particular implementation. A pure Hamming distance model can not provide this information, and its attack efficiency is definitely lower. A natural question is whether larger subspaces than $\mathcal{F}_{9,t;k}$ further increase the attack efficiency and provide more information on the target implementation. On the negative side larger subspaces require larger sample sizes in the profiling phase.

The answer is that larger subspaces may indeed be much more efficient than $\mathcal{F}_{u,t;k}$. To illustrate this fact we briefly sketch the results from [12], which treats the power traces from the DPA contest v2 [7]. The template base of the DPA

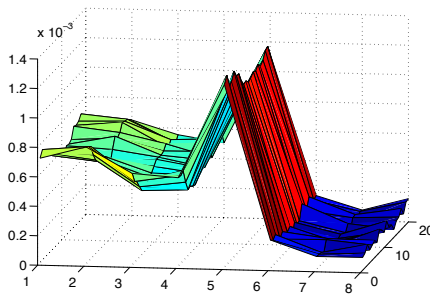


Fig. 3. β-characteristic for subkey $k_{(1)} = 209$ at time instants $t_1, \ldots, t_{21}$ [15]

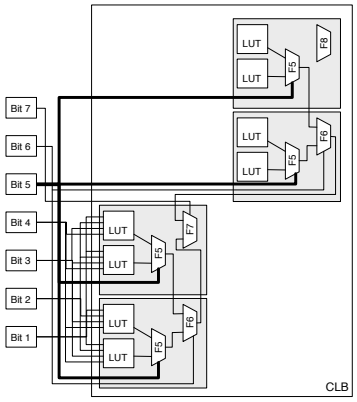


Fig. 4. Section of an TBL-AES design on a Virtex-II pro FPGA [15]

contest v2 consists of 1 million power traces (usable for profiling). Moreover, the organizers provided a public base consisting of 32 sets of 20000 power traces, each set belonging to one *key*. The public base allowed the contestants to check the strength of their attack, which they had developed on basis of the power traces in the template base. Finally, the organizers rated the submitted attacks by a secret private base. Since the DPA contest v2 provides power traces from an AES implementation on an FPGA it is reasonable to start with the basis (10) from Subsction 4.1

$$g_{j,t;k_{(y)}}((x_{(z)}, x_{(y)}), k_{(y)}) := \underbrace{(x_{(z)} \oplus S^{-1}(x_{(y)} \oplus k_{(y)}))_j}_{:=\hat{g}_{j,t;k}} - 2^{-1} \quad \text{for } j = 1, \ldots, 8. \tag{11}$$

We set $\mathcal{B}_0 := \{g_{0,t;k_{(y)}} = 1\}$ and $\mathcal{B}_1 := \{g_{1,t;k_{(y)}}, \ldots, g_{8,t;k_{(y)}}\}$, which captures the independent leakage contributions of the particular bit lines. Further, for $2 \leq i \leq 8$ we define the sets of basis vectors

$$\mathcal{B}_i := \{\hat{g}_{j_1,t;k_{(y)}} \cdots \hat{g}_{j_i,t;k_{(y)}} - 2^{-i} \mid 1 \leq j_1 < \ldots < j_i \leq 8\}. \tag{12}$$

(Example: $\hat{g}_{4,t;k_{(y)}} \cdot \hat{g}_{7,t;k_{(y)}} - 2^{-2} \in \mathcal{B}_2$.) The term -2^{-i} ensures that all basis vectors in $\mathcal{B}_i$ have expectation 0 (for $y \notin \{1, 5, 9, 13\}$). Since the contest traces showed a drifting offset (cf. Subsect. 3.3) we applied the OTM variant. More precisely, we considered the subspaces $\mathcal{F}^{\circ}_{u,t;k}$ for $u \in \{9, 37, 93, 163, 219, 247, 255, 256\}$, which are spanned by the vector space bases $\mathcal{B}_1, \mathcal{B}_1 \cup \mathcal{B}_2, \ldots, \mathcal{B}_1 \cup \cdots \cup \mathcal{B}_8$, respectively.

The attack efficiency increased significantly until $\mathcal{F}^{\circ}_{218,t;k}$ whose basis contains all 5-fold products. For $\mathcal{F}^{\circ}_{9,t;k}$ we needed 13020 power traces (public base) that the attack program ranked the correct key (all 16 key bytes) first with probability $> 80\%$. For $u = 37, 93, 163, 219$ the number of traces decreased to

7533, 6734, 6144, 4564. In combination with vertical trace alignment for $\mathcal{F}^{\circ}_{256,t;k}$ the number of traces dropped down to 3836 power traces (public basse) and to 3589 for the private base. For details we refer the interested reader to [12], Sects. 6-7.

In each of the three main evaluation categories (PSR stable, GSR stable, max PGE stable) the stochastic approach (OTM method) needed only half as many power traces as the contest winners in the respective category. (We submitted after the end of the contest.) This shows that the stochastic approach exploits the information that is contained in the power traces significantly more efficient than the other attacks. It should be noted that even the β-coefficients for some 5-fold products were non-negligible. The corresponding basis vectors capture the leakage, which stems from the interaction of 5 bit lines. This can't be the effect of crosstalk phenomena but might be the effect of propagation glitches [12].

In the meanwhile a Japanese research group has submitted a new attack, which is in the categories PSR stable and GSR stable (but not in max PGE stable) even better than the stochastic approach. This attack combines information from Round 10 (gained by a CPA) with information from Round 9 (clockwise collision analysis [19]), which is new. Combining the stochastic approach in Round 10 with clockwise collision analysis in Round 9 should further improve their result.

4.4 Symmetries

In Subsection 4.1 we assumed that the leakage function $h_t(\cdot,\cdot)$ (AES implementation, last round) depends on its argument $((x_{(z)}, x_{(y)}), k_{(y)})$ only through the term $\phi((x_{(z)}, x_{(y)}), k_{(y)}) := x_{(z)} \oplus S^{-1}(x_{(y)} \oplus k_{(y)})$. If this symmetry assumption is indeed valid there exists a function $\overline{h}_t\colon \{0,1\}^8 \to \mathbb{R}$ such that $h_t = \overline{h}_t \circ \phi$.

In the general case we analogously search for functions $\phi\colon \{0,1\}^p \times \{0,1\}^s \to \Omega$ such that $h_t(x,k) = \overline{h}_t \circ \phi(x,k)$ for suitable $\overline{h}_t\colon \Omega \to \mathbb{R}$. The mapping ϕ clearly depends on the concrete implementation, and not for each implementation such a function ϕ need to exist.

Clearly $h_{t;k} = \overline{h}_t \circ \phi(\cdot,k) \in \mathcal{F}_{k,\phi} := \{\overline{h}' \circ \phi(\cdot,k) \mid \overline{h}'\colon \Omega \to \mathbb{R}\}$, which is a vector subspace of $\mathcal{F}_k$. Consequently, for $k \in \{0,1\}^s$ it is reasonable to select a basis $g_{0,t;k}, \ldots, g_{u-1,t;k}$ of the form $g_{j,t;k} = \overline{g}_{j;t} \circ \phi(\cdot,k)$ with $\overline{g}_{j;t}\colon \Omega \to \mathbb{R}$, i.e. $g_{j,t;k} \in \mathcal{F}_{k,\phi}$. This clearly ensures $\mathcal{F}_{u,t;k} \subseteq \mathcal{F}_{k,\phi} \subseteq \mathcal{F}_k$, and replacing $\phi(\cdot,k)$ by $\phi(\cdot,k')$ provides a basis for $k' \in \{0,1\}^s$.

If $\phi\colon \{0,1\}^p \times \{0,1\}^s \to \Omega$ fulfils the following conditions

$$\text{(i) } |\phi^{-1}(\omega) \cap (\{0,1\}^p \times \{k\})| \text{ is identical for all } (\omega,k) \in \Omega \times \{0,1\}^s \quad (13)$$

$$\text{(ii) The random variable } X \text{ is uniformly distributed.} \quad (14)$$

then for each subkey $k \in \{0,1\}^s$

$$\mathrm{Prob}(\phi(X,k) = \omega) = \frac{1}{|\phi(\{0,1\}^p \times \{0,1\}^s)|} \text{ for all } \omega \in \phi(\{0,1\}^p \times \{0,1\}^s), \quad (15)$$

i.e. the random variable $\phi(X,k)$ is uniformly distributed on $\phi(\{0,1\}^p \times \{0,1\}^s) = \phi(\{0,1\}^p \times \{k\})$. In particular, $\phi(X,k')$ is identically distributed for all subkeys $k' \in \{0,1\}^s$. If the symmetry assumption, expressed by ϕ, is indeed valid (15) implies that the coefficients $\beta_{j,t;k'}$ in $h^*_{t;k'} = \sum_{j=0}^{u-1} \beta_{j,t;k'} \overline{g}_{j,t} \circ \phi(\cdot, k')$ do not depend on k'. This means

$$\beta_{j,t;k'} \equiv \beta_{j,t} \quad \text{for all } k' \in \{0,1\}^s\,, \tag{16}$$

i.e. the coefficients $\beta_{j,t,\cdot}$ are identical for all subkeys $k' \in \{0,1\}^s$. (If condition (15) is only 'nearly' fulfilled one may expect that (16) is at least approximately valid.)

Hence it suffices to estimate the leakage function $h_t(\cdot, k)$ for a single subkey k since $h_t(x,k') = h_t(x^*,k)$ whenever $\phi(x,k') = \phi(x^*,k)$. Moreover, applying ϕ one can 'convert' a power trace for subkey k' into a power trace for any given subkey k^*. We made use of this property in the DPA contest v2 (cf. Subsect. 4.3), which allowed to use all power traces from the template base for a single estimation process. We point out that the concept of symmetry can be generalized to masking techniques in a straight-forward way, e.g. $\phi\colon \{0,1\}^p \times M \times \{0,1\}^s \to \Omega$ is the pendant to $\phi\colon \{0,1\}^p \times \{0,1\}^s \to \Omega$ etc.

Remark 1. (i) The mapping ϕ describes a leakage model, namely that the leakage $h_t(x,k)$ only depends on the term $\phi(x,k)$.
(ii) For specific mappings ϕ the pre-images $\phi^{-1}(\omega)$ can be represented as orbits of an action of a group G on the set $\{0,1\}^p \times \{0,1\}^s$. Examples are $\phi\colon \{0,1\}^8 \times \{0,1\}^8 \to \{0,1\}^8, \phi(x,k) := x \oplus k$ (cf. [23], Subsect. 3.1), with group $G = \{0,1\}^8$ acting on $\{0,1\}^8 \times \{0,1\}^8$ via $(y,(x,k)) \mapsto (x \oplus y, k \oplus y)$, likewise the mapping ϕ_A in Subsection 4.5 below, and Example 2.10 (ii) in [24] (masking case). Although such a representation does not for each mapping ϕ we use the intuitive term 'symmetry', which has already been adopted in literature in this context.

4.5 How to Verify a Leakage Model Assumption

Symmetries have nice properties. It suffices to estimate the leakage function $h_{t;k}$ for a single subkey, which reduces the profiling workload in Phase 1 by factor 2^{-s}, and any power trace (regardless of the corresponding subkey) can be used for profiling. Vector subspaces $\mathcal{F}_{u,t;k}$ need only be searched within $\mathcal{F}_{k,\phi}$. This inclusion is automatically ensured for basis vectors of type $g_{j,t;k} := \overline{g}_{j,t} \circ \phi$. Usually $\dim(\mathcal{F}_{k,\phi})$ is much smaller than $\dim(\mathcal{F}_k)$. For the AES example treated in Subsections 4.1 and 4.3 it is $\dim(\mathcal{F}_{k,\phi}) = 2^8$ whereas $\dim(\mathcal{F}_{k,\phi}) = 2^{16}$. In Subsection 4.3 we used amongst others the whole vector space $\mathcal{F}_{k,\phi}$ (at cost of a large profiling sample size). This is definitely infeasible for $\mathcal{F}_k$.

Of course, it is very likely that a non-justified symmetry assumption leads to wrong conclusions. First of all, for given subkey k the vector space basis might be inappropriate (neglecting relevant sources of the power leakage), and certainly the 'transfer' of β-coefficients to other subkeys by means of (16) is presumably not justified. In particular, the power leakage itself and thus the

threat by (strong) power attacks might be underestimated, which may cause wrong design decisions.

Consequently, we need criteria which allow to verify or to falsify a given symmetry assumption. An important property is (16), which holds if the symmetry assumption and (15) are valid. A straight-forward approach is to profile for several subkeys and then to compare the estimated β-coefficients. Considerably different β-coefficients (for identical indices j) falsify the symmetry assumption while fairly identical β-coefficients supports the symmetry assumption.

This decision process may be quantified. We assume that $g_{0,t;k} = 1, g_{1,t;k} := \overline{g}_{1,t} \circ \phi(\cdot, k), \ldots, g_{u-1,t;k} := \overline{g}_{u-1,t} \circ \phi(\cdot, k) \in \mathcal{F}^{\circ}_{u,t;k}$ is an orthonormal basis of $\mathcal{F}_{u,t;k}$ with regard to the scalar product $(f_1, f_2) \mapsto 2^{-p} \sum_{x \in \{0,1\}^p} f_1(x, k) \cdot f_2(x, k)$, which corresponds to uniformly distributed X. Then $g_{0,t;k'} = 1, g_{1,t;k'} := \overline{g}_{1,t} \circ \phi(\cdot, k'), \ldots, g_{u-1,t;k'} := \overline{g}_{u-1,t} \circ \phi(\cdot, k')$ is an orthonormal basis of $\mathcal{F}_{u,t,k'}$ for each admissible subkey k'. We define

$$\mathrm{sym}_u(k', k'') := \frac{2 \sqrt{\sum_{j=1}^{u-1} (\beta_{j,t,k'} - \beta_{j,t,k''})^2}}{\sqrt{\sum_{j=1}^{u-1} \beta_{j,t,k'}^2} + \sqrt{\sum_{j=1}^{u-1} \beta_{j,t,k''}^2}} \tag{17}$$

which compares the β-coefficients for mutally different subkeys k' and k''. Obviously, $\mathrm{sym}_u(k', k'') = 0$ if the β-coefficients are identical for k' and k'', and if the symmetry assumption and (15) are valid then $\mathrm{sym}_u(k', k'') = 0$ for all admissible pairs of subkeys. The term (17) quantifies a 'symmetry distance' between different subkeys. The term $\mathrm{sym}_u(k', k'')$ remains invariant if all β'- and β''-coefficients are multiplied by a positive scalar or if the basis $g_{0,t;k} = 1, g_{1,t;k} := \overline{g}_{1,t} \circ \phi(\cdot, k), \ldots, g_{u-1,t;k} := \overline{g}_{u-1,t} \circ \phi(\cdot, k)$ is replaced by any other orthonormal basis $g'_{0,t;k} = 1, g'_{1,t;k} := \overline{g}'_{1,t} \circ \phi(\cdot, k), \ldots, g'_{u-1,t;k} := \overline{g}'_{u-1,t} \circ \phi(\cdot, k)$ (for all $k \in \{0,1\}^s$). Since the correct β-coefficients are unknown one substitutes their estimates $\widetilde{\beta}_{j,t,k'}$ and $\widetilde{\beta}_{j,t,k''}$ (gained from independent profiling processes for k' and k'', neglecting (16) for the moment) into (17). Of course, for increasing profiling sample size this term converges to (17).

Now we illustrate the preceding by an example. With regard to the AES implementation in Subsection 4.1 we assumed that the leakage function $h_t(\cdot, \cdot)$ depends on its argument $(x_{(z)}, x_{(y)}), k_{(y)})$ only through $\phi_B((x_{(z)}, x_{(y)}), k_{(y)}) := x_{(z)} \oplus S^{-1}(x_{(y)} \oplus k_{(y)})$. We considered the 9-dimensional vector subspace $\mathcal{F}_{9,t;k}$, which is spanned by the basis vectors

$$\begin{aligned} &g_{0,t;k_{(y)}}((x_{(z)}, x_{(y)}), k_{(y)}) = 1 \qquad (18)\\ &g_{j,t;k_{(y)}}((x_{(z)}, x_{(y)}), k_{(y)}) = 2\left(x_{(z)} \oplus S^{-1}(x_{(y)} \oplus k_{(y)})_j - 0.5\right) \quad \text{for } j = 1, \ldots, 8\,. \end{aligned}$$

Compared to (10) for each index $j > 0$ we multiplied the basis vector by '2'. Of course, this does not change the spanned vector subspace. At least for $y \notin \{1, 5, 9, 13\}$ the basis (18) is orthonormal, and ϕ_B fulfils (13).

Alternatively, we consider a second symmetry assumption $\phi_A(x_{(y)}, k_{(y)}) := S^{-1}(x_{(y)} \oplus k_{(y)})$, which yields the following 9-dimensional orthonormal basis

$$g_{0,t;k_{(y)}}(x_{(y)}, k_{(y)}) = 1 \tag{19}$$

$$g_{j,t;k_{(y)}}(x_{(y)}, k_{(y)}) = 2(S^{-1}(x_{(y)} \oplus k_{(y)})_j - 0.5) \quad \text{for } j = 1, \ldots, 8\,.$$

At first we compared the estimated β-coefficients for both symmetry assumptions with regard to (16). Indeed, for symmetry assumption ϕ_B only small differences occured while for ϕ_A the differences were significant ([10], Fig. 2 and Fig. 3; ϕ_A and ϕ_B refer to Model $\mathcal{A}$, resp. to Model $\mathcal{B}$ in [12]). This is a clear indicator, which speaks for symmetry assumption ϕ_B but against ϕ_A. (Of course, with regard to the implementation this is not really surprising.)

Moreover, we applied (17) to both the symmetry assumptions ϕ_A and ϕ_B, and in each case we compared the estimated β-coefficients for different subkeys. Figure 5 and Figure 6 illustrate the results for the last three AES rounds for key byte $k_{(2)}$. The middle part of Figure 6 shows that the symmetry assumption ϕ_B fits pretty well during the last round. Interestingly, Figure 6 shows that time instants with low symmetry values have (at least one) large β-coefficient, which is an indicator that this time instant provides 'much' leakage information. For details we refer the interested reader to [10], Sect. IV.

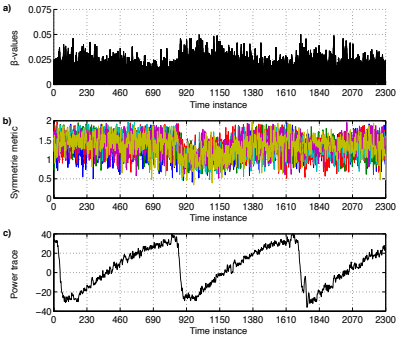


Fig. 5. [10] ϕ_A: a) $\max_{1\le j\le 8}\{|\beta_{j,t;k}|\}$, b) 'symmetry distance' (17), c) power consumption

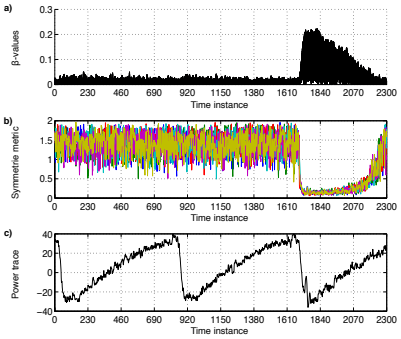


Fig. 6. [10] ϕ_B: a) $\max_{1\le j\le 8}\{|\beta_{j,t;k}|\}$, b) 'symmetry distance' (17), c) power consumption

4.6 Further Aspects

In Subsection 4.3 we rated the suitability of a subspace by its attack efficiency. In [11] we developed a method to estimate the L^2 distance of $\widetilde{h}^*_{t;k}$ to $h_{t;k}$. Let the subkey k be fixed, and further $x_1, \ldots, x_{2N}, x'_1, \ldots, x'_{2N} \in \{0,1\}^p$ with $x'_{2j-1} = x'_{2j}$ for $j = 1, \ldots, N$. As usual, $i_t(x_1, k), \ldots, i_t(x'_{2N}, k)$ denote the corresponding power traces. It follows from formulae (13) and (16) in [11] that

$$2^{-p} \sum_{x\in\{0,1\}^p} (h_{t;k} - \widetilde{h}^*_{t;k})^2 \approx \tag{20}$$

$$\frac{1}{2N}\sum_{j=1}^{2N}\left(i_t(x_j,k)-\widetilde{h}^*_{t;k}(x_j,k)\right)^2-\frac{1}{2N}\sum_{v=1}^{N}\left(i_t(x'_{2v-1},k)-i_t(x'_{2v},k)\right)^2$$

for sufficiently large N, providing an estimate for the searched L^2 distance (cf. [11] for deetails). Similarly, in [11] we also introduced an estimator for the signal-to-noise ratio.

5 Final Remarks

The stochastic approach is a very efficient attack variant. Moreover, it can also be used to gain valuable design information, possibly pointing to weaknesses of the implementation. The stochastic approach allows the verification or falsification of leakage models, and it is a useful tool that supports constructive side-channel analysis.

References

1. C. Archambeau, E. Peeters, F.-X. Standaert, J.-J. Quisquater: Template attacks in Principal Subspaces. In: L. Goubin, M. Matsui (eds.): Cryptographic Hardware and Embedded Systems — CHES 2006, Springer, Lecture Notes in Computer Science 4249, Berlin 2006, 1–14.
2. E. Brier, C. Clavier, F. Olivier: Correlation Power Analysis with a Leakage model. In: M. Joye, J.-J. Quisquater (eds.): Cryptographic Hardware and Embedded Systems — CHES 2004, Springer, Lecture Notes in Computer Science 3156, Berlin 2004, 16–29.
3. S. Chari, C.S. Jutla, J.R. Rao, P. Rohatgi: Towards Sound Approaches to Counteract Power-Analysis Attacks. In: M. Wiener (ed.): Advances in Cryptology — CRYPTO '99, Springer, Lecture Notes in Computer Science 1666, Berlin 1999, 398–412.
4. S. Chari, J.R. Rao, P. Rohatgi: Template Attacks. In: B.S. Kaliski Jr., Ç.K. Koç, C. Paar (eds.): Cryptographic Hardware and Embedded Systems — CHES 2002, Springer, Lecture Notes in Computer Science 2523, Berlin 2003, 13–28.
5. J.-S. Coron and L. Goubin: On Boolean and Arithmetic Masking against Differential Power Analysis. In: Ç.K. Koç, C. Paar (eds.): Cryptographic Hardware and Embedded Systems — CHES 2000, Springer, Lecture Notes in Computer Science 1965, Berlin 2000, 231-237.
6. J. Doget, E. Prouff, M. Rivain, F.-X. Standaert: Univariate Side Channel Attacks and Leakage Modeling. In: COSADE 2011, Darmstadt 2011, 116.
7. DPA contest v2, http://www.dpacontest.org/
8. B. Gierlichs, K. Lemke, C. Paar: Templates vs. Stochastic Methods. In: L. Goubin, M. Matsui (eds.): Cryptographic Hardware and Embedded Systems — CHES 2006, Springer, Lecture Notes in Computer Science 4249, Berlin 2006, 15–29.
9. B. Gierlichs, L. Batina, P. Tuyls, B. Preneel: Mutual Information Analysis - A Generic Side-Channel Distinguisher. In: E. Oswald, P. Rohatgi (eds.): Cryptographic Hardware and Embedded Systems - CHES 2008, Lecture Notes in Computer Science 5154, Springer, Berlin 2008, 426-442.

10. A. Heuser, M. Kasper, W. Schindler, M. Stöttinger: How a Symmetry Metric Assists Side-Channel Evaluation - A Novel Model Verification Method for Power Analysis. In: P. Kitsos (Hrsg.): 14^{th} EUROMICRO Conference on Digital System Design — DSD 2011, IEEE Press 2011, 674–681.
11. A. Heuser, W. Schindler, M. Stöttinger: Revealing Side-Channel Issues of Complex Circuits by High-Dimensional Leakage Models. In: Design, Automation & Test in Europe — DATE 2012, IEEE Press 2012, 1179–1184.
12. A. Heuser, M. Kasper, W. Schindler, M. Stöttinger: A Difference Method for Side-Channel Analysis Exploiting High-Dimensional Leakage Models. In: O. Dunkelman: (Hrsg.): Topics in Cryptology — CT-RSA 2012, Springer, Lecture Notes in Computer Science 7178, Berlin 2012, 365–382.
13. M. Joye, P. Paillier, B. Schoenmakers: On Second-Order Differential Power Analysis. In: J.R. Rao, B. Sunar (eds.): Cryptographic Hardware and Embedded Systems — CHES 2005, Springer, Lecture Notes in Computer Science 3659, Berlin 2005, 293–308.
14. O.J.W.F. Kardaun: Classical Methods of Statistics, Springer, Berlin 2005.
15. M. Kasper, W. Schindler, M. Stöttinger: A Stochastic Method for Security Evaluation of Cryptographic FPGA Implementations. In: 2010 International Conference on Field-Programmable Technology — FPT 2010, IEEE Press, CFP10528_CDR, 2010, 146–153.
16. P. Kocher: Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS and Other Systems. In: N. Koblitz (ed.): Crypto 1996, Springer, Lecture Notes in Computer Science 1109, Heidelberg 1996, 104–113.
17. P. Kocher, J. Jaffe, B. Jun: Differential Power Analysis. In: M. Wiener (ed.): Advances in Cryptology — CRYPTO '99, Springer, Lecture Notes in Computer Science 1666, Berlin 1999, 388–397.
18. K. Lemke-Rust, C. Paar: Analyzing Side Channel Leakage of Masked Implementations with Stochastic Methods. In: J. Biskup, J. Lopez (eds.): Computer Security — ESORICS 2007, Springer, Lecture Notes in Computer Science 4734, Berlin 2007, 454–468.
19. Y. Li, D. Nakatsu, Q. Li, K. Ohta, K. Sakiyama: Clockwise Collision Analysis Overlooked Side-Channel Leakage Inside Your Measurements. Cryptology ePrint Archive, Report 2011/579, http://eprint.iacr.org/2011/579
20. T.S. Messerges: Using Second-Order Power Analysis to Attack DPA Resistant Software. In: Ç.K. Koç, C. Paar (eds.): Cryptographic Hardware and Embedded Systems — CHES 2000, Springer, Lecture Notes in Computer Science 1965, Berlin 2000, 238–251.
21. E. Oswald, S. Mangard: Template Attacks on Masking — Resistance is Futile. In: M. Abe (ed.): Cryptographers' Track — CT-RSA 2007, Springer, Lecture Notes in Computer Science 4377, Berlin 2007, 243–256.
22. E. Peeters, F.-X. Standaert, N. Donckers, J.-J. Quisquater: Improved Higher-Order Side-Channel Attacks with FPGA Experiments. In: J.R. Rao, B. Sunar (eds.): Cryptographic Hardware and Embedded Systems — CHES 2005, Springer, Lecture Notes in Computer Science 3659, Berlin 2005, 309–323.
23. W. Schindler, K. Lemke, C. Paar: A Stochastic Model for Differential Side Channel Analysis. In: J.R. Rao, B. Sunar (eds.): Cryptographic Hardware and Embedded Systems — CHES 2005, Springer, Lecture Notes in Computer Science 3659, Berlin 2005, 30–46.
24. W. Schindler: Advanced Stochastic Methods in Side Channel Analysis on Block Ciphers in the Presence of Masking. Math. Crypt. 2 (2008), 291–310.

25. W. Schindler: Side-Channel Analysis – Mathematics has Met Engineering. In: A. Biedermann, H.G. Molter (Hrsg.): Design Methodologies for Secure Embedded Systems. Springer, Lecture Notes in Electrical Engineering 78, Berlin 2010, 43–62.
26. K. Schramm and C. Paar: Higher Order Masking of the AES. In: David Pointcheval (ed.): The Cryptographers' Track at the RSA Conference 2006, Springer, Lecture Notes in Computer Science 3860, Berlin 2006, 208–225.
27. F.-X. Standaert, F. Koeune, W. Schindler: How to Compare Profiled Side-Channel Attacks. In: M. Abdalla, D. Pointcheval, P.-A. Fouque, D. Vergnaud (eds.): Applied Cryptography and Network Security — ACNS 2009, Springer, Lecture Notes in Computer Science 5536, Berlin 2009, 485–498.
28. F.-X. Standaert, T.G. Malkin, M. Yung: A Unified Framework for the Analysis of Side-Channel Key Recovery Attacks. In: A. Joux (ed.): Eurocrypt 2009, Lecture Notes in Computer Science 5479, Berlin 2009, 443-461.

Toward Formal Design of Cryptographic Processors Based on Galois Field Arithmetic

Naofumi Homma

Graduate School of Information Sciences, Tohoku University
homma@aoki.ecei.tohoku.ac.jp

Abstract. The continuing research and development of arithmetic circuits have been conducted by the growth of LSI design technologies as well as the rapid expansion of the application area. In particular, the importance of GF (Galois-field) arithmetic circuits has been rapidly increasing due to the high demand of error correction codes and cryptographic systems in recent embedded devices. On the other hand, most of such GF arithmetic circuits are designed at the lowest level of abstraction by researchers who had trained in a particular way to understand the basic arithmetic since the conventional HDLs (Hardware Description Languages) do not have high-level arithmetic data structures, arithmetic operations or formulae over GFs. Even the state-of-the-art design environment (e.g. SystemC and System Verilog) can provide only limited capability to design GF arithmetic circuit structures. The lack of a high-level design methodology forces designers to verify structural details of GF arithmetic circuits with a huge amount of time. In fact, it is almost impossible to test all input patterns of GF circuits for practical word lengths. Nevertheless, at the same time, complete verification of such GF circuits is highly demanded in critical applications.
This talk presents a formal approach to describing and verifying GF arithmetic circuits that can be used in many security applications. The proposed method describes GF arithmetic circuits in the form of hierarchical graph structures, where nodes represent sub-circuits whose functions are defined by arithmetic formulae over GFs, and edges represent data dependency between nodes. The proposed description can be formally verified by symbolic computations based on Gröbner Bases and polynomial reduction. The verified description is then translated into the equivalent HDL codes, which are available for the conventional design flow. The proposed approach has a definite possibility of verifying practical GF arithmetic circuits where the conventional simulation techniques failed. The advantage of the proposed approach is demonstrated through experimental designs of parallel multipliers over Galois field $GF(2^m)$ for different word-lengths and irreducible polynomials. The proposed approach is also applied to the design of a 128-bit AES (Advanced Encryption Standard) datapath. The results show that the proposed method can describe the 128-bit datapath in a formal manner, as well as that complete verification of such a datapath can be carried out within a short period of time.

Analysing Cryptographic Hardware Interfaces with Tookan

Invited Talk

Graham Steel

INRIA Project Prosecco, Paris, France
graham.steel@inria.fr

Abstract. Cryptographic hardware offers access to its functionality via an application program interface (API). Designing such interfaces so that they offer flexible functionality but cannot be abused to reveal keys or secrets has proved to be extremely difficult, with a number of published vulnerabilities in widely-used APIs appearing over the last decade. This paper will discuss recent research on the use of formal methods to specify and verify cryptographic device interfaces in order to either detect flaws or prove security properties. We will focus on the example of RSA PKCS#11, the most widely used interface for cryptographic devices. We will describe a tool, Tookan, which can reverse engineer the particular configuration of PKCS#11 in use on some device under test, construct a model of the device's functionality, and call a model checker to search for attacks. If an attack is found, it can be executed automatically on the device. Tookan can also be used to prove security in its abstract model. We will also discuss open problems and future work.

1 Introduction

A security API is an *Application Program Interface* that allows untrusted code to access sensitive resources in a secure way. Examples of security APIs include the interface between the tamper-resistant chip on a smartcard (trusted) and the card reader (untrusted), the interface between a cryptographic *Hardware Security Module*, or HSM (trusted) and the client machine (untrusted). The crucial aspect of a security API is that it is designed to enforce a policy, i.e. no matter what sequence of commands in the interface are called, and no matter what the parameters are, certain security properties should continue to hold. This means that if the less trusted code turns out to be malicious (or just faulty), the carefully designed API should prevent compromise of critical data.

The first security vulnerability that may properly be called an 'API attack' on cryptographic hardware was discovered by Longley and Rigby in the early 1990s [27]. Their article showed how the logic programming language Prolog could be used to analyse a key management interface of a cryptographic device. Although the device was not identified at the time, it later became known that it was an HSM manufactured by Eracom and used in the cash machine network. In

2000, Anderson published an attack on key loading procedures on another similar module manufactured by Visa [1], and the term 'security API' was coined by Bond and Anderson [5, 6] in two subsequent papers giving more attacks. Clayton and Bond showed how one of their more computationally intensive attacks could be implemented against a real IBM device using programmable FPGA hardware [11]. Independently from the Cambridge group, an MSc thesis by Clulow gave more examples of attacks, mostly specific to the PIN translation and verification commands offered by the API of Prism HSMs [12]. Clulow also published attacks on the industry standard for cryptographic key management APIs, RSA PKCS#11 [13].

Up until this point all the attacks had been discovered by manual analysis or by ad-hoc semi-formal techniques specific to the particular API under consideration. A first effort to apply more general formal tools, specifically the automatic first-order theorem prover Otter, was not especially successful, and the results remain unpublished (though they are available in a technical report [34]). The researchers were unable to discover any new attacks, and because the modelling lacked formal groundwork, when no attacks were found they were unable to conclude anything about the security of the device. One member of the team later remarked that "It ended up being more about how to use the tools than about analysing the device" [22].

Meanwhile, the formal analysis of protocols for e.g. cryptographic key exchange and authentication had become a mature field. A particularly successful approach had centred around the so-called Dolev-Yao (DY) abstract model, where bitstrings are modelled as terms in an abstract algebra, and cryptographic functions are functions on these terms [17]. Together with suitable abstractions, lazy evaluation rules and other heuristics, this model has proved highly amenable to automated analysis, by model checking or theorem proving techniques [2, 4, 18, 28]. Modern automated tools can check secrecy and authentication properties of (abstract models of) widely-used protocols such as Kerberos and TLS in a few seconds.

The idea of applying protocol analysis techniques to the analysis of security APIs seemed very attractive. However, initial experiments showed that existing tools were not suitable for the problem [7, 25] for a number of reasons: In particular many of the attacks pertinent to security APIs are outside the scope of the normal DY model. For example, they might involve an attacker learning a secret value, such as a cash machine PIN, by observing error messages returned by the API (a so-called error oracle attack). Furthermore, the functionality of security APIs typically depends on global mutable state which may loop, a feature which invalidates many abstractions and optimisations made by protocol analysis tools, particularly when freshly generated nonces and keys are considered. There are also problems of scale - a protocol might describe an exchange of 5 messages between two participants, while an API will typically offer dozens of commands.

In this introductory paper, we will describe how protocol analysis techniques have been adapted to analyse device APIs, in particular focusing on the example of RSA PKCS#11.

2 Key Management APIs using RSA PKCS#11

Cryptographic key management, i.e. the secure creation, storage, backup, use and destruction of keys has long been identified as a major challenge in applied cryptography. Indeed, Schneier calls it "the hardest part of cryptography and often the Achilles' heel of an otherwise secure system." [32]. In real-world applications, key management often involves the use of HSMs or other cryptographic devices, since these are considered easier to secure than commodity hardware, and indeed are mandated by standards in certain sectors [24]. There is also a growing trend towards enterprise-wide schemes based around key management servers offering cryptographic services over open networks [9]. All these solutions aim to enforce security by dividing the system into trusted parts (HSM, server) and untrusted parts (host computer, the rest of the network). The trusted part makes cryptographic functions available via an API. How can we design an interface for a key management device that can create, delete, import and export keys from the device, as well as permitting their use for encryption, decryption, signature and verification, all so that if the device comes into contact with a malicious application we can be sure the keys stay secure? This is far from trivial, as well will see from our case study of the most widely used standard for such interfaces, RSA PKCS#11.

2.1 The PKCS#11 standard

RSA Public Key Cryptography Standards (PKCS) aim to standardise various aspects of cryptography to promote interoperability and security. PKCS#1, for example, defines the RSA asymmetric encryption and signing algorithm. PKCS#11 describes the 'Cryptoki' API for cryptographic hardware. Version 1.0 was published in 1995. The latest official version is v2.20 (2004) which runs to just under 400 pages [31]. Adoption of the standard is almost ubiquitous in commercial cryptographic tokens and smartcards, even if other interfaces are frequently offered in addition.

In a PKCS#11-based API, applications initiate a *session* with the cryptographic token, by supplying a PIN. Note that if malicious code is running on the host machine, then the user PIN may easily be intercepted, e.g. by a keylogger or by a tampered device driver, allowing an attacker to create his own sessions with the device, a point conceded in the security discussion in the standard [31, p. 31]. PKCS#11 is intended to protect its sensitive cryptographic keys even when connected to a compromised host.

Once a session is initiated, the application may access the *objects* stored on the token, such as keys and certificates. However, access to the objects is controlled. Objects are referenced in the API via *handles*, which can be thought of as pointers to or names for the objects. In general, the value of the handle, e.g. for a secret key, does not reveal any information about the actual value of the key. Objects have *attributes*, which may be bitstrings e.g. the value of a key, or Boolean flags signalling properties of the object, e.g. whether the key may be used for encryption, or for encrypting other keys. New objects can be created by

Initial knowledge: The intruder knows $\mathsf{h}(\mathsf{n}_1, \mathsf{k}_1)$ and $\mathsf{h}(\mathsf{n}_2, \mathsf{k}_2)$. The name n_2 has the attributes wrap and decrypt set whereas n_1 has the attribute sensitive and extractable set.

Trace:

Wrap: $\mathsf{h}(\mathsf{n}_2, \mathsf{k}_2),\ \mathsf{h}(\mathsf{n}_1, \mathsf{k}_1) \rightarrow \{\!|\mathsf{k}_1|\!\}_{\mathsf{k}_2}$
SDecrypt: $\mathsf{h}(\mathsf{n}_2, \mathsf{k}_2),\ \{\!|\mathsf{k}_1|\!\}_{\mathsf{k}_2} \rightarrow \mathsf{k}_1$

Fig. 1. Wrap/Decrypt attack.

calling a key generation command, or by 'unwrapping' an encrypted key packet. In both cases a new handle is returned.

When a function in the token's API is called with a reference to a particular object, the token first checks that the attributes of the object allow it to be used for that function. For example, if the encrypt function is called with the handle for a particular key, that key must have its encrypt attribute set. To protect a key from being revealed, the attribute sensitive must be set to true. This means that requests to view the object's key value via the API will result in an error message. Once the attribute sensitive has been set to true, it cannot be reset to false. This gives us the principal security property stated in the standard: attacks, even if they involve compromising the host machine, cannot "compromise keys marked 'sensitive', since a key that is sensitive will always remain sensitive", [31, p. 31]. Such a key may be exported outside the device if it is encrypted by another key, but only if its extractable attribute is set to true. An object with an extractable attribute set to false may not be read by the API, and additionally, once set to false, the extractable attribute cannot be set to true. Protection of the keys essentially relies on the sensitive and extractable attributes.

2.2 The Wrap-Decrypt Attack

Clulow first published attacks on PKCS#11 based APIs in 2003 [13], where he gave many examples of ways in which keys with the sensitive attribute set to true could be read in clear outside the device. The most straightforward of these is the 'key separation' attack, where the attributes of a key are set in such a way as to give a key conflicting roles. Clulow gives the example of a key with the attributes set for decryption of ciphertexts, and for 'wrapping', i.e. encryption of other keys for secure transport.

To determine the value of a sensitive key, the attacker simply wraps it and then decrypts it, as shown in Fig. 1. The attack is described with a notation for PKCS#11 based APIs briefly defined in the following and more formally in the next section: $\mathsf{h}(\mathsf{n}_1, \mathsf{k}_1)$ is a predicate stating that there is a handle encoded by n_1 for a key k_1 stored on the device. The symmetric encryption of k_1 under key k_2 is represented by $\{\!|\mathsf{k}_1|\!\}_{\mathsf{k}_2}$. Note also that according to the wrapping formats

defined in PKCS#11, the device cannot tell whether an arbitrary bitstring is a cryptographic key or some other piece of plaintext. Thus when it executes the decrypt command, it has no way of telling that the packet it is decrypting contains a key.

It might appear easy to prevent such an attack, but as we shall see, it is in fact rather difficult within the confines of PKCS#11. Before treating this in detail, we will introduce our language for formal modelling of the API.

2.3 Formal Model

Our model follows the approach used by Delaune, Kremer and Steel (DKS) [15]. The device is assumed to be connected to a host under the complete control of an intruder, representing a malicious piece of software. The intruder can call the commands of the API in any order he likes using any values that he knows. We abstract away details of the cryptographic algorithms in use, following the classical approach of Dolev and Yao [17]. Bitstrings are modelled as terms in an abstract algebra. The rules of the API and the abilities of an attacker are written as deduction rules in the algebra.

Basic Notions We assume a given *signature* Σ, i.e. a finite set of *function symbols*, with an arity function $ar : \Sigma \to \mathbb{N}$, a (possibly infinite) set of *names* $\mathcal{N}$ and a (possibly infinite) set of *variables* $\mathcal{X}$. Names represent keys, data values, nonces, etc. and function symbols model cryptographic primitives, e.g. $\{\!|\mathsf{x}|\!\}_\mathsf{y}$ representing symmetric encryption of plaintext x under key y, and $\{\mathsf{x}\}_\mathsf{y}$ representing public key encryption of x under y. Function symbols of arity 0 are called *constants*. This includes the Boolean constants true ($\top$) and false ($\bot$). The set of *plain terms* $\mathcal{PT}$ is defined by the following grammar:

$$\begin{array}{rll} t, t_i := & x & x \in \mathcal{X} \\ \mid & n & n \in \mathcal{N} \\ \mid & f(t_1, \ldots, t_n) & f \in \Sigma \text{ and } ar(f) = n \end{array}$$

We also consider a finite set $\mathcal{F}$ of predicate symbols, disjoint from Σ, from which we derive a set of *facts*. The set of *facts* is defined as

$$\mathcal{FT} = \{p(t, b) \mid p \in \mathcal{F}, t \in \mathcal{PT}, b \in \{\top, \bot\}\}$$

In this way, we can explicitly express the Boolean value b of an attribute p on a term t by writing $p(t, b)$. For example, to state that the key referred to by n has the wrap attribute set we write $\mathsf{wrap}(\mathsf{n}, \top)$.

The description of a system is given as a finite set of rules of the form

$$T; L \xrightarrow{\mathsf{new}\,\tilde{n}} T'; L'$$

where $T, T' \subseteq \mathcal{PT}$ are sets of plain terms $L, L' \subseteq \mathcal{F}$ are sets of facts and $\tilde{n} \subseteq \mathcal{N}$ is a set of names. The intuitive meaning of such a rule is the following. The rule

can be fired if all terms in T are in the intruder knowledge and if all the facts in L hold in the current state. The effect of the rule is that terms in T' are added to the intruder knowledge and the valuation of the attributes is updated to satisfy L'. The $\mathsf{new}\,\tilde{n}$ means that all the names in $\tilde{n}$ need to be replaced by fresh names in T' and L'. This allows us to model nonce or key generation: if the rule is executed several times, the effects are different as different names will be used each time.

Example 1. The following rule models wrapping:

$$\mathsf{h}(\mathsf{x}_1, \mathsf{y}_1), \mathsf{h}(\mathsf{x}_2, \mathsf{y}_2);\ \mathsf{wrap}(\mathsf{x}_1, \top), \mathsf{extract}(\mathsf{x}_2, \top) \rightarrow \{\!|\mathsf{y}_2|\!\}_{\mathsf{y}_1}$$

Intuitively, $\mathsf{h}(\mathsf{x}_1, \mathsf{y}_1)$ is a handle x_1 for key y_1 while term $\{\!|\mathsf{y}_2|\!\}_{\mathsf{y}_1}$ represents a key y_2 wrapped with y_1. Since the attribute wrap for key y_1 is set, noted as $\mathsf{wrap}(\mathsf{x}_1, \top)$, and key y_2 is extractable, written $\mathsf{extract}(\mathsf{x}_2, \top)$, then we can wrap y_2 with y_1, creating $\{\!|\mathsf{y}_2|\!\}_{\mathsf{y}_1}$.

The semantics of the model is defined in a standard way in terms of a transition system. Each state in the model consists of a set of terms in the intruder's knowledge, and a set of global state predicates. The intruder's knowledge increases monotonically with each transition (he can only learn more things), but the global state is non-monotonic (attributes may be set on and then off). For a formal semantics, we refer to the literature [15]. We now present in Fig. 2 a subset of PKCS#11 commands sufficient for some basic symmetric key management commands (the asymmetric key has also been treated in the literature [16]). This will suffice to demonstrate the modelling technique and some attacks.

2.4 Using the Formal Model

We first add some rules to the model for the intruder that allow him to encrypt and decrypt using his own keys (see Fig. 3). The intruder is assumed not to be able to crack the encryption algorithm by brute-force search or similar means, thus he can only read an encrypted message if he has the correct key. We analyse security as reachability, in the model, of *attack* states, i.e. states where the intruder knows the value of a key stored on the device with the $\mathsf{sensitive}$ attribute set to true, or the $\mathsf{extractable}$ attribute set to false. We give the intruder some initial knowledge, typically just some key $\mathsf{k_i}$ that is not loaded on the device, and then use a tool such as a model checker to search the model for a chain of commands and intruder steps that leads to an attack state.

Unfortunately we immediately encounter both theoretical and practical problems. First, we know that the reachability problem in general for languages like this is undecidable: even with fixed message size there is a reduction to the Post correspondence problem [19, table 12, page 298]. Even so we might hope to find some attacks in practice. But in fact the model checker is quickly swamped by the combinatorial possibilities, many of which at first sight seem unlikely to lead to an attack. For example, the intruder can take an already encrypted term $\{\!|\mathsf{k}_1|\!\}_{\mathsf{k}_2}$, and use it as input to the encrypt command along with some handle $\mathsf{h}(\mathsf{n}, \mathsf{k}_3)$ to

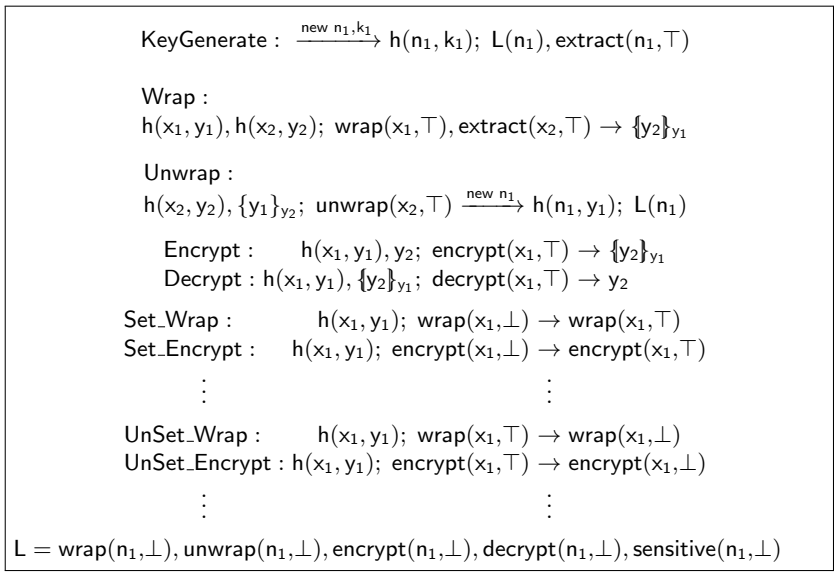


Fig. 2. PKCS#11 Symmetric Key Fragment.

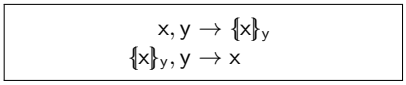


Fig. 3. Intruder rules for symmetric key cryptography.

obtain $\{\!|\{\!|k_1|\!\}_{k_2}|\!\}_{k_3}$. Unfortunately one cannot in general just delete these terms from the intruder knowledge: they may be a necessary step for some fiendish attack. Fortunately we can address both the the theoretical and practical problems at once, by means of a *well-modedness* result [16]. There it is shown that each function symbol such as $h(.,.)$, $\{\!|.|\!\}_.$ can be given a unique interpretation in terms of *modes*. For example, $h(.,.)$ has mode Nonce $\times$ Key $\rightarrow$ Handle. We assign to each constant symbol a mode. A term is well-moded if all the function symbols in it are applied to symbols such that the modes are respected. Furthermore, any reachability query that can be expressed with well-moded terms is satisfiable if and only if it it is reachable by a sequence of steps where the intruder learns only well-moded terms. This allows us to prune the search space dramatically, since any branch that results in the creation of an ill-moded term can be ignored. If the function symbols can be moded acyclically, we can also show decidability provided we bound fresh handles and keys. Consult the paper for details and proofs [16].

2.5 A Suite of Attacks

Equipped with a suitable formal model and a model checker, we can attempt to find secure configurations of the standard. Clulow's first suggestion for preventing the attack in Fig. 1 is to prevent attribute changing operations from allowing a stored key to have both wrap and decrypt set. Note that in order to do this, it is not sufficient merely to check that decrypt is unset before setting wrap, and to check wrap is unset before setting decrypt. One must also add wrap and decrypt to a list of 'sticky' attributes which once set, may not be unset, or the attack is not prevented, [33]. Effectively this means the unset rules will be omitted from the model for these attributes. Having applied these measures, we discover the attack given in Fig. 4. The intruder imports his own key k_3 by first encrypting it under k_2, and then unwrapping it. He can then export the sensitive key k_1 under k_3 to discover its value.

Initial state: The intruder knows the handles $\mathsf{h}(\mathsf{n}_1, \mathsf{k}_1)$, $\mathsf{h}(\mathsf{n}_2, \mathsf{k}_2)$ and the key k_3; n_1 has the attributes sensitive and extract set whereas n_2 has the attributes unwrap and encrypt set.

Trace:

$$
\begin{array}{lrcl}
\text{Encrypt:} & \mathsf{h}(\mathsf{n}_2, \mathsf{k}_2),\ \mathsf{k}_3 & \rightarrow & \{\!|\mathsf{k}_3|\!\}_{\mathsf{k}_2} \\
\text{Unwrap:} & \mathsf{h}(\mathsf{n}_2, \mathsf{k}_2),\ \{\!|\mathsf{k}_3|\!\}_{\mathsf{k}_2} & \xrightarrow{\text{new } \mathsf{n}_3} & \mathsf{h}(\mathsf{n}_3, \mathsf{k}_3) \\
\text{Set\_wrap:} & \mathsf{h}(\mathsf{n}_3, \mathsf{k}_3) & \rightarrow & \mathsf{wrap}(\mathsf{n}_3, \top) \\
\text{Wrap:} & \mathsf{h}(\mathsf{n}_3, \mathsf{k}_3),\ \mathsf{h}(\mathsf{n}_1, \mathsf{k}_1) & \rightarrow & \{\!|\mathsf{k}_1|\!\}_{\mathsf{k}_3} \\
\text{Intruder:} & \{\!|\mathsf{k}_1|\!\}_{\mathsf{k}_3},\ \mathsf{k}_3 & \rightarrow & \mathsf{k}_1
\end{array}
$$

Fig. 4. Attack using encrypt and unwrap.

To prevent the attack shown in Fig. 4, we add encrypt and unwrap to the list of conflicting attribute pairs. Another new attack is discovered (see Fig. 5) of a type discussed by Clulow, [13, Section 2.3]. Here the key k_2 is first wrapped under k_2 itself, and then unwrapped, gaining a new handle $\mathsf{h}(\mathsf{n}_4, \mathsf{k}_2)$. The intruder then wraps k_1 under k_2, and sets the decrypt attribute on handle $\mathsf{h}(\mathsf{n}_4, \mathsf{k}_2)$, allowing him to obtain k_1.

One can attempt to prevent the attack in Fig. 5 by adding wrap and unwrap to our list of conflicting attribute pairs. Now in addition to the initial knowledge from the first three experiments, we give the intruder an unknown key k3 encrypted under k2. Again he is able to affect an attack similar to the one above, this time by unwrapping $\{\!|\mathsf{k}_3|\!\}_{\mathsf{k}_2}$ twice (see Fig. 6).

This sample of the attacks found show how difficult PKCS#11 is to configure in a safe way, and indeed there are several more attacks documented in the literature [8, 13, 16, 21] and perhaps more to discover. Another line of research has consisted of trying to propose secure subsets of the API together with a suitable security proof. An obstacle here is the fresh generation of keys and handles: if

Initial state: The intruder knows the handles $\mathsf{h}(\mathsf{n}_1, \mathsf{k}_1)$, $\mathsf{h}(\mathsf{n}_2, \mathsf{k}_2)$; n_1 has the attributes sensitive, extract and whereas n_2 has the attribute extract set.

Trace:

$$
\begin{array}{lrcl}
\mathsf{Set\_wrap}: & \mathsf{h}(\mathsf{n}_2, \mathsf{k}_2) & \rightarrow & \mathsf{wrap}(\mathsf{n}_2, \top) \\
\mathsf{Wrap}: & \mathsf{h}(\mathsf{n}_2, \mathsf{k}_2), \mathsf{h}(\mathsf{n}_2, \mathsf{k}_2) & \rightarrow & \{\mathsf{k}_2\}_{\mathsf{k}_2} \\
\mathsf{Set\_unwrap}: & \mathsf{h}(\mathsf{n}_2, \mathsf{k}_2) & \rightarrow & \mathsf{unwrap}(\mathsf{n}_2, \top) \\
\mathsf{Unwrap}: & \mathsf{h}(\mathsf{n}_2, \mathsf{k}_2), \{\mathsf{k}_2\}_{\mathsf{k}_2} & \xrightarrow{\text{new } \mathsf{n}_4} & \mathsf{h}(\mathsf{n}_4, \mathsf{k}_2) \\
\mathsf{Wrap}: & \mathsf{h}(\mathsf{n}_2, \mathsf{k}_2), \mathsf{h}(\mathsf{n}_1, \mathsf{k}_1) & \rightarrow & \{\mathsf{k}_1\}_{\mathsf{k}_2} \\
\mathsf{Set\_decrypt}: & \mathsf{h}(\mathsf{n}_4, \mathsf{k}_2) & \rightarrow & \mathsf{decrypt}(\mathsf{n}_4, \top) \\
\mathsf{Decrypt}: & \mathsf{h}(\mathsf{n}_4, \mathsf{k}_2), \{\mathsf{k}_1\}_{\mathsf{k}_2} & \rightarrow & \mathsf{k}_1
\end{array}
$$

Fig. 5. Re-import attack 1.

Initial state: The intruder knows the handles $\mathsf{h}(\mathsf{n}_1, \mathsf{k}_1)$, $\mathsf{h}(\mathsf{n}_2, \mathsf{k}_2)$; n_1 has the attributes sensitive, extract and whereas n_2 has the attribute extract set. The intruder also knows $\{\mathsf{k}_3\}_{\mathsf{k}_2}$.

Trace:

$$
\begin{array}{lrcl}
\mathsf{Set\_unwrap}: & \mathsf{h}(\mathsf{n}_2, \mathsf{k}_2) & \rightarrow & \mathsf{unwrap}(\mathsf{n}_2, \top) \\
\mathsf{Unwrap}: & \mathsf{h}(\mathsf{n}_2, \mathsf{k}_2), \{\mathsf{k}_3\}_{\mathsf{k}_2} & \xrightarrow{\text{new } \mathsf{n}_3} & \mathsf{h}(\mathsf{n}_3, \mathsf{k}_3) \\
\mathsf{Unwrap}: & \mathsf{h}(\mathsf{n}_2, \mathsf{k}_2), \{\mathsf{k}_3\}_{\mathsf{k}_2} & \xrightarrow{\text{new } \mathsf{n}_4} & \mathsf{h}(\mathsf{n}_4, \mathsf{k}_3) \\
\mathsf{Set\_wrap}: & \mathsf{h}(\mathsf{n}_3, \mathsf{k}_3) & \rightarrow & \mathsf{wrap}(\mathsf{n}_3, \top) \\
\mathsf{Wrap}: & \mathsf{h}(\mathsf{n}_3, \mathsf{k}_3), \mathsf{h}(\mathsf{n}_1, \mathsf{k}_1) & \rightarrow & \{\mathsf{k}_1\}_{\mathsf{k}_3} \\
\mathsf{Set\_decrypt}: & \mathsf{h}(\mathsf{n}_4, \mathsf{k}_3) & \rightarrow & \mathsf{decrypt}(\mathsf{n}_4, \top) \\
\mathsf{Decrypt}: & \mathsf{h}(\mathsf{n}_4, \mathsf{k}_3), \{\mathsf{k}_1\}_{\mathsf{k}_3} & \rightarrow & \mathsf{k}_1
\end{array}
$$

Fig. 6. Re-import attack 2.

there are no attacks in the model after generating n fresh keys, how do we know there are no attacks after generating $n+1$? To address this problem, Fröschle and Steel proposed abstractions for handles and keys that allow security proofs for unbounded fresh data [21]. In particular, they showed the security of a symmetric key management subset based around the proprietary extensions to PKCS#11 made by Eracom, where keyed hashes are used to bind attributes to keys under wrapping, ensuring that they are re-imported with the same attributes.

2.6 Finding Attacks on Real Devices

Attacks on the standard are interesting in themselves, but in reality every device implements a different subset of PKCS#11 with different restrictions on the use of each command. How can one know whether a particular device is vulnerable? To address this, Bortolozzo, Centenaro, Focardi and Steel developed the

Fig. 7. Tookan system diagram.

Tookan[1] tool [8]. Tookan functions as shown in Fig. 7. First, Tookan extracts the capabilities of the token following a reverse engineering process (1). The results of this task are written in a meta-language for PKCS#11 models. Tookan uses this information to generate a model the language described above (2), which is encoded for input to the SATMC model checker [3]. If SATMC finds an attack, the attack trace (3) is sent to Tookan for testing directly on the token (4).

Changes to the model There are several differences between Tookan's model and the model of section 2.3. One is that Tookan takes into account *key templates*. In section 2.3, the key generation commands create a key with all attributes unset (see Fig. 2). Attributes are then enabled one by one using the SetAttribute command. In our experiments with real devices, we discovered that some tokens do not allow attributes of a key to be changed. Instead, they use a key template specifying settings for the attributes which are given to freshly generated keys. Templates are used for the import of encrypted keys (unwrapping), key creation using CreateObject and key generation. The template to be used in a specific command instance is specified as a parameter, and must come from a set of valid templates, which we label $\mathcal{G}, \mathcal{C}$ and $\mathcal{U}$ for the valid templates for key generation, creation and unwrapping respectively. Tookan can construct the set of templates in two ways: the first, by exhaustively testing the commands using templates for all possible combinations of attribute settings, which may be very time consuming, but is necessary if we aim to verify the security of a token. The second method is to construct the set of templates that should be allowed based on the reverse-engineered attribute policy (see next paragraph). This is an approximate process, but can be useful for quickly finding attacks. Indeed, in our experiments, we found that these models reflected well the operation of the token, i.e. the attacks found by the model checker all executed on the tokens without any 'template invalid' errors.

Attribute Policies Most tokens tested attempt to impose some restrictions on the combinations of attributes that can be set on a key and how these may be changed. There are four kinds of restriction that Tookan can infer from its reverse engineering process:

`Sticky_on` These are attributes that once set, may not be unset. The PKCS #11 standard lists some of these [31, Table 15]: sensitive for secret keys, for

[1] Tool for cryptoki analysis.

example. The UnsetAttribute rule is only included for attributes which are not sticky on. To test if a device treats an attribute as sticky on, Tookan attempts to create a key with the attribute on, and then calls SetAttribute to change the attribute to off.

`Sticky_off` These are attributes that once unset may not be set. In the standard, extractable is listed as such an attribute. The SetAttribute rule is only included for attributes which are not sticky off. To test if a device treats an attribute as sticky on, Tookan attempts to create a key with the attribute off, and then calls SetAttribute to change the attribute to on.

`Conflicts` Many tokens (appear to) disallow certain pairs of attributes to be set, either in the template or when changing attributes on a live key. For example, some tokens do not allow sensitive and extractable to be set on the same key. The SetAttribute rule is adjusted to prevent conflicting attributes from being set on an object or on the template. When calculating the template sets $\mathcal{C}, \mathcal{G}, \mathcal{U}$ (see above), we forbid templates which have both the conflicting attributes set. To test if a device treats an attribute pair as a conflict, Tookan attempts to generate a key with the the pair of attributes set, then if no error is reported, it calls GetAttribute to check that the token really has created a key with the desired attributes set.

`Tied` Some tokens automatically set the value of some attributes based on the value of others. For example, many tokens set the value of always_sensitive based on the value of the attribute sensitive. The SetAttribute and UnsetAttribute rules are adjusted to account for tied attributes. The template sets $\mathcal{C}, \mathcal{G}, \mathcal{U}$ are also adjusted accordingly. To test if a device treats an attribute pair as tied, Tookan attempts to generate a key with some attribute a on and all other attributes off. It then uses GetAttribute to examine the key as it was actually created, and tests to see if any other attributes were turned on.

Limitations of Reverse Engineering Tookan's reverse engineering process is not complete: it may result in a model that is too restricted to find some attacks possible on the token, and it may suggest false attacks which cannot be executed on the token. This is because in theory, no matter what the results of our finite test sequence, the token may be running any software at all, perhaps even behaving randomly. However, if a token implements its attribute policy in the manner in which we can describe it, i.e. as a combination of sticky on, sticky off, conflict and tied attributes, then our process is complete in the sense that the model built will reflect exactly what the token can do (modulo the usual Dolev-Yao abstractions for cryptography).

In our testing, the model performed very well: the Tookan consistently found true attacks on flawed tokens, and we were unable to find 'by hand' any attacks on tokens which the model checker deemed secure. This suggests that real devices do indeed implement their attribute policies in a manner similar to our model.

	Device		Supported Functionality						Attacks found					
	Company	Model	sym	asym	cobj	chan	w	ws	a1	a2	a3	a4	a5	mc
USB	Aladdin	eToken PRO	✓	✓	✓	✓	✓	✓	✓	✓				a1
	Athena	ASEKey	✓	✓	✓									
	Bull	Trustway RCI	✓	✓	✓	✓	✓	✓	✓	✓				a1
	Eutron	Crypto Id. ITSEC		✓	✓									
	Feitian	StorePass2000	✓	✓	✓	✓	✓	✓		✓	✓	✓		a3
	Feitian	ePass2000	✓	✓	✓	✓	✓	✓		✓	✓	✓		a3
	Feitian	ePass3003Auto	✓	✓	✓	✓	✓	✓		✓	✓	✓		a3
	Gemalto	SEG		✓		✓								
	MXI Security	Stealth MXP Bio	✓	✓		✓								
	RSA	SecurID 800	✓	✓	✓	✓					✓	✓	✓	a3
	SafeNet	iKey 2032	✓	✓	✓		✓							
	Sata	DKey	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	a3
Card	ACS	ACOS5	✓	✓	✓	✓								
	Athena	ASE Smartcard	✓	✓	✓									
	Gemalto	Cyberflex V2	✓	✓	✓		✓	✓		✓				a2
	Gemalto	Classic TPC IS V1		✓		✓								
	Gemalto	Classic TPC IS V2	✓	✓	✓	✓	✓	✓	✓		✓	✓	✓	a3
	Siemens	CardOS V4.3 B	✓	✓	✓		✓					✓		a4
Soft	Eracom	HSM simulator	✓	✓		✓	✓	✓	✓	✓		✓		a1
	IBM	opencryptoki 2.3.1	✓	✓	✓	✓	✓	✓	✓	✓		✓		a1

Table 1. Summary of results on devices.

2.7 Results

Table 1 summarises the results obtained by Tookan on a number of devices as well as two software simulators. Supported functionality and attacks are summarized in Table 2 and described below.

Implemented functionality Columns 'sym' and 'asym' respectively indicate whether or not symmetric and asymmetric key cryptography are supported. Column 'cobj' refers to the possibility of inserting external, unencrypted, keys on the device via `C_CreateObject` PKCS#11 function. This is allowed by almost all of the analysed tokens, although it wasn't included in the original model of the standard used by Delaune, Kremer and Steel [16]. The next column, 'chan', refers to the possibility of changing key attributes through `C_SetAttributeValue`. The following two columns, 'w' and 'ws', respectively indicate whether the token permits wrapping of nonsensitive and sensitive keys.

Attacks Attack a1 is a wrap/decrypt attack as discussed in section 2.2. The attacker exploits a key k_2 with attributes wrap and decrypt and uses it to attack

	Acronym	Description
Supported functionality	sym	symmetric-key cryptography
	asym	asymmetric-key cryptography
	cobj	inserting new keys via `C_CreateObject`
	chan	changing key attributes
	w	wrapping keys
	ws	wrapping sensitive keys
Attacks	a1	wrap/decrypt attack based on symmetric keys
	a2	wrap/decrypt attack based on asymmetric keys
	a3	sensitive keys are directly readable
	a4	unextractable keys are directly readable (forbidden by the standard)
	a5	sensitive/unextractable keys can be changed into nonsensitive/extractable
	mc	first attack found by Tookan

Table 2. Key for table 1.

a sensitive key k_1. Using our notation from section 2.3:

$$\begin{array}{ll} \mathsf{Wrap}: & \mathsf{h}(\mathsf{n}_2,\mathsf{k}_2),\ \mathsf{h}(\mathsf{n}_1,\mathsf{k}_1) \rightarrow \{\!|\mathsf{k}_1|\!\}_{\mathsf{k}_2} \\ \mathsf{Decrypt}: & \mathsf{h}(\mathsf{n}_2,\mathsf{k}_2),\ \{\!|\mathsf{k}_1|\!\}_{\mathsf{k}_2} \rightarrow \mathsf{k}_1 \end{array}$$

As we have discussed above, the possibility of inserting new keys in the token (column 'cobj') might simplify further the attack. It is sufficient to add a known wrapping key:

$$\begin{array}{ll} \mathsf{CreateObject}: & \mathsf{k}_2 \xrightarrow{\mathsf{new}\ \mathsf{n}_2} \mathsf{h}(\mathsf{n}_2,\mathsf{k}_2) \\ \mathsf{Wrap}: & \mathsf{h}(\mathsf{n}_2,\mathsf{k}_2),\ \mathsf{h}(\mathsf{n}_1,\mathsf{k}_1) \rightarrow \{\!|\mathsf{k}_1|\!\}_{\mathsf{k}_2} \end{array}$$

The attacker can then decrypt $\{\!|\mathsf{k}_1|\!\}_{\mathsf{k}_2}$ since he knows key k_2. SATMC discovered this variant of the attack on several vulnerable tokens. Despite its apparent simplicity, this attack had not appeared before in the PKCS#11 security literature [13, 15].

Attack a2 is a variant of the previous ones in which the wrapping key is a public key $\mathsf{pub}(\mathsf{z})$ and the decryption key is the corresponding private key $\mathsf{priv}(\mathsf{z})$:

$$\begin{array}{ll} \mathsf{Wrap}: & \mathsf{h}(\mathsf{n}_2,\mathsf{pub}(\mathsf{z})),\ \mathsf{h}(\mathsf{n}_1,\mathsf{k}_1) \rightarrow \{\mathsf{k}_1\}_{\mathsf{pub}(\mathsf{z})} \\ \mathsf{ADecrypt}: & \mathsf{h}(\mathsf{n}_2,\mathsf{priv}(\mathsf{z})),\ \{\mathsf{k}_1\}_{\mathsf{pub}(\mathsf{z})} \rightarrow \mathsf{k}_1 \end{array}$$

In this case too, the possibility of importing key pairs simplifies even more the attacker's task by allowing him to import a public wrapping key while knowing the corresponding private key. Once the wrap of the sensitive key has been performed, the attacker can decrypt the obtained ciphertext using the private key.

Attack a3 is a clear flaw in the PKCS#11 implementation. It is explicitly required that the value of sensitive keys should never be communicated outside

the token. In practice, when the token is asked for the value of a sensitive key, it should return some "value is sensitive" error code. Instead, we found that some of the analysed devices just return the plain key value, ignoring this basic policy. Attack a4 is similar to a3: PKCS#11 requires that keys declared to be unextractable should not be readable, even if they are nonsensitive. If they are in fact readable, this is another violation of PKCS#11 security policy.

Finally, attack a5 refers to the possibility of changing sensitive and unextractable keys respectively into nonsensitive and extractable ones. Only the Sata and Gemalto SafeSite Classic V2 tokens allow this operation. However, notice that this attack is not adding any new flaw for such devices, given that attacks a3 and a4 are already possible and sensitive or unextractable keys are already accessible.

Model-checking results Column 'mc' reports which of the attacks was automatically rediscovered via model-checking. SATMC terminates once it has found an attack, hence we report the attack that was found first. Run-times for finding the attacks vary from a couple of seconds to just over 3 minutes.

2.8 Finding Secure Configurations

As we observed in the last section, none of the tokens we tested are able to import and export sensitive keys in a secure fashion. In particular, all the analysed tokens are either insecure or have been drastically restricted in their functionality, e.g. by completely disabling wrap and unwrap. In this section, we present CryptokiX, a software implementation of a Cryptoki token which can be fiXed by selectively enabling different patches. The starting point is openCryptoki [30], an open-source PKCS#11 implementation for Linux including a software token for testing. As shown in Table 1, the analysis of openCryptoki software token has revealed that it is subject to all the non-trivial attacks. This is in a sense expected, as it implements the standard 'as is', i.e., with no security patches. CryptokiX extends openCryptoki with:

Conflicting attributes We have seen, for example, that it is insecure to allow the same key to be used for wrapping and decrypting. In CryptokiX it is possible to specify a set of conflicting attributes.

Sticky attributes We know that some attributes should always be sticky, such as sensitive. This is also useful when combined with the 'conflicting attributes' patch above: if wrap and decrypt are conflicting, we certainly want to avoid that the wrap attribute can be unset so as to allow the decrypt attribute to be set.

Wrapping formats It has been shown that specifying a non-conflicting attribute policy is not sufficient for security [13, 15]. A wrapping format should also be used to correctly bind key attributes to the key. This prevents attacks where the key is unwrapped twice with conflicting attributes.

Secure templates We limit the set of admissible attribute combinations for keys in order to avoid that they ever assume conflicting roles at creation time.

This is configurable at the level of the specific PKCS#11 operation. For example, we can define different secure templates for different operations such as key generation and unwrapping.

A way to combine the first three patches with a wrapping format that binds attributes to keys in order to create a secure token has already been demonstrated [21] and discussed in section 2.5. One can also use the fourth patch to produce a secure configuration that does not require any new cryptographic mechanisms to be added to the standard, making it quite simple and cheap to incorporate into existing devices. We consider here a set of templates with attributes sensitive and extractable always set. Other attributes wrap, unwrap, encrypt and decrypt are set as follows:

Key generation We allow three possible templates:

1. wrap and unwrap, for exporting/importing other keys;
2. encrypt and decrypt, for cryptographic operations;
3. neither of the four attributes set, i.e. the default template if none of the above is specified.

Key creation/import We allow two possible templates for any key created with `CreateObject` or imported with `Unwrap`:

1. unwrap,encrypt set and wrap,decrypt unset;
2. none of the four attributes set.

The templates for key generation are rather intuitive and correspond to a clear separation of key roles, which seems a sound basis for a secure configuration. The rationale behind the single template for key creation/import, however, is less obvious and might appear rather restrictive. The idea is to allow wrapping and unwrapping of keys while 'halving' the functionality of created/unwrapped keys: these latter keys can only be used to unwrap other keys or to encrypt data, wrapping and decrypting under such keys are forbidden. This, in a sense, offers an asymmetric usage of imported keys: to achieve full-duplex encrypted communication two devices will each have to wrap and send a freshly generated key to the other device. Once the keys are unwrapped and imported in the other devices they can be used to encrypt outgoing data in the two directions. Notice that imported keys can never be used to wrap sensitive keys. Note also that we require that all attributes are sticky on and off, and that we assume for bootstrapping that any two devices that may at some point wish to communicate have a shared long term symmetric key installed on them at personalisation time. This need only be used once in each direction. Our solution works well for pairwise communication, where the overhead is just one extra key, but would be more cumbersome for group key sharing applications.

The developed solution has been implemented and analysed by extracting a model using Tookan. A model for SATMC was constructed using the abstractions described above (see end of section 2.5). Given the resulting model, SATMC terminates with no attacks in a couple of seconds, allowing us to conclude the patch is safe in our abstract model for unbounded numbers of fresh keys and handles.

Note that although no sensitive keys can be extracted by an intruder, there is of course no integrity check on the wrapped keys that are imported. Indeed, without having an encryption mode with an integrity check this would seem to be impossible. This means that one cannot be sure that a key imported on to the device really corresponds to a key held securely on the intended recipient's device. This limitation would have to be taken into account when evaluating the suitability of this configuration for an application. CryptokiX is available online[2].

3 Conclusions

We have seen how RSA PKCS#11 describes an API for key management where the usage policy for each key is described by a set of attributes. This interface, if not configured carefully, can be vulnerable to a variety of attacks, and we have seen that these vulnerabilities affect not just theoretical models but real devices. We have shown how to use formal modelling techniques to systematically find attacks and then verify the security of models. We have also seen how Tookan can help to automate this process. We saw that it is possible to propose secure configurations, [8, 21], but security proofs here are only in the symbolic model of cryptography: there is more work to be done to reconcile these proofs to the cryptographic details of real implementations, though work in this direction is underway [26]. There have also been articles proposing completely new designs for key management APIs [10, 14]. Indeed at least two new industrial standards which address key management are currently at the draft stage: IEEE 1619.3 [23] (for secure storage) and OASIS Key Management Interoperability Protocol (KMIP) [29]. It remains to be seen how these latter designs will address security concerns.

Security API analysis has also proved useful in other domains. For example, in a longer version of this article co-authored with Focardi and Luccio, we explore the use of API analysis in PIN processing APIs such as are used in the international cash machine networks [20]. Here the problem is to ensure that no combination of API calls can leak the value of a customer PIN.

Our experience has shown that automated tools provide a way for engineers without a formal methods background to benefit from the power of these techniques. We intend to continue researching automated formal analysis tools for API security problems.

References

1. R. Anderson. The correctness of crypto transaction sets. In *8th International Workshop on Security Protocols*, April 2000. `http://www.cl.cam.ac.uk/ftp/users/rja14/protocols00.pdf`.

[2] `http://secgroup.ext.dsi.unive.it/cryptokix`.

2. A. Armando, D.A. Basin, Y. Boichut, Y. Chevalier, L. Compagna, J. Cuéllar, P. Hankes Drielsma, P. Héam, O. Kouchnarenko, J. Mantovani, S. Mödersheim, D. von Oheimb, M. Rusinowitch, J. Santiago, M. Turuani, L. Viganò, and L. Vigneron. The AVISPA tool for the automated validation of internet security protocols and applications. In Kousha Etessami and Sriram K. Rajamani, editors, *CAV*, volume 3576 of *Lecture Notes in Computer Science*, pages 281–285. Springer, 2005.
3. A. Armando and L. Compagna. SAT-based model-checking for security protocols analysis. *Int. J. Inf. Sec.*, 7(1):3–32, 2008. Software available at `http://www.ai-lab.it/satmc`. Currently developed under the AVANTSSAR project, `http://www.avantssar.eu`.
4. B. Blanchet. From secrecy to authenticity in security protocols. In M. Hermenegildo and G. Puebla, editors, *International Static Analysis Symposium (SAS'02)*, volume 2477 of *Lecture Notes in Computer Science*, pages 342–359, Madrid, Spain, September 2002.
5. M. Bond. Attacks on cryptoprocessor transaction sets. In *Proceedings of the 3rd International Workshop on Cryptographic Hardware and Embedded Systems (CHES'01)*, volume 2162 of *LNCS*, pages 220–234, Paris, France, 2001. Springer.
6. M. Bond and R. Anderson. API level attacks on embedded systems. *IEEE Computer Magazine*, 34(10):67–75, October 2001.
7. M. Bond and J. Clulow. Extending security protocol analysis: New challenges. *Electronic Notes in Theoretical Computer Science*, 125(1):13–24, 2005.
8. M. Bortolozzo, M. Centenaro, R. Focardi, and G. Steel. Attacking and fixing PKCS#11 security tokens. In *Proceedings of the 17th ACM Conference on Computer and Communications Security (CCS'10)*, pages 260–269, Chicago, Illinois, USA, October 2010. ACM Press.
9. C. Cachin and J. Camenisch. Encrypting keys securely. *IEEE Security & Privacy*, 8(4):66–69, 2010.
10. C. Cachin and N. Chandran. A secure cryptographic token interface. In *Computer Security Foundations (CSF-22)*, pages 141–153, Long Island, New York, 2009. IEEE Computer Society Press.
11. R. Clayton and M. Bond. Experience using a low-cost FPGA design to crack DES keys. In *Cryptographic Hardware and Embedded System - CHES 2002*, pages 579–592, 2002.
12. J. Clulow. The design and analysis of cryptographic APIs for security devices. Master's thesis, University of Natal, Durban, 2003.
13. J. Clulow. On the security of PKCS#11. In *Proceedings of the 5th International Workshop on Cryptographic Hardware and Embedded Systems (CHES'03)*, volume 2779 of *LNCS*, pages 411–425. Springer, 2003.
14. V. Cortier and G. Steel. A generic security API for symmetric key management on cryptographic devices. In Michael Backes and Peng Ning, editors, *Proceedings of the 14th European Symposium on Research in Computer Security (ESORICS'09)*, volume 5789 of *Lecture Notes in Computer Science*, pages 605–620, Saint Malo, France, September 2009. Springer.
15. S. Delaune, S. Kremer, and G. Steel. Formal analysis of PKCS#11. In *Proceedings of the 21st IEEE Computer Security Foundations Symposium (CSF'08)*, pages 331–344, Pittsburgh, PA, USA, June 2008. IEEE Computer Society Press.
16. S. Delaune, S. Kremer, and G. Steel. Formal analysis of PKCS#11 and proprietary extensions. *Journal of Computer Security*, 18(6):1211–1245, November 2010.
17. D. Dolev and A. Yao. On the security of public key protocols. *IEEE Transactions in Information Theory*, 2(29):198–208, March 1983.

18. A. Durante, R. Focardi, and R. Gorrieri. A compiler for analyzing cryptographic protocols using noninterference. *ACM Transactions on Software Engineering and Methodology*, 9(4):488–528, 2000.
19. N.A. Durgin, P. Lincoln, and J.C. Mitchell. Multiset rewriting and the complexity of bounded security protocols. *Journal of Computer Security*, 12(2):247–311, 2004.
20. Riccardo Focardi, Flaminia L. Luccio, and Graham Steel. An introduction to security API analysis. In Alessandro Aldini and Roberto Gorrieri, editors, *Foundations of Security Analysis and Design – FOSAD Tutorial Lectures (FOSAD'VI)*, volume 6858 of *Lecture Notes in Computer Science*, pages 35–65. Springer, September 2011.
21. S. Fröschle and G. Steel. Analysing PKCS#11 key management APIs with unbounded fresh data. In Pierpaolo Degano and Luca Viganò, editors, *Revised Selected Papers of the Joint Workshop on Automated Reasoning for Security Protocol Analysis and Issues in the Theory of Security (ARSPA-WITS'09)*, volume 5511 of *Lecture Notes in Computer Science*, pages 92–106, York, UK, August 2009. Springer.
22. J. Herzog. Applying protocol analysis to security device interfaces. *IEEE Security & Privacy Magazine*, 4(4):84–87, July-Aug 2006.
23. IEEE 1619.3 Technical Committee. IEEE storage standard 1619.3 (key management) (draft). available from `https://siswg.net/`.
24. International Organization for Standardization. ISO 9564-1: Banking personal identification number (PIN) management and security. 30 pages.
25. G. Keighren. Model checking security APIs. Master's thesis, University of Edinburgh, 2007.
26. S. Kremer, G. Steel, and B. Warinschi. Security for key management interfaces. In *Proceedings of the 24th IEEE Computer Security Foundations Symposium (CSF'11)*, Cernay-la-Ville, France, June 2011. IEEE Computer Society Press. To appear.
27. D. Longley and S. Rigby. An automatic search for security flaws in key management schemes. *Computers and Security*, 11(1):75–89, March 1992.
28. G. Lowe. Breaking and fixing the Needham Schroeder public-key protocol using FDR. In *Proceedings of TACAS*, volume 1055, pages 147–166. Springer Verlag, 1996.
29. OASIS Key Management Interoperability Protocol (KMIP) Technical Committee. KMIP – key management interoperability protocol. available from `http://xml.coverpages.org/KMIP/`, february 2009.
30. openCryptoki. `http://sourceforge.net/projects/opencryptoki/`.
31. RSA Security Inc., v2.20. *PKCS #11: Cryptographic Token Interface Standard.*, June 2004.
32. B. Schneier. *Applied Cryptography*. John Wiley and Sons, 2nd edition, 1996.
33. E. Tsalapati. Analysis of PKCS#11 using AVISPA tools. Master's thesis, University of Edinburgh, 2007.
34. P. Youn, B. Adida, M. Bond, J. Clulow, J. Herzog, A. Lin, R. Rivest, and R. Anderson. Robbing the bank with a theorem prover. Technical Report UCAM-CL-TR-644, University of Cambridge, August 2005.

Committees

Programme Committee:

- Alessandro Barenghi, *Politecnico di Milano, Italy.*
- Gilles Barthe, *Fundación IMDEA Software, Spain.*
- Loïc Correnson, *CEA LIST, France.*
- Emmanuelle Encrenaz, *LIP6, France.*
- Naofumi Homma, *Tohoku U., Japan.*
- Éliane Jaulmes, *ANSSI, France.*
- Gerwin Klein, *NICTA, Australia.*
- Debdeep Mukhopadhyay, *IIT Kharagpur, India.*
- Svetla Nikova, *K.U.Leuven, Belgium.*
- Renaud Pacalet, *TELECOM-ParisTech, France.*
- Bruno Robisson, *ENSMSE, France.*
- Timothy Sherwood, *UCSB, USA.*
- Graham Steel, *LSV, France.*

Steering committee:

- Sylvain Guilley, *TELECOM-ParisTech, France.*
- Çetin Kaya Koç, *UCSB, USA.*
- David Naccache, *ENS, France.*
- Akashi Satoh, *AIST, Japan.*
- Werner Schindler, *BSI, Germany.*

Local committee:

- Prof. Jean-Luc Danger, *TELECOM-ParisTech, France.*
- Dr. Svetla Nikova, *K.U.Leuven, Belgium.*
- Prof. Ingrid Verbauwhede, *K.U.Leuven, Belgium.*

Sub-reviewers:

- David Cock, *CSE/NICTA, Australia.*
- Chester D. Rebeiro, *IIT Kharagpur, India.*