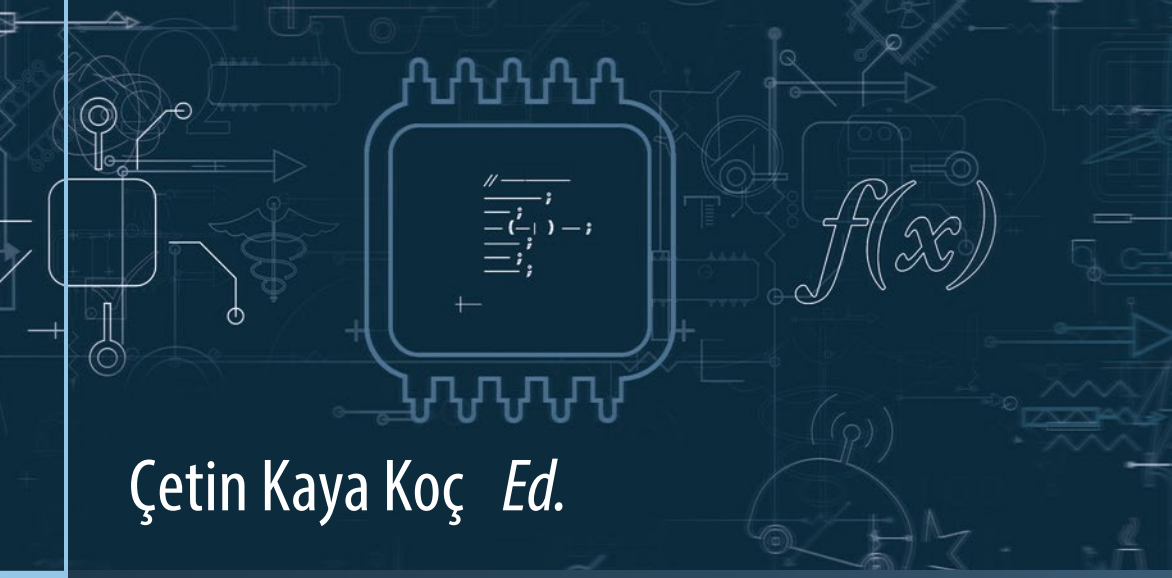


Çetin Kaya Koç *Ed.*

Cyber-Physical Systems Security

Springer

Cyber-Physical Systems Security

Çetin Kaya Koç
Editor

Cyber-Physical Systems Security

Springer

Editor
Çetin Kaya Koç
İstinye University, İstanbul, Turkey

Nanjing University of Aeronautics and
Astronautics, Nanjing, China

University of California Santa Barbara
Santa Barbara, CA, USA

ISBN 978-3-319-98934-1 ISBN 978-3-319-98935-8 (eBook)
https://doi.org/10.1007/978-3-319-98935-8

Library of Congress Control Number: 2018963758

This Springer imprint is published by the registered company Springer Nature Switzerland AG
The registered company address is: Gewerbestrasse 11, 6330 Cham, Switzerland

Preface

Cyber-physical systems involve interactions between computer-controlled and actuator-enabled components, whose dynamics have thus far been modeled, designed, and analyzed separately, but now need to be investigated and taught jointly. They control much of the world's critical infrastructure: power generation, telecommunications, water supply, and industrial control systems. Because of their critical nature, known, predictable, and secure behavior of cyber-physical systems is necessary to ensure the safety of the people whom these systems serve. Yet most cyber-physical systems provide limited operational guarantee outside of nominal conditions.

Understanding how diverse physical and digital systems can be safely and securely combined is not a simple task. Solving the safety and security deficiencies in next generation cyber-physical systems will require contributions from every branch of engineering, from mechanical and power engineering to computer science and mathematics. Partners from university research labs, governments, and industry must come together. It is necessary that we establish an engaged, multidisciplinary cyber-physical security community committed to developing unified foundations, principles, and technologies.

Cyber-physical as a term explains much of the underlying theory and practice; it is the interplay of physics and computation. Our understanding of the physical world through the models of classical and quantum physics, together with our models of computation from analog to digital, helps us build a better understanding of the cyber-physical world. Insights from physics, methods of complex systems theory, and formal methods borrowed from various facets of mathematical and computational sciences will help us to build reliable, safe, and secure systems.

The advantages of digital computation are relatively low power requirements and flexibility; their carefully built discrete combinational states offer abstractions of Boolean logic. However such abstractions are limited, and we need to discover the conditions under which a digital abstraction of a system or subsystem can be valid. Physical properties appear continuous and constrain how cyber-physical systems models can be constructed and analyzed using formal logic. The study of the cyber-physical system also necessitates the study of the topologies of complex systems in addition to the computational and physical properties of their components.

A cyber-physical system is a complex set of systems and subsystems requiring communication channels among the cooperative entities and tasks, for example, a coordinated platoon of interconnected vehicles or a countrywide power system of different generating and consuming plants. Overall stability of the system will be affected by adversary attacks that tamper with the temporal characteristics, causing the delays of signals from nodes to nodes. Understanding which channels are less robust and furthermore what kind of network topologies have more resilience will help us minimize the number and the overall effect of compromised channels.

We created the Cyber-Physical Security Workshop and brought together researchers, engineers, and teachers into a common forum of exchange in order to achieve several goals, the primary one being to expose researchers, educators, and students to the world of CPS research. Invited speakers from academia and government labs offered glimpses of their work on modeling, analyzing, and understanding cyber-physical systems.

Sandro Bartolini,	University of Siena
Alexandre Chapoutot,	ENSTA ParisTech
Hervé Debar,	Télécom SudParis
Georgios Fainekos,	Arizona State University
Jennifer Hasler,	Georgia Institute of Technology
Israel Koren,	University of Massachusetts
Jackson Mayo,	Sandia National Laboratories
Fabio Pasqualetti,	University of California Riverside
Elaine Raybourn,	Sandia National Laboratories
John D. Siirola,	Sandia National Laboratories
Sam Green,	University of California, Santa Barbara

The Workshop was initiated and organized by the steering committee, composed of Çetin Kaya Koç (İstinye University, Nanjing University of Aeronautics and Astronautics, and UC Santa Barbara), Patrick Duvaut (Télécom ParisTech), David Naccache (École normale supérieure), and Jennifer Troup (Sandia National Laboratories).

The primary sponsor of the Workshop was the National Science Foundation with the award number 1638470 and the title “Cyber Physical Systems Security Education Workshop” with the Principal Investigator as Çetin Kaya Koç. The sponsors included Almerys, École normale supérieure, and Sandia National Laboratories.

The Cyber-Physical Security Workshop was held on July 17–19, 2017, on the campus of Télécom ParisTech, located in central Paris, in the heart of a rich urban and cultural environment. My thanks are also due to the faculty and students at Télécom ParisTech for running the Workshop.

I also sincerely thank Ronan Nugent of Springer for his valuable advice and the Editorial Office of Springer for their help in getting the book published.

Paris, France Çetin Kaya Koç

Contents

Contributors

Robert C. Armstrong Sandia National Laboratories, Livermore, CA, USA

Thierno Barry Univ Grenoble Alpes, CEA, List, Grenoble, France

Sandro Bartolini Department of Information Engineering and Mathematical Sciences, Università degli Studi di Siena, Siena, Italy

Nicolas Belleville Univ Grenoble Alpes, CEA, List, Grenoble, France

Gianluca Bianchin Department of Mechanical Engineering, University of California at Riverside, Riverside, CA, USA

Alexandre Chapoutot ENSTA ParisTech, Université Paris-Saclay, Palaiseau, France

Henri-Pierre Charles Univ Grenoble Alpes, CEA, List, Grenoble, France

Simon Cogliani École normale supérieure, Paris, France

Damien Couroussé Univ Grenoble Alpes, CEA, List, Grenoble, France

Jean-Luc Danger Télécom ParisTech, Paris, France

Secure-IC S.A.S., Cesson-Sévigné, France

Julien Alexandre dit Sandretto ENSTA ParisTech, Université Paris-Saclay, Palaiseau, France

Rodrigo Portella do Canto École normale supérieure, Paris, France

M. Abdelaziz Elaabid PSA-GROUPE, Paris, France

Peter Feghali University of California Santa Barbara, Santa Barbara, CA, USA

Bao Feng Huawei Technologies Co. Ltd., Guangdong, China

Houda Ferradi École normale supérieure, Paris, France

David Fritz Sandia National Laboratories, Albuquerque, NM, USA

Rémi Géraud École normale supérieure, Paris, France

Sam Green University of California Santa Barbara, Santa Barbara, CA, USA

Chongyan Gu Centre for Secure Information Technologies (CSIT), Institute of Electronics, Communications & Information Technology (ECIT), Queen's University Belfast (QUB), Belfast, UK

Sylvain Guilley Télécom ParisTech, Paris, France

Secure-IC S.A.S., Cesson-Sévigné, France

École normale supérieure, Paris, France

Jennifer Hasler Georgia Institute of Technology, Atlanta, GA, USA

Annelie Heuser Inria/IRISA Rennes-Bretagne Atlantique, Campus universitaire de Beaulieu, Rennes, France

Karine Heydemann Université Pierre et Marie Curie, Laboratoire d'informatique de Paris 6, LIP6, Paris, France

Geoffrey C. Hulette Sandia National Laboratories, Livermore, CA, USA

Khaled Karray Télécom ParisTech, Paris, France

Çetin Kaya Koç İstinye University, İstanbul, Turkey

Nanjing University of Aeronautics and Astronautics, Nanjing, China

University of California Santa Barbara, Santa Barbara, CA, USA

Michael Kunz Sandia National Laboratories, Albuquerque, NM, USA

Axel Legay Inria/IRISA Rennes-Bretagne Atlantique, Rennes, France

George Legrady University of California Santa Barbara, Santa Barbara, CA, USA

Weiqiang Liu College of Electronic and Information Engineering, Nanjing University of Aeronautics and Astronautics (NUAA), Nanjing, Jiangsu, China

Jieliang Luo University of California Santa Barbara, Santa Barbara, CA, USA

Diana Maimut Département d'informatique de l'ENS, École normale supérieure, CNRS, INRIA, PSL Research University, Paris, France

Advanced Technologies Institute, Bucharest, Romania

Jackson R. Mayo Sandia National Laboratories, Livermore, CA, USA

Tang Ming Wuhan University, Wuhan, China

Olivier Mullier ENSTA ParisTech, Université Paris-Saclay, Palaiseau, France

David Naccache École normale supérieure, Paris, France

Máire O'Neill Centre for Secure Information Technologies (CSIT), Institute of Electronics, Communications & Information Technology (ECIT), Queen's University Belfast (QUB), Belfast, UK

Vlad-Raul Paşca Faculty of Automatic Control and Computers, University Politehnica of Bucharest, Bucharest, Romania

Fabio Pasqualetti University of California, Riverside, CA, USA

Biagio Peccerillo Department of Information Engineering and Mathematical Sciences, Università degli Studi di Siena, Siena, Italy

Gang Qu Department of Electrical and Computer Engineering, University of Maryland, College Park, MD, USA

Elaine M. Raybourn Sandia National Laboratories, Albuquerque, NM, USA

Bruno Robisson CEA/EMSE, Secure Architectures and Systems Laboratory CMP, Gardanne, France

Maher Salloum Sandia National Laboratories, Livermore, CA, USA

Abderrahmane Seriai Univ Grenoble Alpes, CEA, List, Grenoble, France

Emil Simion Faculty of Automatic Control and Computers, University Politehnica of Bucharest, Bucharest, Romania

Andrew M. Smith Sandia National Laboratories, Livermore, CA, USA

Ray Trechter Sandia National Laboratories, Albuquerque, NM, USA

Vince Urias Sandia National Laboratories, Albuquerque, NM, USA

Craig M. Vineyard Sandia National Laboratories, Albuquerque, NM, USA

Guilin Wang Huawei Technologies Co. Ltd., Guangdong, China

Robust Digital Computation in the Physical World

Jackson R. Mayo, Robert C. Armstrong, Geoffrey C. Hulette, Maher Salloum, and Andrew M. Smith

Abstract Modern digital hardware and software designs are increasingly complex but are themselves only idealizations of a real system that is instantiated in, and interacts with, an analog physical environment. Insights from physics, formal methods, and complex systems theory can aid in extending reliability and security measures from pure digital computation (itself a challenging problem) to the broader cyber-physical and out-of-nominal arena. Example applications to design and analysis of high-consequence controllers and extreme-scale scientific computing illustrate the interplay of physics and computation. In particular, we discuss the limitations of digital models in an analog world, the modeling and verification of out-of-nominal logic, and the resilience of computational physics simulation. A common theme is that robustness to failures and attacks is fostered by cyber-physical system designs that are constrained to possess inherent stability or smoothness. This chapter contains excerpts from previous publications by the authors.

1 Introduction

Digital systems are ubiquitous due to the power and flexibility they offer in processing information. A digital system is ultimately a physical system (software must run on some kind of hardware); typically but not necessarily, this takes the form of silicon-based microelectronics. The power of such systems arises because they are carefully designed to have discrete combinatorial states that permit the abstraction of Boolean logic. The Boolean states are attractors for the underlying continuous physics (e.g., high- and low-voltage states). Under typical conditions, transitions between these states enable complex computation that is deterministic and lossless, a remarkable engineering achievement. Digital systems

J. R. Mayo (✉) · R. C. Armstrong · G. C. Hulette · M. Salloum · A. M. Smith
Sandia National Laboratories, Livermore, CA, USA
e-mail: jmayo@sandia.gov

Ç. K. Koç (ed.), *Cyber-Physical Systems Security*,
https://doi.org/10.1007/978-3-319-98935-8_1

can store and process substantial amounts of information because a large number of elements can change independently, creating vast combinations: N bits give 2^N states.

In the modern world, digital hardware and software are increasingly coupled with safety- and security-critical physical systems (industrial, military, medical, etc.) to form high-consequence cyber-physical systems. Engineers may use digital logic to control a physical system directly (embedded computing) and/or to perform simulation as a design aid (scientific computing). Two fundamental problems arise that pose a challenge to relying on digital systems in this way:

1. Turing's halting problem and Rice's theorem [27] indicate that no algorithm exists to predict a priori the behavior of a *generic* information processing system; i.e., such a system is undecidable even if deterministic. In particular, testing a digital system cannot establish bounds on all its possible behaviors, whereas quantifying safety and security requires such bounds. The undecidability result is based on idealizations, but even for finite digital systems, such questions are NP-hard [10]; there is no general practical technique to assure safety and security when the state space is exponentially large and exhaustive testing is infeasible.
2. Because digital systems are also physical, the assumed Boolean abstraction applies only under suitable or "nominal" environmental conditions. Rare physical events, which may become more common in extreme environments, can take the system behavior outside this intended space—e.g., causing bit flips. If the potential consequences of out-of-nominal behavior are of concern, and its likelihood is non-negligible due to either extreme environments (embedded computing) or extreme scale (scientific computing), then additional analysis is needed.

Solutions to enable assurance of digital and cyber-physical systems must constrain system designs to be analyzable in ways that generic systems are not. Ideally, analyzability and robustness are consciously designed-in along with functionality. One approach with wide adoption in industry is *formal methods* [35], which takes advantage of reduced complexity to enable rigorous automated reasoning about all possible behaviors within a model (effectively allowing special instances of undecidable or NP-hard problems to be solved tractably). Another, commonly used for naturally occurring (e.g., biological and social) systems but only beginning to impact cyber-physical engineering, is *complex systems theory* [19], which takes advantage of structured complexity to understand system robustness probabilistically. These approaches can overlap and provide complementary insights.

In the remainder of this chapter, we present several theoretical perspectives and simple examples illustrating techniques to mitigate unexpected behavior in cyber-physical systems. While the discreteness of digital logic is in some ways a radical departure from the smoothness of physics, the latter provides important understanding to help make digital computation more intrinsically robust as well.

2 Limitations of Digital Models in an Analog World

A fundamental question is under what conditions a digital abstraction of a system or subsystem can be valid, given that the system is governed by continuous physics. The successful operation of electronic computers indicates that such digital modeling abstractions are empirically meaningful. However, inherent properties of physics place constraints on how cyber-physical—or hybrid—models can be constructed and analyzed with formal logic.

In the case study presented here, causality and continuity properties of physics require what can appear as complications in reasoning, but in the end contribute to formally assuring a physically meaningful safety property for a digital thermostat. Physical continuity leads to the possibility of "indecision" in digital controllers, meaning that a response intended to be chosen from two discrete options may instead lie in between [21]. The thermostat example illustrates how indecision can be tolerated provided that, when both options ("heat on" and "heat off") are acceptable, an intermediate outcome is also acceptable.

The remainder of Sect. 2 is excerpted from [15],

2.1 Introduction

Much research and development work has targeted enabling formal verification of hybrid systems, typically in the form of model checking for so-called hybrid automata [2, 13]. We argue that this existing work is in different ways too broad and too narrow: Modeling approaches that freely combine discrete and continuous dynamics can readily introduce ill-posed and unphysical behavior due to the delicate interaction between the two types of dynamics [11]. And reasoning about hybrid systems via model checking is limited to properties that can be verified conservatively by enumeration of discrete regions within the continuous state space; even approaches using theorem proving have implemented model-checking strategies [34] or have relied on restrictive logics to formally model hybrid systems [25]. Work exists on formally analyzing continuous differential equations via theorem proving, but without modeling a coupling to digital logic [30]. We propose an approach that can leverage the full power of higher-order logic in the Coq theorem prover [4] to reason about physically consistent hybrid digital-physical models. Unlike model-checking approaches, our goal is not to completely automate the verification, but rather to provide maximum power and scalability for reasoning rigorously about properties of interest, leveraging understanding of system design for both the digital and physical elements.

2.2 Physics of Hybrid Modeling

Causality means that the value of a physical variable depends only on its beginning state and what happens to it subsequently, i.e., an event in the future, cannot affect any variable in the present or past, and any system failing to satisfy this constraint is considered unphysical. Computationally, causality manifests as a requirement that for recursively defined programs describing causal systems, the recursion must always be *well-founded*, i.e., must eventually terminate for any input.

In a formal analysis, it is in effect necessary to show that the time evolution computation terminates or has a solution. This is physically ensured by the causality property. That is, the time evolution operator actually depends only on events that occur between the initial time and the final time. In Sect. 2.5, we will show how this is reflected in a proof of termination within the Coq theorem prover—as opposed to the self-referential inconsistency of a non-causal time evolution operator that depends on events occurring in the future.

Formal verification is often undertaken in order to identify rare but potentially catastrophic corner-case behaviors. *Buridan's Principle* [3, 21] describes an often-overlooked issue in cyber-physical modeling, where discrete decisions about continuous variables are often required, in spite of the fact that such decisions cannot be guaranteed to complete in bounded time. Buridan's Principle manifests in the system as the decision potentially taking an arbitrarily long time to complete or, equivalently, remaining incomplete (with an intermediate, non-digital result) if it is examined after a fixed time. Many cyber-physical analyses digitize the physics prior to analysis, an approach that is convenient but fails to preserve the fundamental continuity properties that can lead to unexpected indecision in the real system—exactly the sort of corner-case behavior that formal verification seeks to uncover.

It is important to note that Buridan's Principle does not conflict with the physical propagation of discrete states by actual computers. *Given* a discontinuous set of initial states (e.g., voltages representing 0 or 1), an appropriate continuous nonlinear electrical circuit can implement logic perfectly by computing resulting discrete states at subsequent clock cycles [21]. Thus purely digital models, and traditional formal analyses thereof, are valid and valuable for a computational component that is set up in this way, but are not sufficient for understanding cyber-physical system behavior comprehensively including continuous inputs and outputs. The latter consideration calls for understanding, e.g., potential non-digital behaviors from indecision in a nominally digital device—either pragmatically bounding them in probability or, as here, incorporating them as far as possible in a consistent model for exhaustive formal analysis.

2.3 Definition of the Thermostat Model

In physical terms, this model describes an idealized, thermally homogeneous object that gains heat from time to time via a rapid heat pulse (idealized as instantaneous)

from a transducer and loses heat to the environment via a linear cooling law. The transducer is designed to maintain the object's temperature T in a desired range above the ambient temperature by, at uniform time intervals, measuring T and applying a heat pulse if T is below a threshold.

For convenience, we take the unit of time to be the interval between sensor measurements and take the zero point of the temperature scale to be the ambient temperature. Our model has four *positive* real parameters: the cooling coefficient α, the temperature rise H due to a heat pulse, the nominal threshold temperature T_*, and the temperature margin ε for indecision. The constraint $T_* > \varepsilon$ is imposed. An additional parameter is an "arbiter" function $\tilde{\theta}\colon \mathbb{R} \to \mathbb{R}$ that approximates the unit step function but allows for indecision rather than requiring an unrealistic discontinuity. For all $\Delta \in \mathbb{R}$, the arbiter must satisfy

$$\tilde{\theta}(\Delta) \in [0, 1], \tag{1}$$

$$\Delta > \varepsilon \implies \tilde{\theta}(\Delta) = 1, \tag{2}$$

$$\Delta < -\varepsilon \implies \tilde{\theta}(\Delta) = 0. \tag{3}$$

The behavior of the physical system is described by the temperature as a function of time, $T\colon \mathbb{R}_{\geq 0} \to \mathbb{R}$, assuming that the system starts running at time $t = 0$. Instead of the traditional differential-equation formulation, we use an integral equation that corresponds to the time evolution operator $\mathcal{U}$. Moreover, following standard techniques in physics, we exploit the *linearity* of the thermal dynamics to express $\mathcal{U}$ via *superposition* in terms of "micro" propagators. The latter propagators represent individual, linearly combining contributions to the solution, and in general include a *kernel* that propagates initial or boundary conditions and a *Green's function* that propagates external forcing events.

In our thermal case, the kernel and the Green's function reduce to the same propagator:

$$G(t, t') = \mathrm{e}^{-\alpha(t-t')}\,\theta(t - t'). \tag{4}$$

The (exact) unit step function θ here is not an arbiter but a means for continuous-time dynamics to enforce causality—that an effect at time t cannot precede its cause at time t'.

The time evolution operator for our thermal system, then, advances the state from t_j to t_i by linearly superposing the effects of the initial condition $T(t_j)$ and the subsequent heating events:

$$\begin{aligned} T(t_i) &= T(t_j)\,G(t_i, t_j) + \int_{t_j}^{\infty} \mathrm{d}t'\, q(t')\,G(t_i, t') \\ &= T(t_j)\,\mathrm{e}^{-\alpha(t_i - t_j)} + \int_{t_j}^{t_i} \mathrm{d}t'\, q(t')\,\mathrm{e}^{-\alpha(t_i - t')} \quad \text{for } t_i > t_j \geq 0. \end{aligned} \tag{5}$$

Here, the part of the integral for $t' > t_i$ vanishes due to the causality-enforcing step function in G; and the function q is the thermal forcing term, which in our case is a sum over heat pulses.

We now introduce the feedback from the transducer, which makes the fully coupled cyber-physical system *nonlinear*. Namely,

$$q(t) = \sum_{l=0}^{\infty} H\,\tilde{\theta}\big(T_* - T(l)\big)\,\delta(t-l). \tag{6}$$

The control logic design seeks to provide a heat pulse of magnitude H if and only if the current temperature is below T_*. Buridan's Principle requires the use here of a continuous arbiter function $\tilde{\theta}$ rather than the exact step function θ, since the transducer cannot be guaranteed to provide a discrete response in bounded time.

2.4 Informal Analysis of the Thermostat Model

A key characterization of the performance of this cyber-physical system is its ability to maintain the temperature in a desired range above the (zero) ambient temperature. Thus, we wish to prove the following as a theorem for some particular bounds $0 < A < B < \infty$:

$$\text{If } T(0) \in [A, B], \text{ then } T(t) \in [A, B] \text{ forall } t \in \mathbb{R}_{\geq 0}. \tag{7}$$

The theorem (7) can be derived straightforwardly from the following lemma:

$$\text{For all } n \in \mathbb{N}, \text{ if } T(n) \in [A, B], \text{ then } T(t) \in [A, B] \text{ for all } t \in (n, n+1]. \tag{8}$$

The derivation is as follows: By first specializing $t = n + 1$ in the lemma, and recalling the hypothesis that $T(0) \in [A, B]$, we obtain by induction that $T(n) \in [A, B]$ for all $n \in \mathbb{N}$. Now, whereas for $t = 0$, the theorem (7) is trivially valid, for $t > 0$ we specialize $n = \lceil t \rceil - 1$ in the lemma and obtain the required result. This establishes the temperature bounds for all $t \in \mathbb{R}_{\geq 0}$.

We now argue that the lemma (8) holds for any A and B that satisfy

$$0 < A \leq \min\left(\frac{H}{\mathrm{e}^{\alpha} - 1},\ (T_* - \varepsilon)\mathrm{e}^{-\alpha}\right) \text{ and } B \geq T_* + \varepsilon + H. \tag{9}$$

Under the constraints of our model, this means that suitable bounds can be found with $0 < A < B < \infty$.

As a starting point, from the governing equations (5) and (6), if we assume $t \in (n, n+1]$ and substitute $\{t_i, t_j\} = \{t, n\}$, we compute

$$T(t) = \Big(T(n) + H\,\tilde{\theta}\big(T_* - T(n)\big)\Big)\,\mathrm{e}^{-\alpha(t-n)}. \tag{10}$$

We are given that $T(n) \in [A, B]$. We note that $t - n \leq 1$ and thus $e^{-\alpha(t-n)} \geq e^{-\alpha}$.

The proof of the lemma (8) is now by case analysis.

2.5 Formal Implementation

We have formalized our analysis within the Coq interactive theorem prover [4], which allows us to precisely define the various terms of our model and then state and prove theorems about those terms. To model continuous variables, we use the `Reals` module provided as part of Coq's standard library [33].

For our analysis, the arbiter function $\tilde{\theta}$ need not be defined explicitly but must have essential properties asserted corresponding to Eqs. (1)–(3):

```
Parameter eps : R.
Hypothesis eps_pos : 0 < eps.

Parameter theta_tilde : R → R.
Hypothesis theta_tilde_bound : ∀ d, 0 ≤ theta_tilde d ≤ 1.
Hypothesis theta_tilde_1 : ∀ d, d > eps → theta_tilde d = 1.
Hypothesis theta_tilde_0 : ∀ d, d < -eps → theta_tilde d = 0.
```

In accordance with Buridan's Principle, this formulation avoids the need to compare exact real numbers, sidestepping the associated undecidability problem while retaining enough structure to support our analysis.

As part of the construction, an assertion that an event at a future time cannot contribute to the computation of the temperature at the current time must be made. To the theorem prover, this requirement manifests itself as a termination condition and reflects the fact that a non-causal function that depends on both the future and the past is self-referential and inconsistent in the general case. Computationally, this is understood as an obligation to demonstrate that the recursively defined function is well-founded.

We have not yet completed the proof that this computational definition corresponds to the original integrated temperature equation (5). The proof is straightforward in principle but depends upon an extension to Coq's `Reals` standard library of theorems, which is the subject of ongoing work [5].

Formalizing the proof of lemma (8) in Sect. 2.4 is straightforward. Here we present the interesting parts of the development, eliding the more tedious details. First, we define Eq. (10) as *T*, using *theta_tilde* from above, along with the other relevant parameters:

```
Definition T Tn tau (tau_bnd : 0 < tau ≤ 1) :=
  (Tn + H × theta_tilde (Tstar - Tn)) × exp (-a×tau).
```

The definition takes three parameters. The first, *Tn*, is the temperature at time *n* where $n \in \mathbb{N}$. The parameter *tau* represents the time increment relative to *n* at which we want to evaluate the temperature. The final parameter, *tau_bnd*, is a proof that *tau* lies in the interval (0, 1]. The definition above corresponds to Eq. (10), with

$t = n+tau$. Using this definition, the statement and proof of lemma (8) are expressed as follows:

```
Theorem T_in_interval (Tn tau : R) (tau_bnd : 0 < tau ≤ 1) :
  A ≤ Tn ≤ B → A ≤ T Tn tau tau_bnd ≤ B.

Proof.
  intros HAB. decompose record HAB. split.
  destruct (Rlt_le_dec Tn (Tstar - eps)).
    apply Tn_heat_keeps_above; auto.
    apply Tn_no_heat_keeps_above; auto.
  destruct (Rle_lt_dec Tn (Tstar + eps)).
    apply Tn_heat_keeps_below; auto.
    apply Tn_no_heat_keeps_below; auto.
Qed.
```

3 Modeling and Verification of Out-of-Nominal Logic

Apart from intended interactions with physical surroundings, digital devices may also be subject to extreme environments that violate their nominal operating parameters and result in computational behavior outside the designed logic. In microelectronics, out-of-nominal insults such as heating or radiation can alter electrical properties and lead to unexpected states (upsets). More generally, untested, out-of-range, or malicious inputs may cause digital systems to respond in unintended ways.

Ongoing work seeks to understand the physical origins of out-of-nominal behavior (see Fig. 1) as well as the principles underlying the susceptibility or robustness of digital systems to cascading failure from such upsets. By including digital upset effects in a model, the consequences for the rest of the digital state space can be quantified and mitigated—e.g., to verify whether a digital safety property still holds even in an accident scenario. We have argued that both formal methods and complex systems theory offer ways to design more intrinsically robust digital systems [22]. Design using formal methods imposes constraints that promote stability similar to that seen in adapted complex systems.

The remainder of Sect. 3 is excerpted from [23], © 2016 Springer.

3.1 Introduction

A widely used technique to improve the tractability of formal verification is to work with abstractions (or overapproximations), which can be simpler to analyze and are conservative in the sense that their verified safety properties are guaranteed

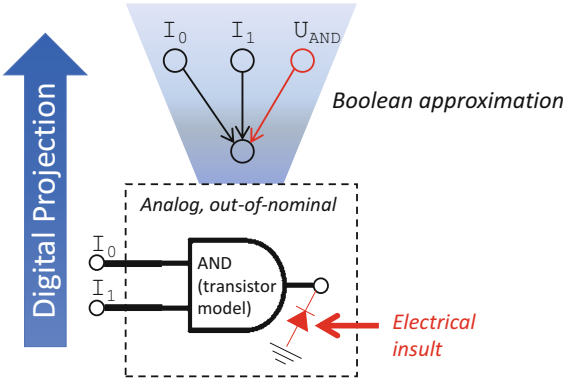


Fig. 1 Conceptual diagram showing the translation of simulated analog electrical behavior (bottom) into a digital upset model (top). The analog model represents a transistor-based AND gate that may be subject to an insult changing its output. The digital representation enlarges on a nominal Boolean AND gate with an additional nondeterministic input U_{AND} that induces the corresponding upset state

to hold also in the actual implementation. This guarantee applies because a valid abstraction permits all behaviors that occur in the implementation and possibly additional behaviors. In current formal methods, the abstraction is a means to an end: either generating a proof of an existing design or generating a provable design.

Here we present a different perspective on abstraction—useful when, under some conditions, a system is physically capable of additional behaviors beyond its "nominal" operation. Critical safety properties may need to be guaranteed under a less restrictive model that permits particular "out-of-nominal" behaviors, if such behaviors may physically occur often enough to be of concern for the risk of catastrophic failure. Our observation is that the abstraction concept, already commonly used in formal methods as a mathematical technique, can be reinterpreted as defining a space of possible "real-world" out-of-nominal behaviors for which the abstraction-verified safety properties are still guaranteed to hold. Thus, by leveraging suitable abstractions, we can gain out-of-nominal safety verification for free.

A primary example of out-of-nominal behavior is the response of digital hardware to electrical or other physical stimuli that produce states not accounted for in the logic design—with the abnormal physical dynamics generating a nominally disallowed digital state transition such as a bit flip. A variety of formal techniques have been investigated for modeling and verifying such behavior [12, 17, 18]; recognizing that out-of-nominal behavior may overlap with other formal abstractions can increase the applicability of these techniques, particularly in earlier stages of the design process.

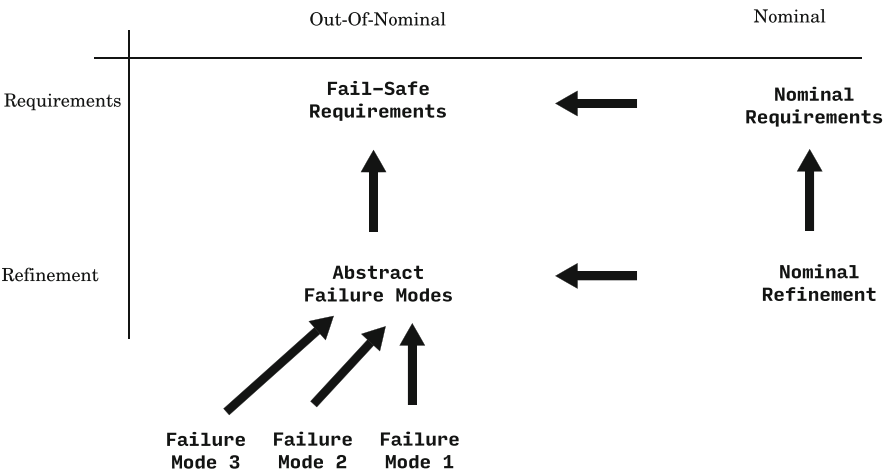


Fig. 2 Refinement/abstraction conceptual diagram for treating out-of-nominal and nominal models in a unified way. The arrows point in the direction of abstraction. Figure from [23], © 2016 Springer

3.2 Modeling Out-of-Nominal Safety Properties

The safety properties of a given model are required to hold at all times over all possible behavioral paths. Such properties, when imposed on an abstraction, require that every path in the abstraction conforms to the properties, and thus every refinement will as well. The use of abstraction in verifying safety requirements is well established.

Here we distinguish "critical" safety requirements that must hold even in out-of-nominal environments (Fig. 2). These out-of-nominal fail-safe requirements are less strict (allow more behaviors) than the requirements for nominal operation and thus constitute an abstraction of the nominal requirements. Safety-critical devices where failure modes can be anticipated are likely candidates for this technique. Nominal requirements can be relaxed to admit acceptable modes of failure.

The safety requirements must ultimately be verified on formal models that reflect the actual nominal and out-of-nominal behavior of the system being designed. Such models are typically tied to the requirements via one or more abstraction/refinement steps ultimately leading to a model of a practical implementation. In our approach, upon refinement, the out-of-nominal model remains an abstraction of the nominal one (Fig. 2). By stipulating that the out-of-nominal refinement has a superset of the behaviors of the nominal refinement, we ensure that the safety properties verified for out-of-nominal operation also hold for nominal operation.

Not all foreseeable failure modes may manifest an abstraction or overapproximation of the system's nominal behavior. A particular failure mode may render the system incapable of performing some nominal behaviors. The removal of possible

behavioral paths, by itself, does not invalidate any of the nominal safety properties, but can affect functional requirements that are outside the scope of the formal refinement methodology applied in this work. Out-of-nominal scenarios of concern for safety would involve adding at least some new behaviors.

Viewing behaviors of anticipated malfunctions as an abstraction of the nominal behavior has some advantages. For complex safety-critical systems that are prone to failure, it is important to "design-in" anticipated failures with their own fail-safe requirements. Recasting such requirements into the familiar abstraction/refinement design practice means that the same tools can be brought to bear on these designed-in benign failure requirements as part of the normal design process. Another advantage is that anticipated failure modes are incorporated into the design process up front rather than as an afterthought.

3.3 Example Turnstile Model

For an illustration, we use the familiar turnstile model [16] in a simplified form. A turnstile requires a coin to permit the patron admission by pushing on the bar. In a simplified description, we can identify three Boolean state variables for the device: C, P, and L, indicating whether a coin is present, whether the bar is being pushed, and whether the bar is locked. If the coin is present and the bar is locked, the bar should become unlocked and remain so until the patron pushes through, after which it should become locked again. If the coin is absent, the bar should remain locked. We can synthesize the desired nominal properties into a TLA+ [20] formula:

$$\begin{aligned} S1 &\triangleq (\neg C \wedge L \Rightarrow L') \ldots\ldots \textbf{ critical safety property} \\ S2 &\triangleq (C \wedge L \Rightarrow \neg L') \\ S3 &\triangleq (\neg P \wedge \neg L \Rightarrow \neg L') \\ S4 &\triangleq (P \wedge \neg L \Rightarrow L') \\ \mathit{Safety} &\triangleq \Box[S1 \wedge S2 \wedge S3 \wedge S4]_{\langle C,P,L\rangle}. \end{aligned} \tag{11}$$

Here, each Sn defines a safety property in terms of a TLA action, which relates the variables C, P, and L in the "current" instant to L', representing the value of L in the "next" instant.

While all of the implications in (11) can be thought of as safety properties, the "critical safety property" $S1$ is one that we wish to preserve in a design for anticipated out-of-nominal conditions. We can interpret $S1$ as "the turnstile will remain locked unless a coin is present" $(\neg C \wedge L \Rightarrow L')$.

The nominal requirements in (11) can be used as an abstraction suitable for refinement. If the refinement is valid, all of $S1$ through $S4$ will be true of the implementation. One initial refinement of the requirements is described by the action

$$L' = (\neg C \wedge L) \vee (P \wedge \neg L). \tag{12}$$

3.4 Design and Out-of-Nominal Verification Via Abstraction

3.4.1 Refinement (High Level)

We now consider a method by which abstraction and refinement can be used in a formal design process in order to account for out-of-nominal conditions. The process starts, as any design process should, with the requirements. These are gathered in the usual ways and must be formalized. These are the nominal requirements.

Next, certain of these requirements are designated as "critical"—these are the out-of-nominal requirements, i.e., those that must hold even under some (predicted) mode of system failure or inconsistency.

Next, we refine the nominal requirements. The refined model is closer to an implementation, although it may still be quite abstract. Refinement of the nominal model is done in the usual way [1, 20], ensuring that the level above simulates the level below.

Finally, we must construct the out-of-nominal refinement such that it both *refines* the out-of-nominal requirements *and abstracts* the nominal refinement, completing the commuting square diagram (shown for the turnstile example in Fig. 3). In this case, our out-of-nominal requirement is only that $\neg C \wedge L \Rightarrow L'$. In the nominal refinement, L evolves based on the action

$$L' = (\neg C \wedge L) \vee (P \wedge \neg L).$$

Since the first disjunct alone already satisfies the out-of-nominal requirement that $\neg C \wedge L \Rightarrow L'$, we can consider the second disjunct to behave "randomly" and, at any step, draw its value from either the nominal behavior $P \wedge \neg L$ or its negation

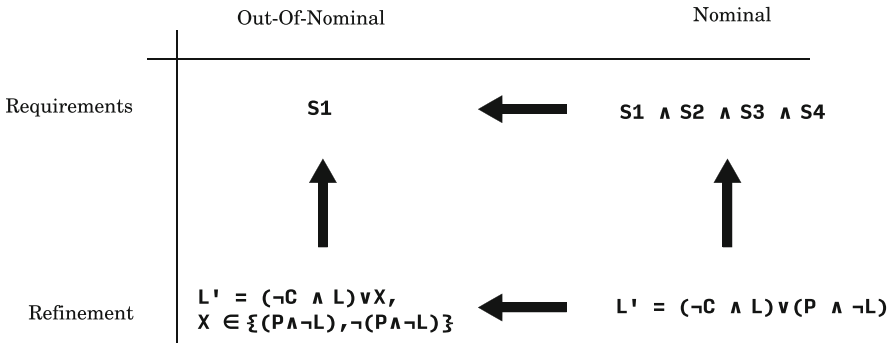


Fig. 3 Refinement/abstraction diagram for the turnstile example. The arrows point in the direction of abstraction. Existing formal abstractions can be reinterpreted in this framework; a technique like CEGAR [9] might already prove that the nominal design (lower right) satisfies a safety property (upper left) by finding an abstraction (lower left) that satisfies the safety property. Figure from [23],

$\neg(P \wedge \neg L)$. In the model, we denote by X a value from this set, and the out-of-nominal refinement is derived by replacing the action above with

$$L' = (\neg C \wedge L) \vee X.$$

By contrast, if we had used the logically equivalent nominal refinement

$$L' = (\neg C \vee \neg L) \wedge (P \vee L),$$

it would not have been straightforward to obtain an out-of-nominal abstraction preserving the critical safety requirement $S1$. That is, while the disjunctive and conjunctive normal forms are of course equivalent in their nominal behavior, in this example one particular choice of design offers the ability to tolerate a faulty out-of-nominal operation. This interpretation gives abstraction an even more central role in driving the design process.

3.4.2 Implementation (Low Level)

We now discuss how the refined logic design for the turnstile (on both the out-of-nominal and nominal sides) can be related to a notional implementation in hardware gates. The out-of-nominal logic $L' = (\neg C \wedge L) \vee X$ implies that the $P \wedge \neg L$ term can be computed by an unreliable gate, but the remaining gates must remain reliable even under out-of-nominal conditions. We discuss an intrinsically robust implementation using Boolean networks (BNs) informed by principles of digital error damping.

We draw on previous work [22] in which example BNs were constructed to compute a half-adder function and their robustness was analyzed with the NuSMV [8] model checker. We ignore the "sum" output and use only the "carry" output, which corresponds directly to an AND operation. Conventionally, a BN is interpreted as a sequential logic circuit. To implement combinational logic, we replicate the gates in "tiers," with each tier providing its results as input to the next and with the final output being read at the end of a specified number of tiers (here, 20).

Two BNs were constructed, differing in the design parameter k, the average number of inputs per node [22]. In accordance with complex systems analysis [19], the BN with $k = 1.5$ shows "quiescent" behavior (perturbations are damped), and the BN with $k = 2.5$ shows "chaotic" behavior (perturbations are amplified). Typical real-world digital implementations are found empirically to be chaotic [24]; such implementations are cheaper to create because they impose fewer restrictions on programmability. Quiescent implementations that damp bit-flip errors are more constrained and generally more difficult to create. Our strategy here is to use the cheaper chaotic implementation for parts of the design that do not impact the critical safety property and to use the more expensive quiescent implementation for parts that need robustness to preserve the critical safety property.

Each of the two AND operations in $L' = (\neg C \wedge L) \vee (P \wedge \neg L)$ can be implemented with either of the BNs as far as *nominal* behavior is concerned. This is verified by exhaustive testing as well as model checking with NuSMV [22] and is as expected because the BNs were chosen to compute their function correctly when operating with their nominal logic.

For out-of-nominal behavior, as before [22], we consider the possibility of any single bit flip (incorrect gate output) within some range of tiers in the BN, again using a nondeterministic formal model of the kind used in other work on soft errors [31]. We have adapted the NuSMV analysis in this case to check the correctness of the carry bit specifically. In these BNs, because bit flips occurring at or shortly before the output stage may not have a chance to self-correct, the bit flip is restricted to the first $n_{\max}$ tiers, where we consider $1 \leq n_{\max} \leq 20$. The NuSMV analysis finds that for no such value of $n_{\max}$ does the chaotic BN reliably implement the AND operation, while the quiescent BN does so for any $n_{\max} \leq 15$. Thus, if we can arrange that the effect of the out-of-nominal environment is not felt in the last 5 tiers, then the quiescent BN can be used to implement the "critical" term $\neg C \wedge L$. Meanwhile, either BN (or for that matter, any nominally correct implementation) can be used for $P \wedge \neg L$ because the out-of-nominal side imposes no constraint on this term.

4 Resilience of Computational Physics Simulation

Large-scale high-performance computing (HPC) is a powerful tool commonly used for digital simulation of physical systems, known as scientific computing. The challenge of representing the continuous equations of physics in discrete logic is addressed by the field of numerical analysis, with much effort devoted to ensuring that results are stable in the face of inevitable roundoff and truncation errors. This stability is closely tied to that of the physics being simulated; the goal is to ensure that the discretization does not introduce gratuitous instability.

At the same time, the HPC platform is itself a physical entity and is subject to out-of-nominal failures from temperature, cosmic rays, etc. that may halt or corrupt computations—known as the resilience problem. The sheer scale of HPC (hundreds of thousands of processors) increases the rate of hardware errors even when running a correct program in a well-controlled physical environment. Thus, HPC is subject to out-of-nominal behavior analogous to that of embedded devices that are smaller but exposed to harsher conditions. Both scientific computing and embedded computing are often power-constrained, driving hardware designs to the "edge" of reliability. Here we discuss how the stability of HPC algorithms can be extended, taking advantage of the structure and smoothness of the physics being *simulated*, to help achieve robustness to physical anomalies in the computer itself.

The remainder of Sect. 4 is excerpted from [29],

4.1 Introduction

The increasing scale of HPC will eventually cause previously negligible hardware errors to become important. Some of these errors will likely arise from unanticipated sources and modes, including silent (undetected) errors [7]. Attempting to hide such faults through enhanced hardware error correction may make computing platforms slower and/or more expensive. Thus, future hardware may regularly expose silent errors that can corrupt application results.

A primary concern is silent data corruption (SDC), which leads to wrong numerical values in a computation that otherwise appears normal.

Algorithm-based fault tolerance [6] (ABFT) is a promising strategy that seeks to efficiently mitigate the effects of hardware errors, including SDC, at the software level for particular kinds of applications. The effectiveness of ABFT is dependent on the characteristics of the application (such as numerical stability properties) and on the assumed hardware error model.

We specifically target physics simulation applications that solve partial differential equations (PDEs). We also initially assume an SDC model consisting of dynamic random-access memory (DRAM) bit flips. This chapter builds on previous "robust stencils" for explicit finite-difference solvers [26, 32] and describes the more general use of robust versions of vector and sparse-matrix operations representing discretized physical space, which are typical ingredients of PDE solvers.

Our approach seeks algorithm-level *stability* to isolated occurrences of SDC, via detection and recovery that ideally occur in cache as a low-cost add-on to the numerical computation. Here we describe preliminary implementations and evaluations of this in situ approach, demonstrating its feasibility and showing the potential for extension to physics solvers of increasing scale and realism.

4.2 Methodology

4.2.1 Operations in PDE Solvers

Physical systems are often modeled by partial differential equations (PDEs) which, when discretized, result in a system of equations with a large yet finite number of unknowns [28]. Algorithms to solve for these variables can typically be expressed in terms of linear systems of equations $Au = b$ with a large sparse matrix A.

Our team developed SDC-tolerant algorithms for the explicit finite-difference solution of linear and nonlinear advection equations [26]. The algorithm developed for the parallel solution of a 1D advection problem can be viewed as computing a relevant sparse product Au using a matrix-free scheme. On this basis, we have generalized the robust stencil technique to a broader class of PDE solvers via robust linear algebra operations, which we now describe.

4.2.2 Error Model and Mitigation Approach

The hardware SDC model we have considered to date consists of independent bit flips in DRAM occurring uniformly in space and time (Poisson process), parameterized by the error probability (rate) per bit with respect to wall time.

We seek to develop a technique to correct this type of SDC when performing linear algebra operations in PDE solvers. The correction method is based on numerical interpolation to replace suspected corrupt points with values computed from neighboring points.

A key advantage of this approach is that, since isolated errors are mitigated throughout the algorithm, a catastrophic breakdown in accuracy should occur only when two or more errors occur in close proximity, such that one contaminates the correction of the other. This resilience property is analogous to that of (expensive) triple modular redundancy.

Consider a data vector $f(x)$ that is a smooth field in the physical domain where it is defined. An operation on f such as *daxpy* involves a loop on points x_i. During the loop, if any value $f(x_i)$ is detected as outlier, it is interpolated based on values at spatially neighboring points as illustrated in Fig. 4. A value $f(x_i)$ is deemed an outlier if it is NaN or Inf or if its deviation from a set of neighboring points exceeds a threshold (chosen empirically and measured relative to the variation in the neighboring points).

4.3 Application to the Conjugate Gradient Solver

The building blocks of robust linear algebra operations we develop are applicable to a wide variety of iterative solvers. Here we report the performance of the building blocks on a conjugate gradient (CG) algorithm for a linear system of equations, used in solving an elliptic PDE.

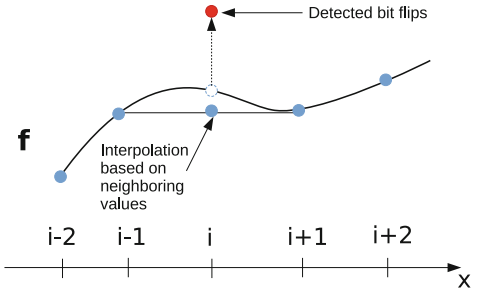


Fig. 4 Schematic of a data vector $f(x)$ with a corrupted entry (red point). If this corruption is detected as an outlier, it is interpolated based on neighboring values. This is performed as the linear algebra computation is sweeping the vector. Figure from [29], © 2016 ACM

4.3.1 The Basic Conjugate Gradient Solver

The basic CG solver is given in Algorithm 1 [28]. It is used when A is symmetric and positive definite, which is the case in elliptic PDE problems. At each iteration, CG involves one *spmv* call, three *daxpy* calls, and two *ddot* calls.

Algorithm 1 Conjugate gradient method for a linear system of equations $Au = b$

$u_0 = 0$ {Initial guess of the solution}
$r_0 = b - Au_0$, $p_0 = r_0$ {Initial residual and direction vectors}
$(R_0)^2 = r_0^T r_0$
for $k = 0$ until convergence **do**
  $q_k = Ap_k$
  $\alpha = (R_k)^2 / \left(p_k^T q_k\right)$
  $u_{k+1} = u_k + \alpha p_k$
  $r_{k+1} = r_k - \alpha q_k$
  $(R_{k+1})^2 = r_{k+1}^T r_{k+1}$
  $\beta = (R_{k+1})^2 / (R_k)^2$
  $p_{k+1} = r_{k+1} + \beta p_k$
end for
Return u_{k+1}

When run in parallel, CG requires communication between the compute nodes during the *ddot* and *spmv* steps. The CG method is more communication-intensive than an explicit scheme, especially at large scale, due to the presence of collective operations.

Our target solver is the HPCCG 1.0 mini-app [14], which uses the CG method to solve an elliptic PDE in a 3D domain of size $N_x \times N_y \times PN_z$, where P is the number of processes. This entails solving a linear system of equations $Au = b$, where A is a sparse heptadiagonal matrix of size $\left(PN_xN_yN_z\right)^2$ equally distributed among the P processes. We run weak-scaling computations such that each process's portion of the vector u is of fixed size $N_xN_yN_z$. The full solution vector is formed by stacking the domains of each process in the z-direction.

4.3.2 Controlling Convergence of the CG Algorithm

The CG algorithm relies on two scalars α and β that govern the solution steps toward convergence. With these coefficients, the method achieves a rapid convergence rate on a reliable machine, but we find that it can become unstable when using our interpolation-based robust linear algebra operations to mitigate SDC throughout the computation. We propose a method to control the CG step size: Define a "tuning parameter" $\theta \leq 1$ as a scaling factor on α. If $\theta < 1$, the step size is decreased and the CG convergence rate is decreased. Since α and β are related, we apply a corresponding scaling to drive β toward 1 as α is reduced. We therefore propose

in Algorithm 1, after computing α and β, the additional steps $\alpha \leftarrow \theta\alpha$ and $\beta \leftarrow 1 - \theta(1 - \beta)$.

We compute θ as a linear interpolation between a smaller value θ_2 at the large initial residual R_0 and a larger value θ_1 when the residual reaches a small value R_c. The CG algorithm with an automatically tuned parameter θ converges two to three times faster than with a constant tuning parameter.

4.3.3 Evaluation Using In Situ Interpolation

In contrast to rollback techniques, the roll-forward error mitigation that we describe in this chapter helps reduce the cost of false positives. This mitigation technique has been implemented for the HPCCG 1.0 mini-app. We inject bit flips in all u, p, q, and r vectors involved in the CG method (see Algorithm 1). We report weak-scaling computations performed with the robust operations and the variable tuning parameter technique described in Sect. 4.3.2. We compare to the standard HPCCG implementation without use of robust linear algebra operations or the tuning parameter.

The HPCCG elliptic PDE is solved with $64 \times 64 \times 4$ degrees of freedom per core, and the solver is required to reach an assumed acceptable accuracy (residual of 5×10^{-2}). A linear interpolation scheme is used to mitigate the corrupted data points. The weak-scaling results are plotted in Fig. 5. The correction strategy

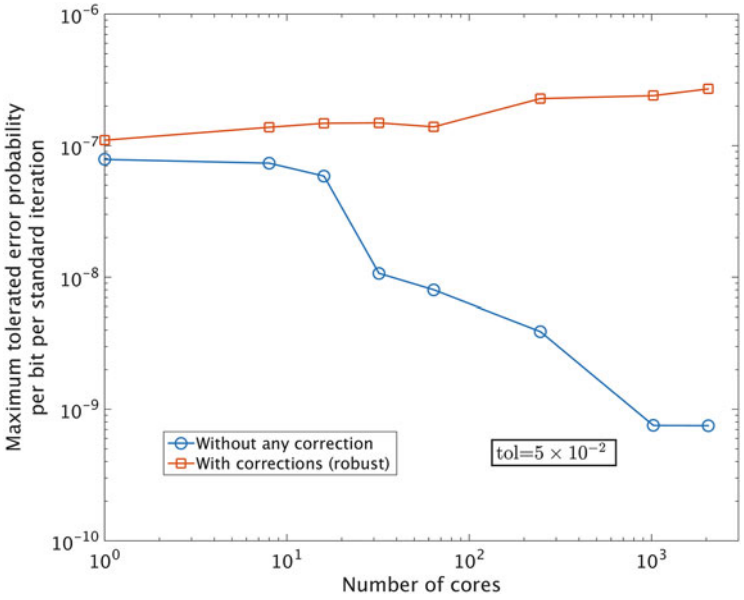


Fig. 5 Maximum tolerated error probability for the CG solver with and without interpolation. Figure from [29], © 2016 ACM

enables increased tolerance of silent errors depending on the number of cores P, i.e., problem size. As shown in Fig. 5, the maximum tolerated error rate is modestly higher than that of the standard solver for smaller problems with $P \leq 16$. However, the correction approach enables tolerating an error rate up to two orders of magnitude higher than a standard solver for larger problems. The robust solver converges in approximately the same number of iterations as the standard solver. The wall time shows decreasing overhead at larger scale from the robust solver operations—down to ∼5% at 2048 cores—because the wall time becomes dominated by the communication cost of each iteration.

5 Conclusion

Analysis of traditional physical systems relies heavily on continuity and stability properties, which allow behaviors not yet observed to be predicted: A small change in input produces a small change in output. Though digital systems are ultimately physical, they are designed to operate on combinatorial spaces that are effectively discontinuous. As a result, for digital systems that are created arbitrarily, the response to untested inputs or perturbations cannot be usefully predicted or bounded. Even when the digital system operates as an idealized logical machine, there is much room for design flaws and vulnerabilities to hide in its complexity.

From a broader systems engineering view, where digital logic is constructed from physics and is used to control or simulate other physics, we encounter the additional complications of cyber-physical systems and out-of-nominal behavior. This represents both a challenge and an opportunity to leverage understanding of the physics to confer analogous stability on the digital logic and the cyber-physical system as a whole. Both formal methods and complex systems theory offer techniques for creating hardware and software with designed-in robustness and analyzability. These insights can help improve the safety and security of high-consequence cyber-physical systems by making them more predictable and resilient.

Acknowledgements Sandia National Laboratories is a multimission laboratory managed and operated by National Technology and Engineering Solutions of Sandia LLC, a wholly owned subsidiary of Honeywell International Inc., for the US Department of Energy's National Nuclear Security Administration (NNSA) under contract DE-NA0003525. This work was funded by NNSA's Advanced Simulation and Computing (ASC) Program.

References

1. J.R. Abrial, *Modeling in Event-B: System and Software Engineering*, 1st edn. (Cambridge University Press, Cambridge, 2010)
2. R. Alur, C. Courcoubetis, T.A. Henzinger, P.H. Ho, Hybrid automata: an algorithmic approach to the specification and verification of hybrid systems, in *Hybrid Systems*, ed. by

R.L. Grossman, A. Nerode, A.P. Ravn, H. Rischel. Lecture Notes in Computer Science, vol. 736 (Springer, Heidelberg, 1993), pp. 209–229
3. J.C. Barros, B.W. Johnson, Equivalence of the arbiter, the synchronizer, the latch, and the inertial delay. IEEE Trans. Comput. **32**(7), 603–614 (1983)
4. Y. Bertot, P. Castéran, *Interactive Theorem Proving and Program Development* (Springer, Heidelberg, 2004)
5. S. Boldo, C. Lelay, G. Melquiond, Improving real analysis in Coq: a user-friendly approach to integrals and derivatives, in *CPP'12*, ed. by C. Hawblitzel, D. Miller. Lecture Notes in Computer Science, vol. 7679 (Springer, Heidelberg, 2012), pp. 289–304
6. G. Bosilca, R. Delmas, J. Dongarra, J. Langou, Algorithm-based fault tolerance applied to high performance computing. J. Parallel Distrib. Comput. **69**, 410–416 (2009)
7. F. Cappello, A. Geist, W. Gropp, S. Kale, B. Kramer, M. Snir, Toward exascale resilience: 2014 update. Supercomput. Front. Innov. **1**(1), 5–28 (2014)
8. A. Cimatti, E.M. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani, A. Tacchella, NuSMV 2: an opensource tool for symbolic model checking, in *Proceedings of the 14th International Conference on Computer Aided Verification* (2002), pp. 359–364
9. E. Clarke, O. Grumberg, S. Jha, Y. Lu, H. Veith, Counterexample-guided abstraction refinement for symbolic model checking. J. ACM **50**, 752–794 (2003)
10. S.A. Cook, The complexity of theorem-proving procedures, in *Proceedings of the 3rd Annual ACM Symposium on Theory of Computing* (1971), pp. 151–158
11. P. Derler, E.A. Lee, A. Sangiovanni-Vincentelli, Modeling cyber-physical systems. Proc. IEEE (special issue on CPS) **100**(1), 13–28 (2012)
12. G. Fey, Assessing system vulnerability using formal verification techniques, in *Mathematical and Engineering Methods in Computer Science*, ed. by Z. Kotásek, J. Bouda, I. Černá, L. Sekanina, T. Vojnar, D. Antoš. Lecture Notes in Computer Science, vol. 7119 (Springer, Heidelberg, 2012), pp. 47–56
13. T.A. Henzinger, The theory of hybrid automata, in *Proceedings of the 11th Annual IEEE Symposium on Logic in Computer Science* (1996), pp. 278–292
14. M.A. Heroux, D.W. Doerfler, P.S. Crozier, J.M. Willenbring, C. Edwards, A. Williams, M. Rajan, E.R. Keiter, H.K. Thornquist, R.W. Numrich, Improving performance via mini-applications. Report SAND2009-5574, Sandia National Laboratories (2009)
15. G.C. Hulette, R.C. Armstrong, J.R. Mayo, J.R. Ruthruff, Theorem-proving analysis of digital control logic interacting with continuous dynamics. Electron. Notes Theor. Comput. Sci. **317**, 71–83 (2015)
16. M. Jackson, P. Zave, Deriving specifications from requirements: an example, in *Proceedings of the 17th International Conference on Software Engineering* (1995), pp. 15–24
17. A. Joshi, S.P. Miller, M. Whalen, M.P. Heimdahl, A proposal for model-based safety analysis, in *Proceedings of the 24th Digital Avionics Systems Conference* (2005)
18. A. Joshi, M.P.E. Heimdahl, S.P. Miller, M.W. Whalen, Model-based safety analysis. NASA Contractor Report CR-2006-213953 (2006)
19. S.A. Kauffman, *The Origins of Order: Self-organization and Selection in Evolution* (Oxford University Press, Oxford, 1993)
20. L. Lamport, *Specifying Systems: The TLA+ Language and Tools for Hardware and Software Engineers* (Addison-Wesley, Boston, 2002)
21. L. Lamport, Buridan's principle. Found. Phys. **42**(8), 1056–1066 (2012)
22. J.R. Mayo, R.C. Armstrong, G.C. Hulette, Digital system robustness via design constraints: the lesson of formal methods, in *Proceedings of the 9th Annual IEEE International Systems Conference* (2015), pp. 109–114
23. J.R. Mayo, R.C. Armstrong, G.C. Hulette, Leveraging abstraction to establish out-of-nominal safety properties, in *Proceedings of the 4th International Workshop on Formal Techniques for Safety-Critical Systems*, ed. by C. Artho, P.C. Ölveczky. Communications in Computer and Information Science, vol. 596 (Springer, Heidelberg, 2016), pp. 172–186
24. T. Mytkowicz, A. Diwan, E. Bradley, Computer systems are dynamical systems. Chaos **19**, 033124 (2009)

25. A. Platzer, J.D. Quesel, KeYmaera: a hybrid theorem prover for hybrid systems, in *Automated Reasoning*, ed. by A. Armando, P. Baumgartner, G. Dowek. Lecture Notes in Computer Science, vol. 5195 (Springer, Heidelberg, 2008), pp. 171–178
26. J. Ray, J.R. Mayo, R.C. Armstrong, Finite difference stencils robust to silent data corruption, in *SIAM Conference on Parallel Processing for Scientific Computing* (2014). https://www.pathlms.com/siam/courses/477/sections/716
27. H.G. Rice, Classes of recursively enumerable sets and their decision problems. Trans. Am. Math. Soc. **74**(2), 358–366 (1953)
28. Y. Saad, *Iterative Methods for Sparse Linear Systems*, 2nd edn. (SIAM, Philadelphia, 2003)
29. M. Salloum, J.R. Mayo, R.C. Armstrong, In-situ mitigation of silent data corruption in PDE solvers, in *Proceedings of the 6th Workshop on Fault Tolerance for HPC at Extreme Scale* (2016)
30. M.U. Sanwal, O. Hasan, Formal verification of cyber-physical systems: coping with continuous elements, in *ICCSA'13*, ed. by B. Murgante, S. Misra, M. Carlini, C.M. Torre, H.Q. Nguyen, D. Taniar, B.O. Apduhan, O. Gervasi. Lecture Notes in Computer Science, vol. 7971 (Springer, Heidelberg, 2013), pp. 358–371
31. S.A. Seshia, W. Li, S. Mitra, Verification-guided soft error resilience, in *Proceedings of the Conference on Design, Automation and Test in Europe* (2007), pp. 1442–1447
32. P. Strazdins, B. Harding, C. Lee, J.R. Mayo, J. Ray, R.C. Armstrong, A robust technique to make a 2D advection solver tolerant to soft faults, in *Proceedings of the International Conference on Computational Science* (2016)
33. The Coq development team, The Coq proof assistant reference manual (2004). http://coq.inria.fr
34. O. Tveretina, Towards the safety verification of real-time systems with the Coq proof assistant, in *Proceedings of the International Multiconference on Computer Science and Information Technology*, vol. 2 (2007), pp. 883–892
35. J. Woodcock, P.G. Larsen, J. Bicarregui, J. Fitzgerald, Formal methods: practice and experience. ACM Comput. Surv. **41**, 19 (2009)

Constraint-Based Framework for Reasoning with Differential Equations

Julien Alexandre dit Sandretto, Alexandre Chapoutot, and Olivier Mullier

Abstract An extension of constraint satisfaction problems with differential equations is proposed. Reasoning with differential equations is mandatory to analyze or verify dynamical systems, such as cyber-physical ones. A constraint-based framework is presented to model a wider class of problems based on logical combination of high-level properties. In addition, the complete correctness is verified using a set-membership approach in this framework. Finally, examples are given to demonstrate the benefits of the presented framework.

1 Introduction

In various domains, such as robotic, control theory, or biology, mathematical models based on differential equations are used to represent the temporal behavior of a particular system. Among the many classes of differential equations, this chapter is interested in *ordinary differential equations* (ODEs) and *differential algebraic equations* (DAEs) which are widely used in these domains. These models are then used for different purposes such as parameter identification, control synthesis, or safety verification. For example, interesting problems involving differential equations are:

- *Control synthesis problem*: A motion planning algorithm for a mobile robot R aims at finding a trajectory of R going from point $\mathbf{a}$ to a point $\mathbf{b}$ of the state space while avoiding an obstacle $\mathbf{o}$. Note that as the movement of R depends on actuators (e.g., engine speed), finding a trajectory is translated into finding control inputs such that R can reach $\mathbf{b}$.
- *Parameters identification problem*: Mathematical models are an approximation of the real word. To make them more faithful, usually an identification step is necessary. Starting from a list of n measures $\mathbf{m}_{i\in\{1,\dots,n\}}$ of the behavior of the real

J. Alexandre dit Sandretto (✉) · A. Chapoutot · O. Mullier
ENSTA ParisTech, Université Paris-Saclay, Palaiseau Cedex, France
e-mail: alexandre@ensta.fr; chapoutot@ensta.fr; mullier@ensta.fr

Ç. K. Koç (ed.), *Cyber-Physical Systems Security*,
https://doi.org/10.1007/978-3-319-98935-8_2

system S and a parametric mathematical model $M(\mathbf{p})$ of the temporal behaviors of S, the goal of the identification step is to find values $\hat{\mathbf{p}}$ of $\mathbf{p}$ such that $M(\hat{\mathbf{p}})$ fits the list of measures $\mathbf{m}_{i\in\{1,\dots,n\}}$.
- *Design problems*: It is closed to the parametric identification problem. The input data are a parametric model $M(\mathbf{p})$ of a system S and a list of n design specifications $P_{i\in\{1,\dots,n\}}$, for example, a car shall reach a speed of 100km/h from 0km/h in less than 6 s on flat dry road. The goal of the design problem is to find values $\hat{\mathbf{p}}$ of $\mathbf{p}$ such that $M(\hat{\mathbf{p}})$ respects all the specifications $P_{i\in\{1,\dots,n\}}$.

Defining automatic methods to solve such kind of problems is challenging. In this context, many attempts have been started based on constraint verification of differential systems [6, 13, 18, 22]. Indeed, the framework of constraint satisfaction problems is an appealing one to express such kinds of problems associated with efficient solving methods producing rigorous results.

We mainly focus on critical problems, coming from aeronautics, robotics, or medical fields. Handling problems in these fields implies to consider the uncertainties in presence using validated methods like interval analysis [24, 28]. We also impose that constraints to solve have to be properly verified. In this context, we use inner approximation to ensure the constraint satisfaction because an enclosure (outer approximation) approach would result in some points that are not solution [24, 28]. The classical approach for these requirements is the use of a Branch-and-Prune algorithm which is a dedicated solver for constraint satisfaction problems (CSP) and the most used in the case of numerical or continuous CSP [31, 32].

To handle differential equations, abstraction based on validated simulation or reachability is a common approach [10, 28–30]. This abstraction needs to be deeply studied to preserve the correctness of a constraint-based problem-solving.

In this chapter, we expand a framework for Constraint Satisfaction Differential Problems (CSDP) with the requirement of preserving the guarantee of the result while dealing with differential constraints. The main contributions are:

- A clearer definition of a CSP framework based on set-membership operations including differential equations ODEs or DAEs compared to previous work [6, 13, 18], namely, SCSDP for *Set-Based Constraint Satisfaction Differential Problems*. In particular, a better handling of quantified constraints is presented, and a better separation between mathematical model and solving algorithm is defined.
- A sound solving algorithm of SCSDP based on interval analysis and guaranteed integration methods is presented as well as a complete study of the impact of the representation of sets by boxes on the guarantee of the solution of SCSDP.
- A set of contractor operators on the solution of differential equations is defined to make Branch-and-Contract solver more efficient for SCSDP.

The chapter is organized as follows. In Sect. 2, the basics of numerical constraint satisfaction problem are provided. The mathematical formulation of CSCP is defined in Sect. 3, while solving algorithm is presented in Sect. 4. Some examples are given in Sect. 5 before concluding in Sect. 6.

Notations
Small italic letters x represent real variables, while real vectors $\mathbf{x}$ are in bold. The set of real numbers is denoted by $\mathbb{R}$. Intervals $[x]$ and interval vectors (boxes) $[\mathbf{x}]$ are represented between brackets. We denote by $\mathbb{IR}$ the set of closed intervals over $\mathbb{R}$. Sets $\mathcal{S}$ are in uppercase calligraphic. The powerset of a set $\mathcal{X}$ is denoted by $\wp(\mathcal{X})$. The derivative of a function x with respect to time t is denoted by $\dot{x}$. Uppercase typewriter letters stand for algorithmic data structures such as a stack `S` or a queue `Q`.

2 Preliminary Notions

A brief presentation of the *constraint satisfaction problem* framework is given in Sect. 2.1 to better understand how it is then extended in the rest of the chapter. A generic solving method based on Branch-and-Prune algorithm is presented in Sect. 2.2

2.1 Numerical Constraint Satisfaction Problems

In this section, we recall the numerical constraint satisfaction problem (NCSP) formalism, following the description given in [32], and present some basics on constraint programming. The approach of NSCP is both powerful to address complex problems (NP-hard problem with numerical issues, even in critical applications) and simple in the definition of a solving framework [1, 26].

A NCSP $(\mathcal{V}, \mathcal{D}, \mathcal{C})$ is defined as follows:

- $\mathcal{V} := \{v_1, \ldots, v_n\}$ is a finite set of variables which can also be represented by the vector $\mathbf{v}$.
- $\mathcal{D} := \{[v_1], \ldots, [v_n]\}$ is a set of intervals such that $[v_i]$ contains all possible values of v_i. It can be represented by a box $[\mathbf{v}]$ gathering all $[v_i]$.
- $\mathcal{C} := \{c_1, \ldots, c_m\}$ is a set of constraints of the form $c_i(\mathbf{v}) \equiv \mathbf{g}_i(\mathbf{v}) = 0$ or $c_i(\mathbf{v}) \equiv \mathbf{g}_i(\mathbf{v}) \leqslant 0$, with nonlinear $\mathbf{g}_i : \mathbb{R}^n \to \mathbb{R}$ for $1 \leqslant i \leqslant m$. Constraints $\mathcal{C}$ are interpreted as a conjunction of equalities and inequalities, i.e., $\mathcal{C} \equiv c_1 \wedge c_2 \wedge \cdots \wedge c_m$.

The solution of a NCSP is a valuation of $\mathbf{v}$ ranging in $[\mathbf{v}]$ and satisfying the constraints $\mathcal{C}$.

2.2 Branch-and-Contract Solving Method

The classical algorithm to solve a NCSP, as previously defined, is the *Branch-and-Prune* method which needs only an interval evaluation of the constraints and an

initial domain for variables. More precisely, an interval $[\underline{x}, \overline{x}] = \{x \in \mathbb{R} : \underline{x} \leqslant x \leqslant \overline{x}\} \in \mathbb{IR}$ is defined by its lower and upper bounds $\underline{x}$ and $\overline{x}$ as a compact set of $\mathbb{R}$. An interval vector (or *box*) $[\mathbf{x}]$ of dimension n is a Cartesian product of intervals $[\underline{x}_0, \overline{x}_0] \times \cdots \times [\underline{x}_n, \overline{x}_n]$. An inclusion function $[f] : \mathbb{IR}^n \to \mathbb{IR}$ for $f : \mathbb{R}^n \to \mathbb{R}$ satisfies

$$\forall [\mathbf{x}] \in \mathbb{IR}\{f(\mathbf{x}) | \mathbf{x} \in [\mathbf{x}]\} \subseteq [f]([\mathbf{x}]). \tag{1}$$

A *natural inclusion* function $[f]$ is obtained by substituting all variables and operations involved in f by their interval counterpart. The evaluation of the range of functions over intervals using inclusion function leads in general to some overapproximation; see [24] for more details.

A more elaborated solving method named *Branch-and-Contract* is usually applied to accelerate the solving process of an NCSP. A generic version, using interval analysis, of this algorithm is given in Algorithm 1.

Algorithm 1 A generic Branch-and-Contract

Require: $\mathbf{v}$, $[\mathbf{v}]$, $\mathscr{C}_{\text{acc}}$, $\mathscr{C}_{\text{rej}}$
Ensure: $\mathbb{S}_{\text{acc}}$, $\mathbb{S}_{\text{rej}}$, $\mathbb{S}_{\text{unc}}$
1: $\mathbb{S} = \{[\mathbf{v}]\}$
2: $\mathbb{S}_{\text{acc}} = \emptyset$, $\mathbb{S}_{\text{rej}} = \emptyset$, $\mathbb{S}_{\text{unc}} = \emptyset$
3: **while** $\mathbb{S} \neq \emptyset$ **do**
4: Pop a $[\mathbf{v}]_{\text{current}}$ from $\mathbb{S}$
5: $[\mathbf{v}]_{\text{current}} = \text{Contract}\ (\mathscr{C}_{\text{acc}}, ([\mathbf{v}]_{\text{current}}))$ ▷ May be omitted to get branch-and-prune method
6: **if** Check($\mathbf{v}$,$[\mathbf{v}]_{\text{current}}$,$\mathscr{C}_{\text{acc}}$) **then** ▷ Satisfiability check
7: Push $[\mathbf{v}]_{\text{current}}$ in $\mathbb{S}_{\text{acc}}$
8: **else if** Check($\mathbf{v}$,$[\mathbf{v}]_{\text{current}}$,$\mathscr{C}_{\text{rej}}$) **then** ▷ Unsatisfiability check
9: Push $[\mathbf{v}]_{\text{current}}$ in $\mathbb{S}_{\text{rej}}$
10: **else if** Width($[\mathbf{v}]_{\text{current}}$) $> \varepsilon$ **then**
11: $([\mathbf{v}]_{\text{left}}, [\mathbf{v}]_{\text{right}}) = \text{Bisect}([\mathbf{v}]_{\text{current}})$ ▷ Spliting method
12: Push $[\mathbf{v}]_{\text{left}}$ in $\mathbb{S}$
13: Push $[\mathbf{v}]_{\text{right}}$ in $\mathbb{S}$
14: **else**
15: Push $[\mathbf{v}]_{\text{current}}$ in $\mathbb{S}_{\text{unc}}$
16: **end if**
17: **end while**

The key feature of Algorithm 1 is the function Check(),

$$\text{Check}() : \mathscr{V} \times \mathscr{D} \times \mathscr{C} \to \mathbb{B}$$

with $\mathbb{B} = \{\text{True}, \text{False}\}$. Check() is then a *decision procedure* which is able to verify if constraints $\mathscr{C}$ are satisfied by all the values of the domain $\mathscr{D}$. Note that due to pessimism of interval analysis approach, it may be not possible to decide if $\mathscr{C}$ is satisfied or not. In this case, False has to be interpreted to "undecidable."

Two kinds of constraints are considered in Algorithm 1, one with constraints $\mathscr{C}_{\text{acc}}$ and one with constraints $\mathscr{C}_{\text{rej}}$. The first set of constraints is used to accept the solutions, while the second is used to reject the nonsolutions, i.e., the points guaranteed to be outside the solution domain. It is optional but useful to speed up the algorithm by quickly eliminating unfeasible domain. $\mathscr{C}_{\text{rej}}$ is in a certain point of view the negation of $\mathscr{C}_{\text{acc}}$, but in general, $\mathscr{C}_{\text{acc}} \neq \neg\mathscr{C}_{\text{rej}}$, due to the abstraction of continuous domains (and the issue of disjunction). Note that this approach follows classical method in SIVIA (Set Inversion Via Interval Analysis) method [23].

The second key feature in *Branch-and-Contract* algorithm is the Contract() procedure which simultaneously reduces the domain studied ($[\mathbf{v}]_{\text{current}}$ in the algorithm) by the help of contractors [3]. A contractor associated to a constraint $c \equiv g(\mathbf{x}) \diamond 0$ with $\diamond \in \{=, \leqslant\}$ is a function C_c taking a box $[\mathbf{x}]$ as parameter and returns a box such that

$$C_c([\mathbf{x}]) \subseteq [\mathbf{x}] \qquad \text{(Reduction)} \qquad (2a)$$

$$g([\mathbf{x}]) \cap [z] = g(C_c([\mathbf{x}])) \cap [z] \qquad \text{(Correction)} \qquad (2b)$$

where $[z] = [0, 0]$ if $\diamond \equiv =$ and $[z] = [-\infty, 0]$ if $\diamond \equiv \leqslant$. The main strength of contractors is that they can reduce the domain $[\mathbf{x}]$ while preserving solution without using bisection, and so they can reduce, in practice, the algorithmic complexity of Algorithm 1. For more details on contractors, see [3].

The result of Algorithm 1, also known as *a paving*, is made of three lists of boxes $\mathbb{S}_{\text{acc}}$, $\mathbb{S}_{\text{rej}}$, and $\mathbb{S}_{\text{unc}}$ such that

- There is no solution of the NCSP in $\mathbb{S}_{\text{rej}}$.
- All the solutions of the NCSP, included in the initial domain, are in $\mathbb{S}_{\text{acc}} \cup \mathbb{S}_{\text{unc}}$.
- All the values in $\mathbb{S}_{\text{acc}}$ are solution of the NCSP.

Example 1 The result of a Branch-and-Contract method on the constraints $1 \leqslant x^2 + y^2 \leqslant 2$ with $(x, y) \in [-2, 2] \times [-2, 2]$ is given in Fig. 1. In this figure, blue boxes are elements of $\mathbb{S}_{\text{acc}}$, red boxes are in $\mathbb{S}_{\text{rej}}$, and white boxes (between red and blue boxes) are in $\mathbb{S}_{\text{unc}}$. ■

2.3 Some Limitations on NCSP

2.3.1 Equality Constraints

One of the main difficulties in NCSP approach is the handling of equality constraint, i.e., $\mathbf{g}(\mathbf{v}) = 0$. Indeed, a classical solving approach for NCSP is based on interval analysis which considers computations over boxes instead of points. So proving equality constraints usually involves some relaxation techniques such as proving a simpler constraint of the form $\mathbf{g}(\mathbf{v}) \in [-\varepsilon, \varepsilon]$ for a small positive value ε. Moreover specific algorithms have to be used to prove the existence and uniqueness of the

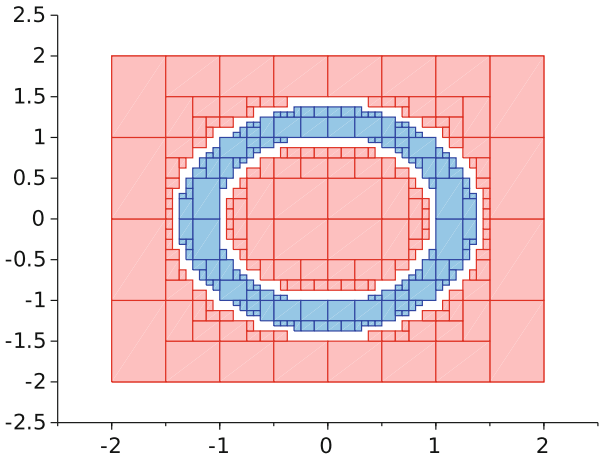


Fig. 1 Paving of a circle

solution of $\mathbf{g}(\mathbf{v}) = 0$ such as the interval Newton operator [24]. In consequence, solving equality constraints is often dependent on the solving algorithm as the relaxation is generally done internally in the solver. The aim of our work is to avoid this implementation trick and push out the relaxation choice to the designer by only allowing set-based constraints as inclusion constraint; see Sect. 3.

2.3.2 Differential Constraints

The framework of NCSP lacks expressiveness when dealing with differential equation. In [6], a first approach was given by introducing *Constraint Satisfaction Differential Problems* (CSDP). Basically, new variables are added to the set of variables of NCSP to represent time derivative, and a new type of constraints is added too to represent the dynamic of the differential system. The time variable being handled separately from the other variables, temporal properties, cannot be encoded with CSDP. For example, if a trajectory described by a differential system has to avoid an obstacle in a given time interval, modeling this using CSDP cannot be done in an obvious manner.

Another work in [18] also dealt with this problem. The dynamical system is abstracted with a solution operator ϕ representing the solution of the system. Differential equations are then naturally embedded in the NCSP framework. Its limitation is also this abstraction because constraints on the dynamical system cannot be easily expressed.

In these previous works, another drawback is the lack of quantification on variables. Bringing a solution to this is one of the motivations of this work.

3 Set-Based Constraint Satisfaction Differential Problems

The proposed extension of NSCP is based on set-based constraints and the embedding of differential constraints. These extensions allow to increase the class of problems which can be modeled and solved.

3.1 Dynamical Systems

In the rest of this article, a general class of differential equations is considered which can represent ordinary differential equations (ODEs), differential algebraic equations (DAEs) of index 1, and a mix of these equations with additional constraints, e.g., to model energy preservation. More precisely, differential systems of the form

$$\begin{cases} \dot{\mathbf{y}}(t) = \mathbf{f}(t, \mathbf{y}(t), \mathbf{x}(t), \mathbf{p}), \\ \quad 0 = \mathbf{g}(t, \mathbf{y}(t), \mathbf{x}(t)) \\ \quad 0 = \mathbf{h}(\mathbf{y}(t), \mathbf{x}(t)) \end{cases} . \tag{3}$$

with nonlinear functions $\mathbf{f} : \mathbb{R} \times \mathbb{R}^n \times \mathbb{R}^m \times \mathbb{R}^p \to \mathbb{R}^n$, $\mathbf{g} : \mathbb{R} \times \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^m$, $\mathbf{h} : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}$, $t \in [0, t_{\text{end}}]$, $\mathbf{y}(0) \in \mathscr{Y}_0$, and $\mathbf{p} \in \mathscr{P}$ are considered. More precisely, initial value problems (IVP) for parametrized differential equations are considered over a finite time horizon $[0, t_{\text{end}}]$. Note that a bounded set of initial values and a bounded set of parameters are considered in this framework. This implies to deal with set of trajectories solution of Eq. (3). We assume classical hypothesis on **f**, **q**, and **h** to ensure the existence and uniqueness of the solution of Eq. (3).

In the rest of this section, we denote by $\mathscr{Y}(\mathscr{T}, \mathscr{Y}_0, \mathscr{P})$ the set

$$\mathscr{Y}(\mathscr{T}, \mathscr{Y}_0, \mathscr{P}) = \{\mathbf{y}(t; \mathbf{y}_0, \mathbf{p}) : t \in \mathscr{T}, \mathbf{y}_0 \in \mathscr{Y}_0, \mathbf{p} \in \mathscr{P}\} . \tag{4}$$

Intuitively, $\mathscr{Y}(\mathscr{T}, \mathscr{Y}_0, \mathscr{P})$ gathers all the points reached by the solution $\mathbf{y}(t; \mathbf{y}_0, \mathbf{p})$ of Eq. (3) starting from all scalar initial values $\mathbf{y}_0$ and all scalar parameters **p**. Note that $\mathscr{Y}(\mathscr{T}, \mathscr{Y}_0, \mathscr{P})$ is hardly computable in general, and the implementation issue is addressed in Sect. 4. Note also that the difference of the proposed approach comparing to [18] is that we consider a set-based solution operator which offers a convenient way to deal with quantification over variables.

The purpose of the proposed framework is to check if $\mathscr{Y}(\mathscr{T}, \mathscr{Y}_0, \mathscr{P})$ fulfills some specification defined in terms of set-based constraints.

3.2 Set-Based Constraints

To avoid problematic issue due to equality constraints (see Sect. 2.3), set-based constraints are considered. More precisely, *inclusion* and *disjunction* operators are considered. The proposed framework will consider constraints of the form

$$\mathbf{g}(\mathscr{A}) \subseteq \mathscr{B} \tag{5a}$$

$$\mathbf{g}(\mathscr{A}) \supseteq \mathscr{B} \tag{5b}$$

$$\mathbf{g}(\mathscr{A}) \cap \mathscr{B} = \emptyset \tag{5c}$$

$$\mathbf{g}(\mathscr{A}) \cap \mathscr{B} \neq \emptyset \tag{5d}$$

where $\mathscr{A}$ and $\mathscr{B}$ are real compact sets and $\mathbf{g}$ is a nonlinear function. The lifting of $\mathbf{g}$ over sets is defined as usual by $\mathbf{g}(\mathscr{X}) = \{\mathbf{g}(x) : x \in \mathscr{X}\}$.

Note that these constraints can be seen as Boolean functions, but while, from a mathematical formulation, the truth value can always be obtained, it may not be the case when they have to be solved on a computer. The safe computer resolution of these kinds of constraints is one of the main contributions of this article, detailed in Sect. 4.

3.3 Set-Based Differential Constraint Satisfaction Problems

The handling of differential constraints here follows the approach given in [18] in the exception of the solution operator of Eq. (3), which is here represented as a set of solution $\mathscr{Y}(\mathscr{T}, \mathscr{Y}_0, \mathscr{P})$ in order to unify the objects manipulated into constraints which are also sets.

Set-Based Constraint Satisfaction Differential Problems (SCSDP) based on a set-membership constraints and embedding differential constraints can now be defined.

Definition 1 (SCSDP) A SCSDP is a NCSP made of

- A finite set $\mathscr{S}$ of differential systems S_i as defined in Eq. (3)
- A finite set of variables $\mathscr{V}$ including the parameters of the differential systems S_i, i.e., $(\mathbf{y}_0, \mathbf{p})$, a time variable t and some other algebraic variables $\mathbf{q}$
- A domain $\mathscr{D}$ made of the domain of parameters $\mathbf{p} : \mathscr{D}_p$, of initial values $\mathbf{y}_0 : \mathscr{D}_{y_0}$, of the time horizon $t : \mathscr{D}_t$, and the domains of algebraic variables $\mathscr{D}_q$
- A set of constraints $\mathscr{C}$ which may be defined by inclusion or disjunction constraints (see Sect. 3.2) over variables of $\mathscr{V}$ and special variables $\mathscr{Y}_i(\mathscr{D}_t, \mathscr{D}_{y_0}, \mathscr{D}_p)$ representing the set of the solution of S_i in $\mathscr{S}$

Example 2 A cruise control system based on PI controller for a nonlinear dynamic of a car is considered [12]. The dynamic of the car is defined by

$$S \equiv \begin{cases} \dot{v} = \dfrac{k_p(v_{\text{set}} - v) + k_i s - 50.0v - 0.4v^2}{m} \\ \dot{s} = v_{\text{set}} - v \end{cases} . \tag{6}$$

with v the speed of the vehicle and s the integral part of the PI controller, k_p and k_i the parameters of the PI controller, $m \in [990, 1010]$ the mass of the vehicle, and $v_{\text{set}} = 10$ the target speed of the car from initial conditions $v(0) = 0$ and $s(0) = 0$. The term $k_p(v_{\text{set}} - v) + k_i s$ is the PI controller, $-50.0v$ is the resisting force due to the road, and $-0.4v^2$ is the aerodynamic friction.

The specification of the PI controller is such that it should stabilize in 10 s with a tolerance of 2%, and its overshoot should not be more than 5% of the target speed. These are translated into constraints such that

$$v(10) \subseteq [9.8, 10.2] \qquad \text{(At t=10, } v_{\text{set}} \pm 2\%\text{)}$$

$$\dot{v}(10) \subseteq [-\varepsilon, \varepsilon] \qquad \text{(At t =10, acceleration is around zero, with a small } \varepsilon > 0\text{)}$$

$$v([0, 10]) \subseteq [0, 10.5] \qquad \text{(For } t \in [0, 10], v \text{ should not be above } v_{\text{set}} + 5\%\text{)}$$

Note that in Eq. (6), the mass m is uncertain, so the solution of S is a thick function so the use of inclusion constraints. In summary, a SCSDP is defined by

- $\mathscr{S} = \{S \text{ defined in Eq. (6)}\}$
- $\mathscr{V} = \{k_p, k_i\}$
- $\mathscr{D} = \{[1, 4000], [1, 120]\}$
- $\mathscr{C} = \{v(10) \subseteq [9.8, 10.2], \dot{v}(10) \subseteq [-\varepsilon, \varepsilon], v([0, 10]) \subseteq [0, 10.5]\}$

■

Note that following NCSP and its solving algorithm, variables in $\mathscr{V}$ are quantified existentially, and other variables (not in $\mathscr{V}$) are quantified universally, e.g., the mass m in Eq. (6). Hence, there is no need to introduce quantifier in constraints. The proposed SCSDP framework is hence simpler than previous work [6, 13, 18] in embedding quantification constraints while taking into account differential equations. Nevertheless, one important challenge is to solve SCSDP in a guaranteed way, and for this purpose a computable representation of sets has to be defined. As interval analysis [28] brings very efficient techniques over boxes, it is a natural mean to solve SCSDP on computers.

4 Solving SCSDP

To solve a SCSDP as defined in Sect. 3.3, a proper representation of sets has to be defined. Interval analysis provides a simple representation of compact sets by the mean of interval values or boxes. This representation is simple enough to represent complex sets while being associated with fast computational methods. To solve SCSDP on a computer, set-based constraints defined in Sect. 3.2 have to be properly translated into boxes, and dynamical systems as defined in Sect. 3.1 have to be solved.

4.1 Interval-Based Constraints

In order to solve set-based constraints appearing in SCSDP, as defined in Sect. 3.2, an interval-based abstraction is given in this section. As shown in Sect. 2.1, complex compact sets can be represented by paving and so can be represented either by inner approximation (boxes in $\mathbb{S}_{\text{acc}}$) or outer approximation (boxes in $\mathbb{S}_{\text{acc}} \cup \mathbb{S}_{\text{unc}}$). In consequence, translating constraints defined in Eq. (5) to interval-based constraints, a proper representation of sets has to be defined. In particular, the validity of the translation is important to guarantee the result of solving a SCSDP on a computer.

In the sequel, $\text{Int}\mathscr{X}$ will stand for the interior of the compact set $\mathscr{X}$, while $\text{Hull}\mathscr{X}$ will stand for the outer approximation of $\mathscr{X}$. In each case, the inner approximation or the outer approximation can be defined by a box or a list of boxes. Note that the outer approximation of a nonlinear function g in interval arithmetic is given by inclusion function as defined in Eq. (1), and more information can be found in [24], while inner approximation of g requires special treatments as defined in [16, 17, 19, 21]. Note also that excepting a complete computation of inner approximation or outer approximation, there is no meaning to consider outer approximation of g with inner approximation of its parameter and reciprocally.

Table 1 Set-based constraint evaluation in interval analysis framework

			$\mathscr{A}$	
			$\text{Int}\,\mathscr{A}$	$\text{Hull}\,\mathscr{A}$
$\mathbf{g}(\mathscr{X})$	$\text{Hull}\,\mathbf{g}(\text{Hull}\,\mathscr{X})$	$\subseteq$	`true`	?
		$\supseteq$	`false`	?
		$\cap_{=\emptyset}$	?	`true`
		$\cap_{\neq\emptyset}$	?	`false`
	$\text{Int}\,\mathbf{g}(\text{Int}\,\mathscr{X})$	$\subseteq$	?	`false`
		$\supseteq$	?	`true`
		$\cap_{=\emptyset}$	`false`	?
		$\cap_{\neq\emptyset}$	`true`	?

In Table 1, a summary of the translation of constraints defined in Eq. (5) into an interval counterpart is given. For each case, inner approximation or outer approximation, the truth value of the constraints is inspected. When a value is `true`, that is, the constraint can be proved to be true, and when a value is `false`, then the constraint can be proved to be false. Otherwise, no conclusion can be made on the constraint and it is denoted by "?." For example, a proof of a constraint of the form $g(\mathscr{X}) \subseteq \mathscr{A}$ can be obtained only by considering an outer approximation of g and its parameter $\mathscr{X}$ and an inner approximation of $\mathscr{A}$. As a second example, a proof of unsatisfiability of the constraints $g(\mathscr{X}) \subseteq \mathscr{A}$ can be obtained considering an inner approximation of g and its parameter $\mathscr{X}$ and an outer approximation of $\mathscr{A}$.

As it is clear from Table 1, an interval counterpart of constraints defined in Eq. (5) is not so obvious in order to guarantee the result of a SCSDP.

4.2 *Interval-Based Differential Constraints*

As for the interval representation of compact sets which can be from an inner approximation or an outer approximation, dynamical systems can be solved to produce interval-based representation containing all the trajectories or a subset of the set of trajectories. In other terms, $\mathscr{Y}(\mathscr{T}, \mathscr{Y}_0, \mathscr{P})$ can be inner-approximated or outer-approximated. A short review of these methods is given in the rest of this section as $\mathscr{Y}(\mathscr{T}, \mathscr{Y}_0, \mathscr{P})$ can appear in constraints of SCSDP.

4.2.1 Outer Approximation of Differential Constraints

The computation of $\mathscr{Y}(\mathscr{T}, \mathscr{Y}_0, \mathscr{P})$ solution of a system S described in Eq. (4) has been studied for a long time with interval analysis methods. A complete approach named *guaranteed numerical integration* has been defined from the seminal work of Moore [28]. More precisely, initial value problems of ordinary differential equations have been solved with interval analysis mainly based on Taylor series [4, 27–29] and more recently with Runge–Kutta-based methods [2, 10, 15]. Initial value problems for algebraic differential equations have been studied in [11, 30]. All the works aim at producing an outer approximation of the solution of the dynamical systems using interval analysis tools.

The goal of a guaranteed numerical integration method to solve Eq. (3) is to compute a sequence of time instants $0 = t_0 < t_1 < \cdots < t_n = t_{end}$ and a sequence of boxes $[\mathbf{y}_0], \ldots, [\mathbf{y}_n]$ such that $\forall j \in [0, n], [\mathbf{y}_{j+1}] \supseteq \mathbf{y}(t_j; [\mathbf{y}_j], [p])$. In this article, we focus on single-step methods that only use $[\mathbf{y}_j]$ and approximations of $\dot{\mathbf{y}}(t)$ to compute $[\mathbf{y}_{j+1}]$.

The main approach in a guaranteed numerical integration method, as presented in [29], is that each step of a validated integration scheme consists of two phases

Phase 1 One computes an a priori enclosure $[\tilde{\mathbf{y}}_j]$ of the solution such that

- $\mathbf{y}(t;[\mathbf{y}_j])$ is guaranteed to exist for all $t \in [t_j, t_{j+1}]$, i.e. along the current step, and for all $\mathbf{y}_j \in [\mathbf{y}_j]$.
- $\mathbf{y}(t;[\mathbf{y}_j]) \subseteq [\tilde{\mathbf{y}}_j]$ for all $t \in [t_j, t_{j+1}]$.
- the step-size $h_j = t_{j+1} - t_j > 0$ is as large as possible in terms of accuracy and existence proof for the *IVP* solution.

Phase 2 One computes a tighter enclosure of $[\mathbf{y}_{j+1}]$ at time t_{j+1} such that $\mathbf{y}(t_{j+1}, [\mathbf{y}_j]) \subseteq [\mathbf{y}_{j+1}]$.

A guaranteed numerical integration for a system S, as defined in Eq. (3), starts with an outer approximation of initial condition $\text{Hull}\mathscr{U}_0 = [\mathbf{y}_0]$, the parameters $\text{Hull}\mathscr{P} = [\mathbf{p}]$, and an integration step size h (or a finite horizon). It applies the two-step approach until the end of the simulation time is reached. This process builds two lists of boxes:

- The list of discretization time steps: $\{[\mathbf{y}_0], \ldots, [\mathbf{y}_{\text{end}}]\}$
- The list of a priori enclosures: $\{[\tilde{\mathbf{y}}_0], \ldots, [\tilde{\mathbf{y}}_{\text{end}}]\}$

Based on these lists, two functions depending on time can be defined

$$R : \begin{cases} \mathbb{R} \mapsto \mathbb{IR}^n \\ t \to [\mathbf{y}] \end{cases} \tag{7}$$

with $\{\mathbf{y}(t;\mathbf{y}_0) : \forall \mathbf{y}_0 \subseteq [\mathbf{y}_0]\} \subseteq [\mathbf{y}]$ and

$$\widetilde{R} : \begin{cases} \mathbb{IR} \mapsto \mathbb{IR}^n \\ [\underline{t}, \overline{t}] \to [\tilde{\mathbf{y}}] \end{cases} \tag{8}$$

with $\{\mathbf{y}(t;\mathbf{y}_0) : \forall \mathbf{y}_0 \in [\mathbf{y}_0] \wedge \forall t \in [\underline{t}, \overline{t}]\} \subseteq [\tilde{\mathbf{y}}]$.

Function R, defined in (7), is obtained by new applications of validated integration method starting from $[\mathbf{y}_k]$ at t_k and finishing at t with $t_k < t < t_{k+1}$. Function $\widetilde{R}$, defined in (8), is obtained with the union of $[\tilde{\mathbf{y}}_k]$ with $k = a, \ldots, b$ and $t_a < \underline{t} < \overline{t} < t_b$. These functions are then strictly conservative.

More abstractly, the functions R and $\widetilde{R}$ define two interval enclosures of the solution function of differential equations defined in Eq. (3).

4.2.2 Inner Approximation of Differential Constraints

More recent work deals with the inner approximation of the reachable sets of ordinary differential equations such as [5, 20]. But a lot of work remains to be done to elevate this technique to a maturity level of guaranteed numerical integration as defined in Sect. 4.2.1. A short review of [20] is made in this section as it closely follows the two-step approach of guaranteed numerical integration.

In [20] a new approach for computing inner approximations of reachable sets of dynamical systems defined by nonlinear, uncertain, ordinary differential equations is defined. It extends [19, 21] which focused on discrete-time dynamical systems. It consists in using generalized affine forms combining modal interval analysis [25] (an extension of interval arithmetic dealing with quantifiers) with affine arithmetic [8] (an extension of interval arithmetic which can take into account some correlation between variables) to produce both inner and outer approximations of the flow of an uncertain ODE with a Taylor series approach.

The given algorithm consists of three steps: (1) computing rough enclosures over a time interval $[t_i, t_{i+1}]$ of the solution and its Jacobian over the initial conditions (which is the solution of the variational equation), (2) building the Taylor models of the solution and its Jacobian, and (3) computing the inner approximations of the flow pipe using generalized affine forms.

4.3 Revisiting Branch-and-Contract Solving Method

After defining a correct interval representation of compact sets in Sect. 4, a focus on the application of Branch-and-Contract algorithm to solve SCSDP is given. More precisely, as differential constraints imply set of trajectories, an extension of contractors to deal with this new object has to be defined.

In our approach based on interval analysis and contractor programming [3], an application of constraints at some given instants in the set of trajectories and a propagation can be performed on the interval representation of the trajectories. In the rest of this section, only outer approximation of dynamical systems is considered. This section defines the two methods, contraction and propagation, on set of trajectories.

4.3.1 Contraction

The considered approach allows one to contract a specific value $[\mathbf{y}_*] = R(t^*)$, an outer approximation of the solution of the IVP at time t^* such that $\mathbf{y}(t^*) \in [\mathbf{y}_*]$ with respect to a constraint g.

The simplest example is as follows: considering a system defined by $\dot{y} = f(y)$ and $y(0) = y_0$, if a set of measures $\{\mathbf{y}_1^*, \ldots, \mathbf{y}_m^*\}$ are taken at some specific instants $t_1^*, \ldots, t_m^*$, then a contraction can be applied following the rule $[\mathbf{y}(t_i^*)] = [\mathbf{y}(t_i^*)] \cap [y_i^*], \forall i = 1, \ldots, m$. In a more complex example, if the states of the system are constrained by the help of a function g, then a contractor such as HC4-Revise or interval Newton [24] can be used. For example, a constraint such as $y(t^*)^2 - 3\cos(y(t^*)) \subset [-\infty, 0]$ can be also considered.

Remark 1 If t^* is not in the already computed time steps, then the contraction procedure adds a kth integration step to the time discretization: $\{[y_0], \ldots, [y_i], [y_k], [y_{i+1}], \ldots, [y_N]\}$ such that $t_k = t^*$.

Remark 2 The contraction at a time t^* can be easily generalized to a contraction along an interval of instants $[\underline{t}^*, \overline{t}^*]$, by the help of the $\tilde{R}$ function and a priori enclosures $[\mathbf{y}_*]$.

4.3.2 Propagation

If a contraction has been obtained, then a Picard contractor [11] on $[\tilde{\mathbf{y}}_i]$ and a validated Runge–Kutta contractor [11] on $[\mathbf{y}_i]$ can be applied on each integration step i, in order to propagate this information on the whole simulation, i.e., on all the boxes in the lists:

- Forward for $t > t^*$ with the considered differential equation
- Backward for $t < t^*$ with the inverse of the considered differential equation

A fixed-point algorithm (a loop calling alternatively the forward and the backward steps till not enough improvement is obtained with respect to a given threshold) can be also used.

Example 3 Van der Pol system is considered and it is defined by

$$\dot{x} = y$$
$$\dot{y} = 2.0(1.0 - x^2)y - x$$

with the initial conditions $x(0) \in [2.0, 2.2]$ and $y(0) \in [0.0, 0.1]$, simulated from $t = 0$ to $t = 2.0$ (see Fig. 2). A measure is obtained at $t = 1.0$, such that $y(1.0) \in [1.58, 1.62]$ and $x(1.0) \in [-0.74, -0.69]$. A contraction and a forward propagation are applied; then a backward and finally a fixed point are applied (see Fig. 2). ■

5 Numerical Example

DynIBEX library [9] implements the outer-approximation version of the proposed CSDP framework. This library allows to solve complex constraint satisfaction problems mixing bounded uncertainties, variable quantification, and differential constraints.

Kinetic parameter estimation of an enzymatic reaction example has already been considered in [18]. It aims at illustrating the SCSDP framework described in this chapter. The goal is to obtain the kinetic parameters of an enzymatic reaction as

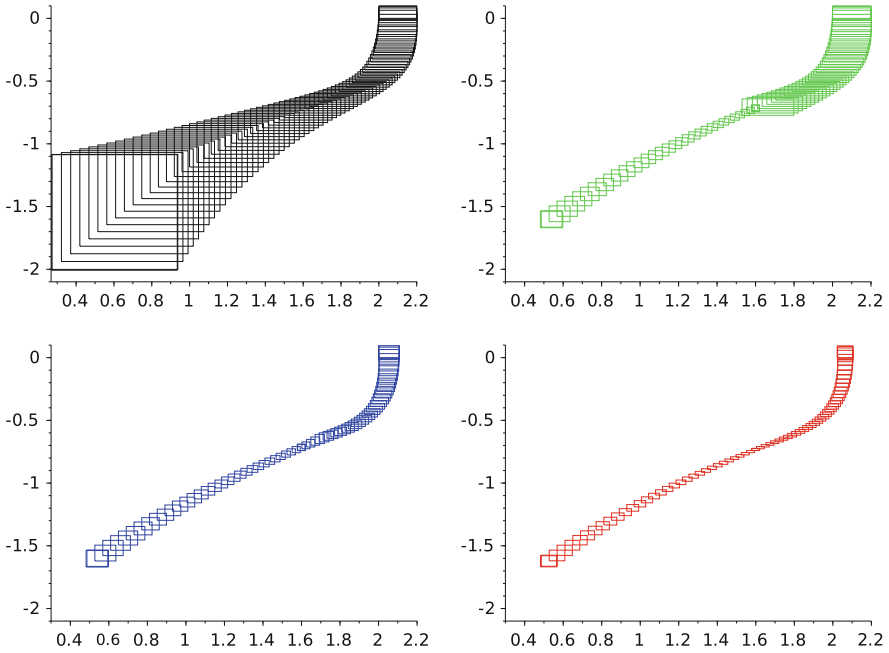


Fig. 2 Van Der Pol problem: initial (up, left), after contraction and forward propagation (up, right), after backward propagation (down, left), and after fixed-point mixing forward and backward propagation (down, right)

described in [14]. The differential equation is as follows:

$$(\mathscr{S}) \begin{cases} \dot{s}(t) = -\frac{V_{\max} s(t)}{k_s + s(t)} \\ \dot{p}(t) = \frac{V_{\max} s(t)}{k_s + s(t)} \end{cases} \tag{9}$$

with $p(t)$ and $s(t)$ the two concentrations and $V_{\max}$ and k_s the two parameters to infer from a series of measures on the concentration p during time. These measurements are shown in Table 2.

The corresponding SCSDP is as follows:

- $\mathscr{S}$ from Eq. (9)
- $\mathscr{V} = \{p_0, s_0, V_{\max}, k_s, t\}$
- $\mathscr{D} = \{[25], [0], [90, 110], [0, 10], [0, 1.0]\}$

Table 2 Measurements for the enzymatic reaction (± 0.1)

t_i	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
p_i	8.01	15.32	21.01	23.92	24.74	24.96	24.97	25.02	24.95	24.91

- $\mathcal{C} = \begin{cases} \text{Proj}_p(\mathcal{Y}(t_i, p_0, s_0, V_{\max}, k_s)) \subset p_i & \text{(Measurements)} \\ \mathcal{Y}(t, p_0, s_0, V_{\max}, k_s) \subset [0, +\infty]^2 & \text{(Nonnegative concentrations)} \end{cases}$

with the operator $\text{Proj}_x(\mathcal{Y})$ the projection over component x of the set $\mathcal{Y}$. This problem can be treated in two ways, whether we choose to consider contractors or not as depicted in Algorithm 1 with the addition of line 5 or not. A first resolution scheme is to directly apply a Branch-and-Prune algorithm on the parameters p. The function $Check()$ used to verify constraints $\mathcal{C}$ is as defined in Algorithm 2.

Algorithm 2 Check for the kinetic parameter estimation

Require: $(t_i, p_i)_{i=1,\ldots,10}$, $p_0, s_0, [V_{\max}], [k_s]$)
  bool IsUndecidable = false
  for all j = 1 to 10 **do**
    $[p]_{\text{current}} \leftarrow \text{Proj}_p(\mathcal{Y}(t_j, p_0, s_0, [V_{\max}], [k_s]))$
    $[y]_i \leftarrow \mathcal{Y}([t_{i-1}, t_i], p_0, s_0, V_{\max}, k_s)$
    if $[p]_{\text{current}} \cap p_j = \emptyset$ or $[y]_i \subseteq [0, +\infty] = \emptyset$ **then**
      return false
    end if
    if $[p]_{\text{current}} \not\subseteq p_j$ or $[y]_i \not\subseteq [0, +\infty]$ **then**
      IsUndecidable = true
    end if
  end for
  if IsUndecidable **then**
    return "undecidable"
  else
    return true
  end if

Another way is to consider the contractor described in Sect. 4.3.1. We recall that these contractors only apply to the state space of the solution of $(\mathcal{S})$, so the parameters have to be embedded into the state space. This is done in a classical way as follows:

$$(\mathcal{S}') \begin{cases} \dot{s}(t) = -\frac{V_{\max} s(t)}{k_s + s(t)} \\ \dot{p}(t) = \frac{V_{\max} s(t)}{k_s + s(t)} \\ \dot{V}_{\max} = 0 \\ \dot{k_s}(t) = 0 \end{cases} \tag{10}$$

A contractor can then be defined for the resolution of our problem by contracting and propagating each measurement. The solution of this example is depicted in Fig. 3.

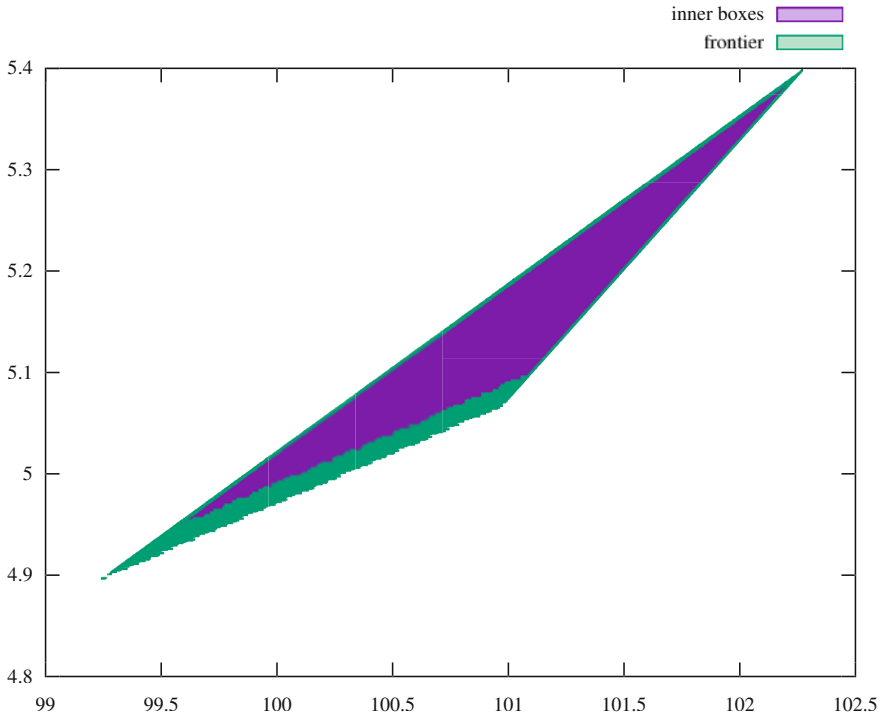


Fig. 3 Solution for the parameter estimation problem for enzymatic reaction

6 Conclusion

A constraint satisfaction problem framework has been extended to deal with differential constraints and quantification on variables. It extends other approaches [7, 18] by dealing more naturally with uncertainties and variable quantification. A discussion on the correctness of the interval representation of compact sets has been given. It emphasizes the problem of preserving the correctness of the approach when computing with tools coming from interval analysis.

A future work and extension of DynIBEX with inner approximation will be useful to address a more important class of problems. From a theoretical point of view, an extension of SCSDP with techniques coming from SMT approaches could be beneficial to increase the expressiveness of the framework, as for example, dealing with disjunctive constraints.

Acknowledgements This research benefited from the support of the "Chair Complex Systems Engineering – Ecole Polytechnique, THALES, DGA, FX, Dassault Aviation, DCNS Research, ENSTA ParisTech, Télécom ParisTech, Fondation ParisTech, and FDO ENSTA," and it is also partially funded by DGA MRIS "Safety for Complex Robotic Systems."

References

1. F. Benhamou, D. McAllester, P. Van Hentenryck, CLP (intervals) revisited. Technical Report, Providence (1994)
2. O. Bouissou, A. Chapoutot, S. Mimram, HySon: precise simulation of hybrid systems with imprecise inputs, in *IEEE Rapid System Prototyping* (2012)
3. G. Chabert, L. Jaulin, Contractor programming. Artif. Intell. **173**(11), 1079–1100 (2009)
4. X. Chen, E. Ábrahám, S. Sankaranarayanan, Flow*: an analyzer for non-linear hybrid systems, in *Proceedings of the International Conference on Computer Aided Verification* (Springer, Berlin, 2013), pp. 258–263
5. X. Chen, S. Sankaranarayanan, E. Ábrahám, Under-approximate flowpipes for non-linear continuous systems, in *Proceedings of the Conference on Formal Methods in Computer-Aided Design* (FMCAD Inc., Austin, 2014), pp. 59–66
6. J. Cruz, P. Barahona, Constraint satisfaction differential problems, in *Principles and Practice of Constraint Programming*. Lecture Notes in Computer Science, vol. 2833 (Springer, Berlin, 2003), pp. 259–273
7. J. Cruz, P. Barahona, Constraint reasoning over differential equations. Appl. Numer. Anal. Comput. Math. **1**(1), 140–154 (2004)
8. L.H. de Figueiredo, J. Stolfi, Self-validated numerical methods and applications. Brazilian Mathematics Colloquium Monographs. IMPA/CNPq, Rio de Janeiro (1997)
9. J.A. dit Sandretto, A. Chapoutot, DynIBEX: a differential constraint library for studying dynamical systems, in *Conference on Hybrid Systems: Computation and Control* (2016). Poster
10. J.A. dit Sandretto, A. Chapoutot, Validated explicit and implicit Runge-Kutta methods. Reliab. Comput. **22**, 56–77 (2016)
11. J.A. dit Sandretto, A. Chapoutot, Validated simulation of differential algebraic equations with Runge-Kutta methods. Reliab. Comput. **22**, 56–77 (2016)
12. J.A. dit Sandretto, A. Chapoutot, O. Mullier, Tuning PI controller in non-linear uncertain closed-loop systems with interval analysis, in *2nd International Workshop on Synthesis of Complex Parameters*, *OpenAccess Series in Informatics*, vol. 44, pp. 91–102. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik (2015)
13. J.A. dit Sandretto, A. Chapoutot, O. Mullier, Formal verification of robotic behaviors in presence of bounded uncertainties. J. Softw. Eng. Robot. **8**(1), 78–88 (2017)
14. D. Eveillard, D. Ropers, H. De Jong, C. Branlant, A. Bockmayr, A multi-scale constraint programming model of alternative splicing regulation. Theor. Comput. Sci. **325**(1), 3–24 (2004)
15. K. Gajda, M. Jankowska, A. Marciniak, B. Szyszka, A survey of interval Runge–Kutta and multistep methods for solving the IVP, in *Parallel Processing and Applied Mathematics*. Lecture Notes in Computer Science, vol. 4967 (Springer, Berlin, 2008), pp. 1361–1371
16. A. Goldsztejn, W. Hayes, Rigorous inner approximation of the range of functions, in *12th GAMM - IMACS International Symposium on Scientific Computing, Computer Arithmetic and Validated Numerics* (2006), p. 19
17. A. Goldsztejn, L. Jaulin, Inner approximation of the range of vector-valued functions. Reliab. Comput. **14**, 1–23 (2010)
18. A. Goldsztejn, O. Mullier, D. Eveillard, H. Hosobe, Including ODE based constraints in the standard CP framework, in *Principles and Practice of Constraint Programming*. Lecture Notes in Computer Science, vol. 6308 (Springer, Berlin, 2010), pp. 221–235
19. E. Goubault, S. Putot, Under-approximations of computations in real numbers based on generalized affine arithmetic, in *Proceedings of the Static Analysis Symposium*, vol. 4634. Lecture Notes in Computer Science (Springer, Berlin, 2007), pp. 137–152
20. E. Goubault, S. Putot, Forward inner-approximated reachability of non-linear continuous systems, in *Proceedings of the International Conference on Hybrid Systems: Computation and Control* (ACM, New York, 2017), pp. 1–10

21. E. Goubault, O. Mullier, S. Putot, M. Kieffer, Inner approximated reachability analysis, in *Proceedings of the International Conference on Hybrid Systems: Computation and Control* (ACM, New York, 2014), pp. 163–172
22. M. Janssen, Y. Deville, P. Van Hentenryck, Multistep filtering operators for ordinary differential equations, in *Principles and Practice of Constraint Programming*. Lecture Notes in Computer Science, vol. 1713 (Springer, Berlin, 1999), pp. 246–260
23. L. Jaulin, E. Walter, Set inversion via interval analysis for nonlinear bounded-error estimation. Automatica **29**(4), 1053–1064 (1993)
24. L. Jaulin, M. Kieffer, O. Didrit, E. Walter, *Applied Interval Analysis* (Springer, Berlin, 2001)
25. E. Kaucher, *Interval Analysis in the Extended Interval Space IR* (Springer, Berlin, 1980), pp. 33–49
26. Y. Lebbah, O. Lhomme, Accelerating filtering techniques for numeric CSPs. J. Artif. Intell. **139**(1), 109–132 (2002)
27. R.J. Lohner, Enclosing the solutions of ordinary initial and boundary value problems, in *Computer Arithmetic* (B.G. Teubner, Stuttgart, 1987), pp. 255–286
28. R.E. Moore, *Interval Analysis*. Series in Automatic Computation (Prentice Hall, Englewood Cliffs, 1966)
29. N.S. Nedialkov, K. Jackson, G. Corliss, Validated solutions of initial value problems for ordinary differential equations. Appl. Math. Comput. **105**(1), 21–68 (1999)
30. A. Rauh, M. Brill, C. GüNther, A novel interval arithmetic approach for solving differential-algebraic equations with ValEncIA-IVP. Int. J. Appl. Math. Comput. Sci. **19**(3), 381–397 (2009)
31. F. Rossi, P. Van Beek, T. Walsh, *Handbook of Constraint Programming* (Elsevier, Amsterdam, 2006)
32. M. Rueher, Solving continuous constraint systems, in *International Conference on Computer Graphics and Artificial Intelligence* (2005)

Approximate Computing and Its Application to Hardware Security

Weiqiang Liu, Chongyan Gu, Gang Qu, and Máire O'Neill

Abstract The demand for high speed and low power in nanoscale integrated circuits (ICs) for many applications, such as image and multimedia data processing, artificial intelligence, and machine learning, where results of the highest accuracy may not be needed, has motivated the development of approximate computing. Approximate circuits, in particular approximate arithmetic units, have been studied extensively and made significant impact on the power performance of such systems. The first goal of this chapter is to review both the existing approximate arithmetic circuitries, which include adders, multipliers, and dividers, and popular approximate algorithms. The second goal of this chapter is to explore broader applications of approximate computing. As an example, we review two case studies, one on a lightweight device authentication scheme based on erroneous adders and the other one on information hiding behind a newly proposed approximate data format. This approach of applying approximate computing in security is interesting and promising in the Internet of things (IoT) domain where the devices are extremely resource constrained and cannot afford conventional cryptographic solutions to provide data security and user privacy. We also discuss the potential of approximate computing in building hardware security primitives for cyber physical system (CPS) and IoT devices.

W. Liu (✉)
College of Electronic and Information Engineering, Nanjing University of Aeronautics and Astronautics (NUAA), Jiangsu, China
e-mail: liuweiqiang@nuaa.edu.cn

C. Gu · M. O'Neill
Centre for Secure Information Technologies (CSIT), Institute of Electronics, Communications & Information Technology (ECIT), Queen's University Belfast (QUB), Belfast, UK
e-mail: cgu01@qub.ac.uk; m.oneill@ecit.qub.ac.uk

G. Qu
Department of Electrical and Computer Engineering and Institute for Systems Research, University of Maryland, College Park, MD, USA
e-mail: gangqu@umd.edu

Ç. K. Koç (ed.), *Cyber-Physical Systems Security*,
https://doi.org/10.1007/978-3-319-98935-8_3

1 Introduction

The performance of various computing systems, from sensors, smartphones, and other mobile devices to servers, supercomputers, and cloud computing data centers, has been increasing dramatically in the past several decades in line with the advances in IC design according to the famous Moore's Law. However, as Moore's Law is approaching its limit [34], the conventional techniques are unable to further improve the computing performance of systems with limited power budget, i.e., the power consumption restricts the performance of computing systems. It becomes challenging to continue improving system performance by conventional CMOS technologies. One of the major concerns is the increasing on-chip power density and the power consumption requirements by the application. Chip designs at the nanoscale urgently require new approaches and paradigms to reduce low-power and high-performance computing systems.

Dynamically adjusting the supply voltage and clock frequency is one of the most effective low-power design methods [32]. However, as we push the supply voltage closer and closer to the threshold voltage, the circuit delay increases and may malfunction [19]. This coupled with the high integration density makes it very challenging to test and verify the design. Indeed, due to the lower-power supply voltage and the higher integration density at the nanoscale of a circuit design, ensuring fully correct computation results from ICs will result in a dramatic increase in cost. The International Technology Roadmap for Semiconductors (ITRS) states that the cost of manufacturing verification and testing can be greatly reduced by tolerating errors for devices [39]. Therefore, without affecting the usage and perception, acceptable reduction of the computing accuracy can effectively reduce both the power consumption and test/verification cost.

Due to the error-resilient and fault-tolerant ability of the human brain, visual and auditory systems, certain level of processing errors will not affect the quality of human perception and recognition of the processed data [14, 59]. Examples have been reported in artificial intelligence (AI), machine learning, data mining, multimedia signal processing [14, 35, 36, 59] etc. In these applications, the data includes noisy or redundant information, and therefore it makes little sense to compute the precise result based on erroneous data or perform redundant computation.

Motivated by the above challenges, approximate computing (also known as inexact computing) has attracted significant attention from both academia and industry in recent years [25, 41, 80]. Approximate computing can reduce power consumption and improve system performance by introducing acceptable errors. Therefore, we can introduce computation accuracy as a third design metrics in addition to delay and power consumption as shown in Fig. 1. It depicts a three-dimension (3D) design space by taking into account the computational accuracy, performance, and power consumption of approximate computing circuits.

Not surprisingly, some of the early research results have also made their impact on industry. Google's deep learning (DL) chip, the tensor processing unit (TPU), achieves a significant improvement in processing performance by using

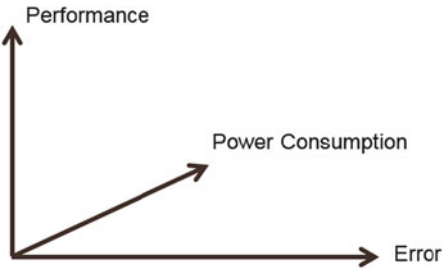


Fig. 1 A 3D design relationship of performance, power, and accuracy for approximate computing

approximate computing techniques [42]. The performance of TPU outperforms over traditional GPU and CPU processors by 15–30 times. It is a crucial component in AlphaGo which has defeated human Go champion. As another example, with the support of the Defense Advanced Research Projects Agency (DARPA), Bates developed an approximate computing chip based on an approximate arithmetic unit and founded a company known as Singular Computing [72]. This chip is used in DARPA's UPSIDE project to enable real-time video target tracking on drones. Compared to traditional processors, it can increase the speed of video processing by 100 times and consumes less than 2% of a traditional processor power by using a Singular Computing chip. Finally, we mention that both IBM [8] and ARM [65] have investigated heavily on approximate computing. This evidence shows that approximate computing is already making significant impact on the design of today's application-specific processors, and it will have higher potential in the design for future systems.

Speaking of future systems, the emerging IoT are perhaps the one that will have the most influence on our lives. The IoT era has already arrived with billions of electronics devices surrounding us, and it is predicted that there will be more than 50 billion connected IoT devices by 2020 [62]. They will have a large impact on a wide range of markets, from wearable health-care devices to embedded systems in smart cars, many of which will be underpinned by devices which are limited with regard to computation and power consumption. This has led to a high demand for cryptographic devices that can provide authentication to protect user privacy and data security. Conventional cryptographic approaches, which involve complex cryptographic algorithms, are unsuitable to be implemented on IoT devices as they incur significant timing, energy, and area overhead [66]. This opens the opportunity for developing low-cost lightweight security primitives based on approximate computing. For example, information could be hidden into the process and results of the approximate computing to protect design intellectual protection (IP) as watermark, fingerprint, or lightweight encryption [19].

Approximate computing has also been used to implement deep neural network (DNN) algorithms which have found applications in solving hardware security problems such as side-channel analysis (SCA)-based attacks [20], attacks on physical unclonable function (PUF) [38], Hardware Trojan (HT) detection [28],

etc. Hence, an approximate DNN design could benefit and revolutionize hardware security-related applications.

Previously, there are several excellent surveys on approximate computing. Jiang et al. [41] reviewed and classified current designs of approximate arithmetic circuits. A complete survey of existing approximate computing work is presented in [80]. Unlike this work, we focus our discussion on the implementation of approximate arithmetic circuits and their applications in cybersecurity. Specifically, this chapter contributes in the following ways:

- A detailed classification and review of current approximate circuits, in particular approximate arithmetic circuits, including adders, multipliers, and dividers are introduced.
- Current approximate error-tolerant algorithms are briefly reviewed, and their applications are discussed.
- Two case studies demonstrating lightweight authentication and security primitives using approximate computing are presented.
- Future works on applying approximate computing into different cyber-security scenarios, including SCA techniques, PUFs, and logic obfuscation techniques, are also discussed.

2 Approximate Circuit

Arithmetic units including adders, multipliers, and dividers play important roles in processors, which significantly influence the performance and the power consumption of the whole computing system. It is expected to achieve higher speed and power efficiency as well as error tolerance for cognitive applications, e.g., recognition, data analysis, and computer vision. These motivated the fast development of approximate arithmetic designs. The design of approximate computing circuits mainly uses voltage-based probability CMOS techniques and logic reduction and pruning methods. Probability CMOS technique reduces energy consumption by allocating higher supply voltages to important areas to ensure the accuracy of most significant bits (MSBs) while appropriately reducing the supply voltage of least significant bits (LSBs) that have a less effect on the result. Cheemalavagu et al. [9] proposed a probabilistic adder that uses a conventional precision adder structure by providing various supply voltages for different bits depending on the degree of importance. However, this technique requires a higher implementation cost and generates uncontrollable errors, which restrict its subsequent applications. Therefore, most of the approximate computing circuits are based on the logic reduction and pruning methods. In cognitive computing applications, e.g., image recognition, machine learning, and pattern recognition, the key arithmetic units mainly include adders and multipliers. Therefore, high-performance and low-power adders and multipliers have been extensively studied.

Table 1 An overview of approximate adder circuits

Type			Previous works
Speculative adders	Non-segmented		SSA [54], ACA [78], SHCA [18]
	Segmented	Non-MUX	ESA [60], ETAII [88], ACAA [43], GeArA [71]
		MUX	SCSA [17], ACSA [44], GDA [83], CCBA [6]
Transistor-based approximate full adders			LOA [57], AMAs [24], AXAs [81], InXAs [2]

2.1 Approximate Adders

An overview and classification of current approximate adders are listed in Table 1. The concept of an approximate adder was first proposed for asynchronous adders [63], while the first synchronous speculative adder was proposed by Intel [54]. It has been found that full adders have a shorter carry propagation length for random operands than the length of a full carry chain. Hence, it gets faster and more energy-efficient adders by designing shorter carry chains using some specific bits. Similar as this idea, the researchers designed a family of speculative approximate adders, including non-segmented speculative approximate adders and segmented speculative approximate adders.

The non-segmented speculative approximate adder includes synchronous speculative adder (SSA) [54], almost correct adder (ACA) [78], speculative Han-Carlson adder (SHCA) [18], etc. The segmented approximate adder is a type of speculative approximate adder. The main difference is that the segmented adder divides the adder into several sub-adders and the carry propagation is computed in parallel in each sub-adder. Based on whether they have a multiplexer (MUX) or not, the segmented approximate adder can be divided into two categories, MUX-based segmented approximate adder and non-MUX-based segmented adder. The non-MUX-based segmented approximate adder includes equal segmentation adder (ESA) [60], error tolerant adder type II (ETAII) [88], accuracy configurable approximate adder (ACAA) [43], and generalized accuracy configurable approximate adder (GeArA) [71]. The MUX-based segmented approximate adder is mainly based on a carry skip or carry-select adder, including speculative carry select adder (SCSA) [17], approximate carry skip adder (ACSA) [44], gracefully-degrading adder (GDA) [83], and carry cut-back adder (CCBA) [6].

The speculative approximate adder is primarily targeted at increasing the speed and performance, while the transistor-based approximate full adder can significantly reduce power consumption. By reducing the number of transistors and basic gates from the exact full adder, an energy-efficient approximate full adder can be achieved. The first approximate full adder is a bio-inspired LOA [57], in which the MSB is implemented by approximate full adders and the LSB uses OR gates. An AND gate is used for carry propagation and the critical path delay is determined by the MSBs, which consumes very little power due to its simple structure. Gupta et al. [24] proposed five approximate mirror adders (AMAs) based on the traditional

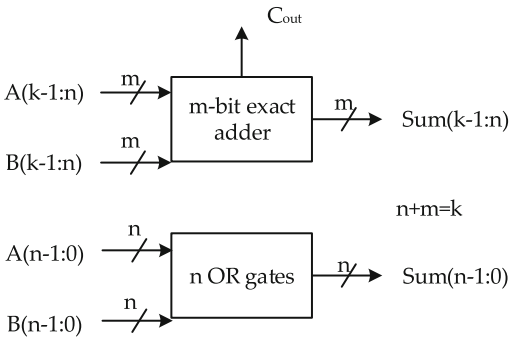


Fig. 2 The revised LOA adder structure

mirror adder. The approximate full adder also includes approximate XOR-/NXOR-based full adders (AXAs) [81] and Inexact Adder cells (InXAs) [2].

The research in [41] shows that SCSA and ACA adders present better accuracy, while ESA has the lowest accuracy and LOA exhibits medium accuracy. In terms of hardware performance, SCSA has higher power consumption. The speed of speculative approximate adder is faster; however it consumes more power. Although the speed of approximate full adder is slower, it demonstrates low power consumption and consumes less hardware resources.

The LOA design is chosen in this chapter as an example to illustrate the approximate adder. For an approximate floating-point adder, a revised LOA adder is used, as it significantly reduces the critical path by ignoring the lower carry bits [51]. A k-bit LOA consists of two parts as shown in Fig. 2, an m-bit exact adder and an n-bit inexact adder. The m-bit adder is used for the m MSBs of the sum, while the n-bit adder consists of OR gates to compute the addition of n LSBs, i.e., the lower n-bit adder is an array of n 2-input OR gates. In the original LOA design, an additional AND gate is used for generating the most significant carry bit of the n-bit adder; all carry bits in the n-bit inexact adder are ignored to further reduce the critical path.

2.2 Approximate Multipliers

The approximate multipliers shown in Table 2 can be classified based on the approximate design of different components. The idea of approximating operands, known as logarithmic multiplier (LM), has been proposed by Mitchell in the 1960s [58]. The LM transforms multiplication operation into additions in the logarithm domain to achieve low power consumption. However, its accuracy is low. An approximate logarithmic multiplier (ALM) and an iterative approximation logarithmic multiplier (IALM) have been proposed in [53]. Compared to the traditional LM, ALM achieves higher accuracy and lower power consumption by introducing

Table 2 An overview of approximate multiplier circuits

Type of approximate multipliers		Previous works
Approximate operand	Logarithmic	LM [58], ALM [53], IALM [53]
	Non-logarithmic	ETM [48], DRUM [29]
Approximate partial product generation	Non-booth encoding	UDM [46]
	Booth encoding	R4ABMs [52], R8ABMs [40]
Approximate partial product tree	Truncated	BAM [57]
	Untruncated	PPPM [86], R4ABMs [52]
Approximate counters or compressors	Normal binary	ANBCs [61]
	Redundant binary	ARBCs [7]

an approximate mantissa adder. IALM significantly improves the performance of the LM by introducing an iterative mechanism; however, its power consumption is relatively higher. Recently, the design of approximate multipliers based on the dynamic scaling of operands has been proposed, including fault tolerant multipliers (ETM) [48] and dynamic range multipliers (DRUM) [29]. They have very low power consumption; however, their accuracy is also lower than others [53].

The state-of-the-art high-performance multipliers normally include three parts: partial product generation, partial product accumulation, and final addition. Much research has been conducted on the approximate design of each part. Kulkarni et al. [46] proposed an approximate 2×2 multiplier, which can be used to construct larger sized underdesigned multipliers (UDMs). Approximate Booth multipliers, a radix-4 approximate Booth multiplier (R4ABM) and a radix-8 approximate Booth multiplier (R8ABM), based on approximate radix-4 modified Booth encoding (MBE) algorithms and a regular partial product array that employs an approximate Wallace tree, have been proposed in [52] and [40]. The R4ABM multiplier with an approximate factor of 14 is the most efficient design when considering both power-delay product and the error metric. Traditional Booth multipliers, e.g., broken-array multiplier (BAM) [57], truncate partial product compression trees; however, this design has a lower accuracy. Zervakis et al. [86] proposed a partial product perforation (PPP) technique that reduces the number of partial products.

The approximate radix-4 Booth multiplier is further illustrated as an example in this chapter to show the design of approximate multipliers. A Booth multiplier consists of three parts: partial product generation using a Booth encoder, partial product accumulation using compressors, and final product generation using a fast adder.

The Booth encoder plays an important role in the Booth multiplier, which reduces the number of partial product rows by half. Consider the multiplication of two N-bit integers, i.e., a multiplicand $\mathbf{A}$ and a multiplier $\mathbf{B}$ in two's complement, which is

given as follows:

$$A = -a_{N-1}2^{N-1} + \sum_{i=0}^{N-2} a_i 2^i \tag{1}$$

$$B = -b_{N-1}2^{N-1} + \sum_{i=0}^{N-2} b_i 2^i \tag{2}$$

In a Booth encoder, each group is decoded by selecting the partial products as $-2\mathbf{A}$, $-\mathbf{A}$, 0, $\mathbf{A}$, or $2\mathbf{A}$. The negation operation is performed by inverting each bit of $\mathbf{A}$ and adding a "1" (defined as Neg) to the LSB [45, 84].

The circuit diagrams of the radix-4 Booth encoder and decoder are provided in [84]. The output, i.e., the partial product pp_{ij}, of the Booth encoder is given as follows:

$$pp_{ij} = (b_{2i} \bigoplus b_{2i-1})(b_{2i} \bigoplus a_j) + \overline{(b_{2i} \bigoplus b_{2i-1})}(b_{2i+1} \bigoplus b_{2i})(b_{2i+1} \bigoplus a_{j-1}) \tag{3}$$

The first R4ABM, which uses radix-4 approximate Booth encoding-2 (R4ABE2) and the regular approximate partial product array, has been proposed in [52]. The truth table of the R4ABE2 method is shown in Fig. 3, where ① denotes a "0" entry that has been replaced by a "1"; eight entries in the K-map are modified to simplify the logic of the Booth encoding. The strategy for R4ABE2 is that in addition to having a symmetric truth table with a small error, the number of prime implicants (identified by rectangle) should be as small as possible.

The gate-level circuit of R4ABE2 is shown in Fig. 4. R4ABE2 only requires one XOR-2 gate by using transmission gates, so the transistor count of R4ABE2 is 4.

a_ia_{j-1} \ $b_{2i+1}b_{2i}b_{2i-1}$	000	001	011	010	110	111	101	100
00	0	0	0	0	1	①	1	1
01	0	0	⓪	0	1	①	1	①
11	①	1	1	1	0	0	0	0
10	①	1	①	1	0	0	0	⓪

Fig. 3 K-map of R4ABE2

Fig. 4 The gate-level circuit of R4ABE2

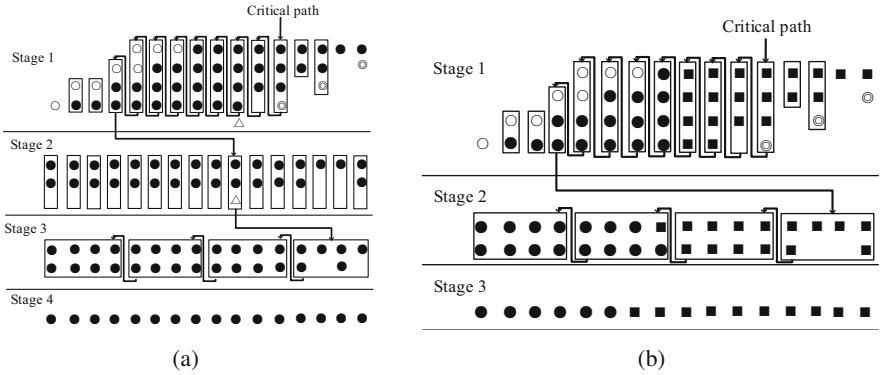


Fig. 5 The 8 × 8 Booth multiplier: (**a**) Exact irregular partial product array, (**b**) Approximate regular partial product array by ignoring the Neg term in the fifth partial product row. The exact partial product term is represented by filled circle, while the approximate partial product term is represented by filled square. Open circle and circle within circle represent the sign extension bit and the Neg term

R4ABE2 reduces the complexity of the Booth encoder by over 88% and improves the delay by 60% compared with MBE.

For a more regular partial product array (requiring a smaller reduction stage), the Neg term in the $(N/2 + 1)$th row of the approximate design of a Booth multiplier can be ignored (shown as $\triangle$ in Fig. 5a). For an N-bit radix-4 Booth multiplier when N is a power of 2, removing the extra Neg term significantly reduces the critical path, area, and power when the 4-2 compressor is used for the partial product accumulation. In the approximate partial product array (Fig. 5b), one reduction stage is saved; this significantly reduces the complexity and critical path delay. The error rate of the approximate partial product array with the ignored Neg bit is 37.5%, and its logic function is given as follows:

$$\mathrm{Neg}_{\frac{N}{2}-1} = (b_{2N+1}\overline{b_{2N}} + b_{2N+1})\overline{b_{2N-1}} = b_{2N+1}\overline{b_{2N}b_{2N-1}} \tag{4}$$

2.3 *Approximate Dividers*

As mentioned above, both approximate adders and approximate multipliers have been studied quite extensively. However, the design of approximate arithmetic division has not been fully analyzed. The computation of division is different from multiplication; division is mostly a sequential process, while multiplication can be executed as a multi-operand parallel addition. Thus, when considering approximate computing for division, an approach targeting the sequential nature of division must be developed; for example, when calculating the quotient, the error introduced previously will affect the next iteration. Therefore, a proper approximate design has to mitigate error propagation.

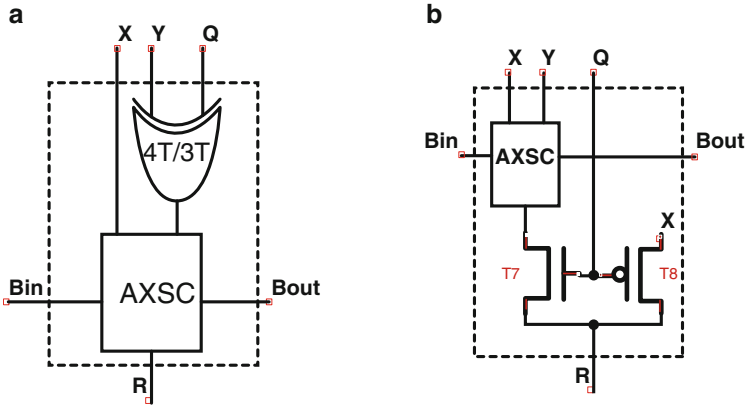


Fig. 6 Examples of restoring and non-restoring divider cells: (**a**) Non-restoring divider cells, AXDnr [64], (**b**) Restoring divider cells, AXDr [12]

Chen et al. [11] have proposed the design of an AXDnr, shown in Fig. 6a; different AXDnr designs have been proposed by replacing the logic primitives with approximate subtractors. Chen et al. [13] have proposed designs of an approximate high-radix divider, in which an approximate signed-digit adder cell is utilized to replace the exact signed-digit adder cell. A type of dynamic approximate divider has been investigated in [30], in which, for different lengths of input operands, leading-one detectors and a barrel shifter are utilized to reduce the inaccuracy. Chen et al. [11] have proposed a few inexact subtractor cells inexact subtractor cells (AXSCs) at transistor level for the design of an AXDnr. As different types of divider, restoring and non-restoring dividers have been analyzed for approximate computing; [12] has shown that an AXDr has better performance than AXDnr with respect to power consumption while also introducing a small degradation in accuracy.

The AXDr is shown in Fig. 6b. A non-restoring divider needs a remainder correction circuit for adjusting the sign of the remainder to be consistent with the dividend, thus incurring additional circuit complexity and power consumption. This can be improved by utilizing a restoring array divider [64]. As shown in Fig. 7, four

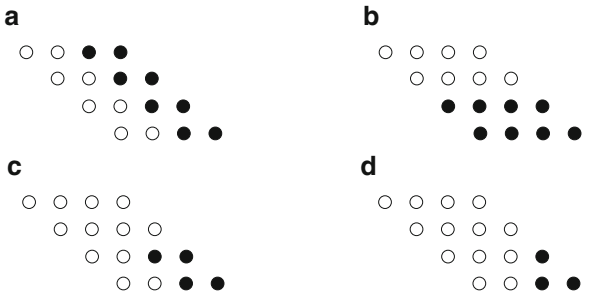


Fig. 7 Four division replacement schemes used in approximate array dividers [12]: (**a**) vertical replacement, (**b**) horizontal replacement, (**c**) square replacement, and (**d**) triangle replacement

types of replacement schemes, including vertical, horizontal, square, and triangle replacements, are used for the division operation.

3 Approximate Software/Algorithm

The main techniques used in the design of approximate algorithms include precision scaling [85], loop perforation [74], task skipping [70], and task dropping [21]. Accuracy scaling techniques reduce computational and storage requirements by varying the precision or length of the operation. Yeh et al. [85] proposed an architecture with a hierarchical floating-point unit that leverages dynamic precision reduction to enable efficient float-point unit sharing among multiple cores. This technique can gradually reduce the accuracy of the run time until the minimum accuracy of the value is reached. Tian et al. [74] proposed a precision-scaled off-chip data access technique for clustering problems to reduce energy consumption. The loop perforation technique reduces computations by skipping some iterations of the loop. An example of code without the loop perforation technique that involves skipping iterations is shown in Fig. 8 (Table 3).

The application of approximate computing, e.g., using the precision scaling technique, in DNN algorithms has already been widely studied. Since the training is more sensitive to accuracy, to reduce the cost of storage and the computational requirements, the precision scaling technique mainly focuses on the precise reduction of operands and operations, e.g., dynamic fixed-point technique [55], weight

```
1   // Original code without loop perforation
2   for ( int i = 0; i < N; i++ ) {
3       // ...
4   }
5
6   // Modified code with skipping n iterations each time
7   for ( int i = 0; i < N; i++ ) {
8       // ...
9       i = i + skipping_factor;
10  }
```

Fig. 8 An example of loop perforation technique

Table 3 An overview of approximate algorithms

Approximate algorithms	Previous works
Precision scaling	[85]
Loop perforation	[74]
Task skipping/dropping	[21, 70]
Low-precision DNN	[10, 15, 16, 55, 87]
Sparsity and pruning	[1, 26]

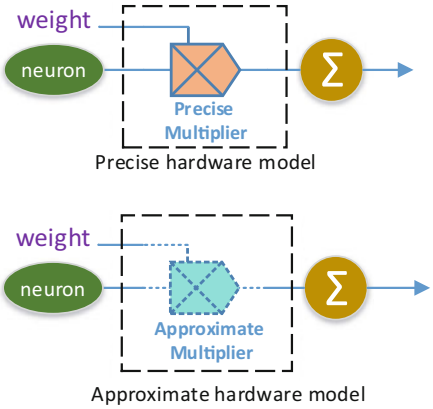


Fig. 9 The application of approximate computing to neural networks

reduction [15], activation reduction function [16], nonlinear quantization [87], and weight sharing [10]. In addition, DNNs also utilize other techniques, including the sparsity of activation functions [1] and network pruning techniques [26], to reduce computations and the size of network models.

Venkataramani et al. [77] comprehensively studies various applications for approximate computing, including image searching, recognition and detection, image segmentation, as well as data classification. Yazdanbakhsh et al. [82] presented a set of approximate computing benchmarks for different platforms. Figure 9 shows an example of the application of approximate computing to energy-efficient machine learning implementation. Since the approximate circuit could reduce the cost of storage and the computational requirements, an approximate circuit is utilized to replace the precise circuit. Then, to accelerate the computing, machine learning algorithms are involved by setting neuron and weight as parameters.

4 Approximate Computing for Hardware Security

4.1 Security Primitives Based on Approximate Computing

To minimize the power cost of IoT devices while still providing a practical security solution, Gao et al. proposed a security primitive in [19], based on basic arithmetic operations carried out by approximate function units, to embed information for authentication and other security-related applications.

4.1.1 Floating-Point Format with Embedding Security

In the work [19], it has been shown that floating-point-based approximate arithmetic computing can be employed for embedding security as shown in Fig. 10. The IEEE

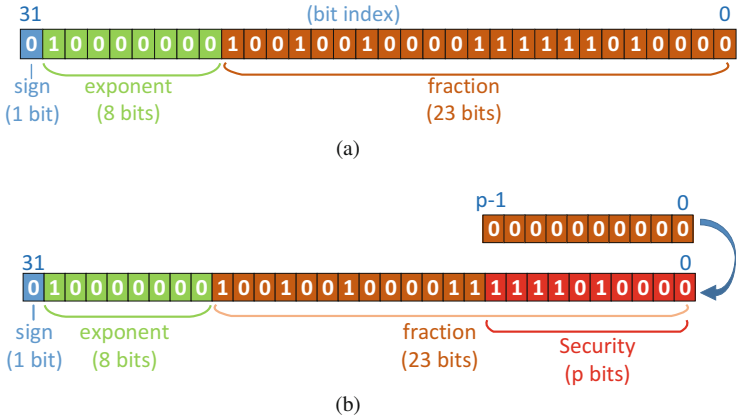


Fig. 10 The application of approximate computing to extract security: (**a**) IEEE 754 single-precision floating-point format for 32-bit data and (**b**) approximate format with security extraction. The last p LSB bits can be used as security bits to embed information

754 standard [37] specifies a binary floating-point format as having 1 sign bit, 8 exponent bits, and 23 fraction bits as shown in Fig. 10a. The sign bit determines the sign of the number, and it represents 1 or -1 if the leading bit is 0 or 1, respectively. The exponent is either an 8-bit signed integer from -128 to 127 or an 8-bit unsigned integer from 0 to 255. The significand includes 23 fraction bits to the right of the binary point.

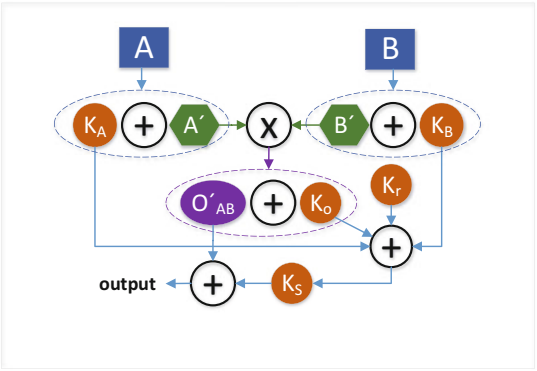
The value of IEEE 754-formatted data is computed using Eq. (5) by a given 32-bit binary data with a given biased sign, exponent e (the 8-bit unsigned integer), and a 23-bit fraction. For the example of Fig. 10a, the value is equal to 3.14159 in decimal format using Eq. (5):

$$\text{value} = (-1)^{b_{31}} \times \left(1 + \sum_{i=1}^{23} b_{23-i} 2^{-i}\right) \times 2^{e-127} \tag{5}$$

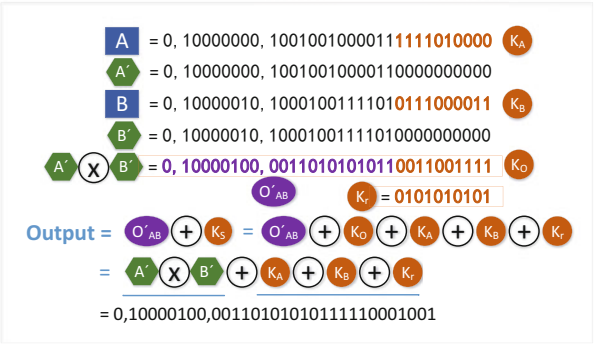
Since the LSB p bits in the fraction have little impact on the value, they can be directly used as *security* bits, as shown in Fig. 10b, to embed information without impacting the other $32-p$ bits. In this example, the approximate value is 3.1413574 by setting the last 10 bits ($p = 9$) to 0. The error introduced to the precision value is 0.0074%, which means the last p bits introduce less than 2^{p-24} error compared to the precision format.

4.1.2 Approximate Computing with Embedded Security Information

Figure 11 shows the process and an example of applying approximate computing to information hiding. Two real numbers A and B can be written as $A = A' \oplus K_A$ and



(a)



(b)

Fig. 11 An example of the application of approximate computing to information embedding: (**a**) Flowchart of approximate computing with information embedding proposed by Gao et al. [19] and (**b**) an example of approximate computing with information hiding

$B = B' \oplus K_B$ using the approximate format introduced in Sect. 4.1.1, where A' and B' are the numbers A and B in approximate format that the last p bits are replaced by 0s; K_A and K_B are the last p bits of A and B. $\oplus$ is an XOR operation.

The process of executing information-embedded approximate computing proposed in [19] mainly includes the following steps. A multiplication operation of A and B, $A \times B$, is demonstrated in this example:

- Represent A and B in the approximate format: $A = A' \oplus K_A$ and $B = B' \oplus K_B$, respectively.
- Calculate and represent $A' \times B'$ in the approximate format: $A' \times B' = O'_{AB} \oplus K_O$.
- Generate $K_S = K_A \oplus K_B \oplus K_O \oplus K_r$, where K_r is a random key.
- Calculate the result $O'_{AB} \oplus K_S$ as the result of $A \times B$.

An example of the process of hiding information into approximate computing is shown in Fig. 11a. The numbers A and B are 3.14159 and 12.31, respectively. $A \times B = 3.14159 \times 12.31 = 38.6729729$ is obtained for the precise computation;

$O'_{AB} = A' \times B' = 3.1413574 \times 12.30957 = 38.6687588$ is calculated for the approximate computation with $p = 10$. The final result with security information embedded is computed as $O'_{AB} \oplus K_S = 38.6729729$, with only a 0.00448 percentage accuracy loss over the accurate result. Hence, compared to the straight approximate computing, this approach achieves approximate computing and information hiding at the same time, which can significantly reduce power and hardware resource consumption. Moreover, K_S can be used as a function of K_A, K_B, K_O, and K_r, e.g., $F(K_A, K_B, K_O, K_r)$, for the application of IP watermarking, digital fingerprinting, and lightweight encryption. For example, the IP owner's digital signature can be used as the key K_r to enable information embedding for the application of IP watermarking. Similarly, for digital fingerprinting, a unique fingerprint of each device can be utilized and embedded in the p LSBs. For the same operands of approximate computing, different key K_r values can be embedded and used to differentiate individual devices.

4.2 A Low-Voltage Approximate Computing Adder for Authentication

Due to the ubiquitous nature of IoT devices, lightweight authentication of an entity is one of the most fundamental problems in providing IoT security. A novel voltage over-scaling (VOS)-based lightweight authentication approach is presented in [3] to address this challenge. By utilizing the VOS technique, commonly employed in approximate computing to reduce the power, to exacerbate the effects of process variation and extract information related to its variation, it can be used for security purpose. Digital circuits and systems are normally operated under the nominal voltage to guarantee correct outputs. Properly reducing the operating voltage under the prescribed margin can considerably save power consumption. However, over scaling voltage can generate timing errors and thus sacrifice the output quality. The errors are related to the process variation and could be tolerated by certain applications such as image processing. Hence, a two-factor authentication scheme that uses passwords and hardware properties is proposed to achieve lightweight authentication for IoT.

The authentication protocol, shown in Fig. 12, utilizes a VOS computation unit that can generate process variation-dependent errors. The authentication protocol is divided into two stages, enrollment and authentication. For the enrollment, device i has a password $\mathbf{K}$, composed of two keys $\mathbf{K} = (k1, k2)$, and enrolled in a server's database. Moreover, the error pattern of an adder unit in device i is derived and stored in the server. For the authentication, a random string $\mathbf{R}$ is generated by the server and sent to device i. Device i calculates $\mathbf{L}$ according to the equation $\mathbf{L} = \mathbf{R} + k1$ using the adder unit and then computes $\mathbf{Y}$, where $\mathbf{Y} = \mathbf{L} \oplus k2$. $\mathbf{Y}$ is sent back to the server. The server calculates $\mathbf{L}$ and $\mathbf{L'}$, where $\mathbf{L}' = \mathbf{M}(\mathbf{R}, k1)$. If the hamming distance of $\mathbf{L}$ and $\mathbf{L'}$ is smaller than τ, the threshold of error tolerance, the authentication succeeds. Otherwise, the authentication event aborts.

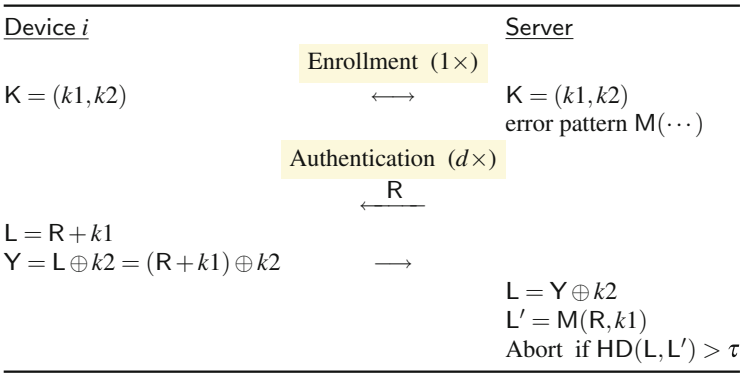


Fig. 12 The lightweight authentication protocol based on approximate computation unit [3]

5 Future Research Directions

Accelerating machine learning using approximate computing can be generally applied to side-channel attacks (SCAs), physical unclonable function (PUF) modeling attacks, and the detection of Hardware Trojans, which will be discussed in details as follows.

5.1 PUFs and SCAs

A PUF is a security primitive which utilizes the inherent process variations present during manufacturing in order to generate a unique digital fingerprint that is intrinsic to the device itself. As this natural variation between the silicon dies is out of the manufacturer's control, they are inherently difficult to clone, as well as providing additional tamper-evident properties [22]. PUFs also offer improved security as they can produce unique keys on the fly without the need for storage in non-volatile memory (NVM) on the device which reduces the risk of physical attack and saves hardware resources. These properties have a number of advantages over current state-of-the-art alternatives, opening up interesting opportunities for higher-level security protocols such as key storage and device authentication for both application specific integrated circuit (ASIC) and field programmable gate array (FPGA)-based devices.

PUF architectures can be broadly classified into Weak PUF and Strong PUF (SPUF) types as discussed in [23]. Weak PUFs have a limited challenge response pair (CRP) space and, in the extreme case, only have a single response. Therefore, they are more suited to applications such as key storage or for seeding a pseudo random number generator (PRNG), where the response never leaves the chip and is only accessed as required. In contrast, SPUFs have a large number of possible

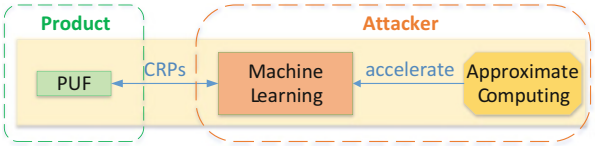


Fig. 13 An example of the application of machine learning to PUFs. The approximate computing is presented to accelerate/improve the efficiency of machine learning attacks

CRPs, whereby a large number of random challenges will return a random response unique to each challenge, as well as the physical device. By design, this implies that the requirement for a much larger entropy pool such that related challenges should not lead to related responses on the same device. Hence, SPUFs have been proposed for applications such as lightweight mutual authentication.

However, most SPUF architectures based on linear and additive functions have been shown to be vulnerable to machine learning (ML) attacks. To date, linear regression (LR), support vector machine (SVM), and evolutionary strategies (ES)-based ML methods have been widely utilized to attack PUFs [4, 5, 68, 69, 75].

In order to prevent modeling attacks, SPUF designs have been enhanced by increasing their complexity to raise the bar of attacking efforts of the adversaries. Figure 13 shows an example of the application of machine learning to SPUFs. Since approximate computing can be used to improve significantly the effectiveness of machine learning attacks, applying approximate computing-based modeling attacks to break SPUF designs could dramatically increase the attack success rate and how to mitigate this will be a more interesting and challenging problem.

5.2 SCAs

Machine learning techniques have also been used for improving SCAs attacks. A relatively new approach to profiling attacks involves the application of machine learning techniques to improve their efficiency and success. It has been shown that these attacks can be even more powerful than template attacks in practice, as less assumptions are required on the distribution of the underlying trace data [49, 56]. Much of the research to date has centered on the use of SVMs [31, 33] and random forests [50]. Research by Lerman et al. [49] showed how such approaches can be used to uncover the key of a protected (masked) advanced encryption standard (AES) implementation. A general process illustration of this idea is shown in Fig. 14. Gilmore et al. in [20] improved upon this research by investigating the novel application of a neuron network (NN)-based attack against a masked AES design. This two-stage attack first uses a NN model to recover the mask, with a second NN model built to recover the masked secret data. Combining the knowledge recovered from both attacks allows subsequent key recovery with only a single trace. Parallel work has shown how to recover the secret key with only a single model and no

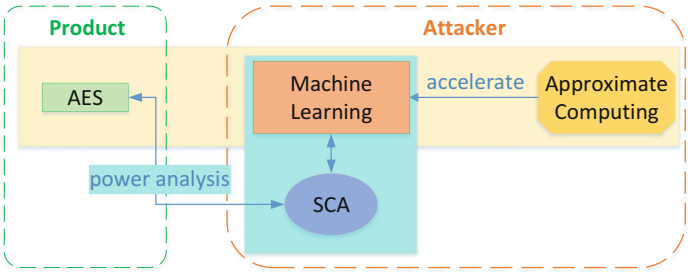


Fig. 14 An example of the application of machine learning to SCAs. The approximate computing is presented to accelerate/improve the efficiency of machine learning attacks

mask knowledge requirements at a cost of additional traces in the attack stage [56]. As shown in Fig. 14, approximate computing can be also applied for accelerating the machine learning algorithms for side channel attacks.

5.3 Hardware Trojans (HTs)

Resulting from the globalization of the semiconductor supply chain, the design and fabrication of ICs are now distributed worldwide. It brings great benefit to IC companies, which means a lower design cost and a shorter time-to-market window [47]. However, it also raises serious concern about IC trustworthiness triggered by the use of third-party vendors. As a result, it is becoming very difficult to ensure the integrity and authenticity of devices. A hardware trojan (HT) can be inserted into IC products at any untrusted phase of the IC production chain by third-party vendors or adversaries with an ulterior motive [79].

DL is a data-driven approach, where the goal is to ensure the learning algorithm is agnostic to the problem at hand; only the data changes [73]. This type of approach is often based on NN-type architectures with multiple hidden layers. With advances in training algorithms and computational power, it is now possible to train vast amounts of data leading to today's rapid advancements and adoption.

Hasegawa et al. [27] proposed a Trojan classification method for gate-level netlists using SVMs. By analyzing the netlists from the Trust-HUB benchmark suite [76], they identify several features strongly related to HTs. Trained by these features, their SVM approach results in high true positive rates, but relatively poor true negative rates when applied to the benchmark suite. Very recently, it was proposed to use DL in HT detection on gate-level netlists [27].

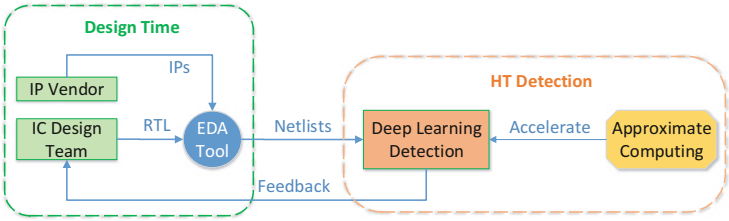


Fig. 15 The application of approximate computing to accelerate the detection of HTs

Figure 15 shows an approach using approximate computing to accelerate DL algorithms for HT detection. According to the effectiveness of the approximate circuit and algorithm development, the efficiency of the HT detection will be significantly improved.

5.4 Approximate Arithmetic Circuit for Logic Obfuscation

Logic obfuscation involves hiding important information, e.g., functionality and implementation, related to a circuit design by inserting additional logic components into the original design so that reverse engineering will not work without authorization. In order to execute its valid functionality to generate correct outputs, a secret key is implemented to the logic obfuscated circuit. If a wrong key is applied, the functionality will be incorrect and wrong outputs are generated by the obfuscated circuit. Logic obfuscation techniques have been utilized to protect IP and evaluate the trust of hardware [3]. However, an attacker can decipher the key by sensitizing the key values to the output or isolating the key-related gates since the logic obfuscation circuit, additionally added, can be removed from the original circuit [67].

To counter this, Fig. 16 shows a potential application of approximate arithmetic circuits in logic obfuscation. If the underlying design to be obfuscated is an approximate arithmetic circuit, logic obfuscation can be applied to the MSB or LSB

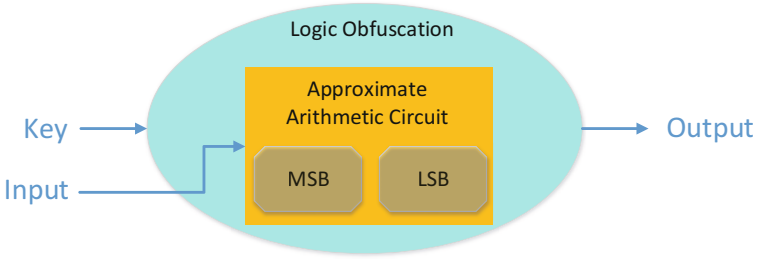


Fig. 16 A potential application of approximate arithmetic circuit to logic obfuscation

of an approximate arithmetic circuit that can only be used correctly by applying the key of the logic obfuscation circuit. Otherwise, the computation results will be too erroneous to use.

6 Conclusion

In this chapter, current approximate hardware approaches, in particular approximate arithmetic circuits, including adders, multipliers, and dividers as well as approximate software/algorithms are briefly reviewed. Two case studies, a security primitive based on approximate arithmetic circuits and a low-voltage approximate computing adder for authentication, are presented. Possible research directions for the application of approximate computing in hardware security scenarios, including SCAs, PUFs, and logic obfuscation techniques, are introduced and discussed. The goal of this chapter is to inspire future research on applying approximate computing techniques to hardware security applications.

Acknowledgements This work was partly supported by the National Natural Science Foundation of China (61871216 and 61771239), by Nature Science Foundation of Jiangsu Province (BK20151477), by Six Talent Peaks Project in Jiangsu Province (2018XYDXX-009), by the Institute for Information and Communications Technology Promotion (IITP) grant funded by the Korean government (MSIT) (No. 2016-0-00399, Study on secure key hiding technology for IoT devices [KeyHAS Project]), and by the Engineering and Physical Sciences Research Council (EPSRC) (EP/N508664/-CSIT2).

References

1. J. Albericio, P. Judd, T. Hetherington, T. Aamodt, N.E. Jerger, A. Moshovos, Cnvlutin: ineffectual-neuron-free deep neural network computing, in *Proceedings of ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA'16)* (IEEE, New York, 2016), pp. 1–13
2. H. Almurib, N. Kumar, F. Lombardi, Inexact designs for approximate low power addition by cell replacement, in *Proceedings of IEEE Design, Automation & Test in Europe Conference & Exhibition (DATE)* (2016), pp. 660–665
3. M. Arafin, M. Gao, G. Qu, VOLtA: voltage over-scaling based lightweight authentication for IoT applications, in *Proceedings of 22nd Asia and South Pacific Design Automation Conference (ASP-DAC)* (2017), pp. 336–341. https://doi.org/10.1109/ASPDAC.2017.7858345
4. G.T. Becker, On the pitfalls of using arbiter-PUFs as building blocks. IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. **34**(8), 1295–1307 (2015)
5. G.T. Becker, The gap between promise and reality: on the insecurity of XOR arbiter PUFs, in *Proceedings of International Workshop on Cryptographic Hardware and Embedded Systems (CHES'15)* (Springer, Berlin, 2015), pp. 535–555
6. V. Camus, J. Schlachter, C. Enz, A low-power carry cut-back approximate adder with fixed-point implementation and floating-point precision, in *Proceedings of 53rd Annual Design Automation Conference (DAC)* (2016), p. 127

7. T. Cao, W. Liu, C. Wang, X. Cui, F. Lombardi, Design of approximate redundant binary multipliers, in *Proceedings of IEEE/ACM International Symposium on Nanoscale Architectures (NANOARCH)* (2016), pp. 31–36. https://doi.org/10.1145/2950067.2950094
8. L. Chang, Cognitive data-centric systems, in *Proceedings of Great Lakes Symposium on VLSI (GLSVLSI)* (2017), pp 1–1. http://doi.acm.org/10.1145/3060403.3060491
9. S. Cheemalavagu, P. Korkmaz, K. Palem, B. Akgul, L. Chakrapani, A probabilistic CMOS switch and its realization by exploiting noise, in *Proceeding of IFIP International Conference on VLSI* (2005), pp. 535–541
10. W. Chen, J. Wilson, S. Tyree, K. Weinberger, Y. Chen, Compressing neural networks with the hashing trick, in *Proceeding of International Conference on Machine Learning* (2015), pp. 2285–2294
11. L. Chen, J. Han, W. Liu, F. Lombardi, Design of approximate unsigned integer non-restoring divider for inexact computing, in *Proceedings of ACM 25th Edition on Great Lakes Symposium on VLSI (GLSVLSI)* (2015), pp. 51–56. http://doi.acm.org/10.1145/2742060.2742063
12. L. Chen, J. Han, W. Liu, F. Lombardi, On the design of approximate restoring dividers for error-tolerant applications. IEEE Trans. Comput. **65**(8), 2522–2533 (2016). https://doi.org/10.1109/TC.2015.2494005
13. L. Chen, F. Lombardi, P. Montuschi, J. Han, W. Liu, Design of approximate high-radix dividers by inexact binary signed-digit addition, in *Proceedings of Great Lakes Symposium on VLSI (GLSVLSI)*, New York (2017), pp. 293–298. http://doi.acm.org/10.1145/3060403.3060404
14. V. Chippa, S. Chakradhar, K. Roy, A. Raghunathan, Analysis and characterization of inherent application resilience for approximate computing, in *Proceedings of 50th Annual Design Automation Conference (DAC)* (2013), p. 113
15. M. Courbariaux, Y. Bengio, J.-P. David, Binaryconnect: training deep neural networks with binary weights during propagations, in *Advances in Neural Information Processing Systems* (2015), pp. 3123–3131
16. M. Courbariaux, I. Hubara, D. Soudry, R. El-Yaniv, Y. Bengio, Binarized neural networks: training deep neural networks with weights and activations constrained to +1 or −1 (2016). http://arxiv.org/abs/1602.02830. 1602.02830
17. K. Du, P. Varman, K. Mohanram, High performance reliable variable latency carry select addition, in *Proceedings of Design, Automation & Test in Europe Conference & Exhibition (DATE)* (IEEE, New York, 2012), pp. 1257–1262
18. D. Esposito, D.D. Caro, E. Napoli, N. Petra, A.G.M. Strollo, Variable latency speculative Han-Carlson adder. IEEE Trans. Circuits Syst. Regul. Pap. **62**(5), 1353–1361 (2015). https://doi.org/10.1109/TCSI.2015.2403036
19. M. Gao, Q. Wang, M.T. Arafin, Y. Lyu, G. Qu, Approximate computing for low power and security in the internet of things. Computer **50**(6), 27–34 (2017). https://doi.org/10.1109/MC.2017.176
20. R. Gilmore, N. Hanley, M. O'Neill, Neural network based attack on a masked implementation of AES, in *Proceedings of IEEE International Symposium on Hardware Oriented Security and Trust (HOST)* (2015), pp. 106–111. https://doi.org/10.1109/HST.2015.7140247
21. I. Goiri, R. Bianchini, S. Nagarakatte, T. Nguyen, ApproxHadoop: bringing approximations to mapreduce frameworks, in *Proceedings of ACM SIGARCH Computer Architecture News*, vol. 43 (ACM, New York, 2015), pp. 383–397
22. C. Gu, N. Hanley, M. O'Neill, Improved reliability of FPGA-based PUF identification generator design. ACM Trans. Reconfig. Technol. Syst. **10**(3), 20:1–20:23 (2017). http://doi.acm.org/10.1145/3053681
23. J. Guajardo, S.S. Kumar, G.J. Schrijen, P. Tuyls FPGA intrinsic PUFs and their use for IP protection. Vienna (2007)
24. V. Gupta, D. Mohapatra, A. Raghunathan, K. Roy, Low-power digital signal processing using approximate adders. IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. **32**(1), 124–137 (2013)

25. J. Han, M. Orshansky, Approximate computing: an emerging paradigm for energy-efficient design, in *Proceedings of the 18th IEEE European Test Symposium (ETS)* (2013), pp. 1–6. https://doi.org/10.1109/ETS.2013.6569370
26. S. Han, J. Pool, J. Tran, W. Dally, Learning both weights and connections for efficient neural network, in *Proceedings of Advances in Neural Information Processing Systems* (2015), pp. 1135–1143
27. K. Hasegawa, M. Oya, M. Yanagisawa, N. Togawa, Hardware trojans classification for gate-level netlists based on machine learning, in *Proceedings of IEEE 22nd International Symposium on On-Line Testing and Robust System Design (IOLTS)* (2016), pp. 203–206. https://doi.org/10.1109/IOLTS.2016.7604700
28. K. Hasegawa, M. Yanagisawa, N. Togawa, Hardware trojans classification for gate-level netlists using multi-layer neural networks, in *Proceedings of IEEE 23rd International Symposium on On-Line Testing and Robust System Design (IOLTS)* (2017), pp. 227–232. https://doi.org/10.1109/IOLTS.2017.8046227
29. S. Hashemi, R. Bahar, S. Reda, Drum: a dynamic range unbiased multiplier for approximate applications, in *Proceedings of IEEE/ACM International Conference on Computer-Aided Design* (IEEE Press, New York, 2015), pp. 418–425
30. S. Hashemi, R.I. Bahar, S. Reda, A low-power dynamic divider for approximate applications, in *Proceedings of 53rd Annual Design Automation Conference (DAC)*, New York (2016), pp. 105:1–105:6. http://doi.acm.org/10.1145/2897937.2897965
31. A. Heuser, M. Zohner, Intelligent machine homicide, in *Proceeding of International Workshop on Constructive Side-Channel Analysis and Secure Design* (Springer, New York, 2012), pp. 249–264
32. I. Hong, D. Kirovski, G. Qu, M. Potkonjak, M.B. Srivastava, Power optimization of variable-voltage core-based systems. IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. **18**(12), 1702–1714 (1999). https://doi.org/10.1109/43.811318
33. G. Hospodar, B. Gierlichs, E. De Mulder, I. Verbauwhede, J. Vandewalle, Machine learning in side-channel analysis: a first study. J. Cryptogr. Eng. **1**(4), 293 (2011). https://doi.org/10.1007/s13389-011-0023-x
34. J. Hruska, Nvidia's CEO declares Moore's law dead (2017). https://www.extremetech.com/computing/256558-nvidias-ceo-declares-moores-law-dead
35. S. Hua, G. Qu, S.S. Bhattacharyya, An energy reduction technique for multimedia application with tolerance to deadline misses, in *Proceedings of Design Automation Conference* (IEEE, New York, 2003), pp. 131–136
36. S. Hua, G. Qu, S.S. Bhattacharyya, Probabilistic design of multimedia embedded systems. ACM Trans. Embed. Comput. Syst. **6**(3) (2007). http://doi.acm.org/10.1145/1275986.1275987
37. IEEE Standard for Floating-Point Arithmetic (2008). IEEE Std 754-2008, pp. 1–70. https://doi.org/10.1109/IEEESTD.2008.4610935
38. Y. Ikezaki, Y. Nozaki, M. Yoshikawa, Deep learning attack for physical unclonable function, in *2016 IEEE 5th Global Conference on Consumer Electronics* (2016), pp. 1–2. https://doi.org/10.1109/GCCE.2016.7800478
39. ITRS 2.0 home page (Last accessed 16 January 2018). http://www.itrs2.net/
40. H. Jiang, J. Han, F. Qiao, F. Lombardi, Approximate radix-8 booth multipliers for low-power and high-performance operation. IEEE Trans. Comput. **65**(8), 2638–2644 (2016)
41. H. Jiang, C. Liu, L. Liu, F. Lombardi, J. Han, A review, classification, and comparative evaluation of approximate arithmetic circuits. Emerg. Technol. Comput. Syst. **13**(4), 60:1–60:34 (2017). http://doi.acm.org/10.1145/3094124
42. N. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers, In-datacenter performance analysis of a tensor processing unit, in *Proceeding of 44th Annual International Symposium on Computer Architecture* (ACM, New York, 2017), pp. 1–12
43. A.B. Kahng, S. Kang, Accuracy-configurable adder for approximate arithmetic designs, in *Proceedings of 49th Annual Design Automation Conference (DAC)* (2012), pp. 820–825

44. Y. Kim, Y. Zhang, P. Li, An energy efficient approximate adder with carry skip for error resilient neuromorphic VLSI systems, in *Proceedings of the International Conference on Computer-Aided Design* (IEEE Press, New York, 2013), pp. 130–137
45. S.R. Kuang, J.P. Wang, C.Y. Guo, Modified booth multipliers with a regular partial product array. IEEE Trans. Circuits Syst. Express Briefs **56**(5), 404–408 (2009). https://doi.org/10.1109/TCSII.2009.2019334
46. P. Kulkarni, P. Gupta, M. Ercegovac, Trading accuracy for power with an underdesigned multiplier architecture, in *Proceedings of 24th IEEE International Conference on VLSI Design* (2011), pp. 346–351
47. A. Kulkarni, Y. Pino, T. Mohsenin, SVM-based real-time hardware trojan detection for many-core platform, in *Proceedings of 17th International Symposium on Quality Electronic Design (ISQED)* (2016), pp. 362–367. https://doi.org/10.1109/ISQED.2016.7479228
48. K. Kyaw, W. Goh, K. Yeo, Low-power high-speed multiplier for error-tolerant application, in *Proceedings of IEEE International Conference on Electron Devices and Solid-State Circuits (EDSSC)* (IEEE, New York, 2010), pp. 1–4
49. L. Lerman, G. Bontempi, O. Markowitch, A machine learning approach against a masked aes. J. Cryptogr. Eng. **5**(2), 123–139 (2015). https://doi.org/10.1007/s13389-014-0089-3
50. L. Lerman, R. Poussier, O. Markowitch, F.X. Standaert, Template attacks versus machine learning revisited and the curse of dimensionality in side-channel analysis: extended version. J. Cryptogr. Eng. (2017). https://doi.org/10.1007/s13389-017-0162-9
51. W. Liu, L. Chen, C. Wang, M. O'Neill, F. Lombardi, Design and analysis of inexact floating-point adders. IEEE Trans. Comput. **65**(1), 308–314 (2016). https://doi.org/10.1109/TC.2015.2417549
52. W. Liu, L. Qian, C. Wang, H. Jiang, J. Han, F. Lombardi, Design of approximate radix-4 booth multipliers for error-tolerant computing. IEEE Trans. Comput. **66**(8), 1435–1441 (2017). https://doi.org/10.1109/TC.2017.2672976
53. W. Liu, J. Xu, D. Wang, C. Wang, P. Montuschi, F. Lombardi, Design and evaluation of approximate logarithmic multipliers for low power error-tolerant applications. IEEE Trans. Circuits Syst. Regul. Pap. **65**(9), 2856–2868 (2018)
54. S.L. Lu, Speeding up processing with approximation circuits. Computer **37**(3), 67–73 (2004)
55. Y. Ma, N. Suda, Y. Cao, J. Seo, S. Vrudhula, Scalable and modularized RTL compilation of convolutional neural networks onto FPGA, in *Proceedings of 26th International Conference on Field Programmable Logic and Applications (FPL)* (IEEE, New York, 2016), pp. 1–8
56. H. Maghrebi, T. Portigliatti, E. Prouff, Breaking cryptographic implementations using deep learning techniques, in *Proceedings of International Conference on Security, Privacy, and Applied Cryptography Engineering* (Springer, Berlin, 2016), pp. 3–26
57. H. Mahdiani, A. Ahmadi, S. Fakhraie, C. Lucas, Bio-inspired imprecise computational blocks for efficient VLSI implementation of soft-computing applications. IEEE Trans. Circuits Syst. Regul. Pap. **57**(4), 850–862 (2010)
58. J. Mitchell, Computer multiplication and division using binary logarithms. IRE Trans. Electron. Comput. **EC-11**(4), 512–517 (1962)
59. D. Modha, R. Ananthanarayanan, S. Esser, A. Ndirango, A. Sherbondy, R. Singh, Cognitive computing. Commun. ACM **54**(8), 62–71 (2011)
60. D. Mohapatra, V. Chippa, A. Raghunathan, K. Roy, Design of voltage-scalable meta-functions for approximate computing, in *Proceedings of Design, Automation & Test in Europe Conference & Exhibition (DATE)* (IEEE, New York, 2011), pp. 1–6
61. A. Momeni, J. Han, P. Montuschi, F. Lombardi, Design and analysis of approximate compressors for multiplication. IEEE Trans. Comput. **64**(4), 984–994 (2015)
62. A. Nordrum, Popular internet of things forecast of 50 billion devices by 2020 is outdated (2016). https://spectrum.ieee.org/tech-talk/telecom/internet/popular-internet-of-things-forecast-of-50-/billion-devices-by-2020-is-outdated
63. S. Nowick, Design of a low-latency asynchronous adder using speculative completion. IEEE Proc. Comput. Digital Technol. **143**(5), 301–307 (1996)

64. B. Parhami, *Computer Arithmetic: Algorithms and Hardware Designs* (Oxford University Press, Oxford, 2000)
65. C. Peter, ARM's lead engineer discusses inexact processing EE Times (2013). https://www.eetimes.com/author.asp?section_id=36&doc_id=1318829
66. G. Qu, L. Yuan, Design things for the internet of things: an eda perspective, in *2014 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)* (2014), pp. 411–416. https://doi.org/10.1109/ICCAD.2014.7001384
67. J. Rajendran, Y. Pino, O. Sinanoglu, R. Karri, Security analysis of logic obfuscation, in *Proceedings of Design Automation Conference (DAC)* (2012), pp. 83–89. https://doi.org/10.1145/2228360.2228377
68. U. Rührmair, F. Sehnke, J. Sölter, G. Dror, S. Devadas, J. Schmidhuber, Modeling attacks on physical unclonable functions, in *Proceedings of 17th ACM Conference on Computer and Communications Security(CCS'10)*, Chicago (2010), pp. 237–249
69. U. Rührmair, J. Sölter, F. Sehnke, X. Xu, A. Mahmoud, V. Stoyanova, G. Dror, J. Schmidhuber, W. Burleson, S. Devadas, PUF modeling attacks on simulated and silicon data. IEEE Trans. Inf. Forensics Secur. **8**(11), 1876–1891 (2013)
70. M. Samadi, S. Mahlke, CPU-GPU collaboration for output quality monitoring, in *Proceedings of 1st Workshop on Approximate Computing Across the System Stack* (2014), pp. 1–3
71. M. Shafique, W. Ahmad, R. Hafiz, J. Henkel, A low latency generic accuracy configurable adder, in *Proceedings of 52nd Design Automation Conference (DAC)* (2015), pp 1–6. https://doi.org/10.1145/2744769.2744778
72. T. Simonite, (Last accessed 12 January 2018) This chip is hardwired to make mistakes but could help computers understand the world. https://www.technologyreview.com/s/601263/why-a-chip-thats-bad-at-math-can-help-computers/-tackle-harder-problems/
73. V. Sze, Y. Chen, T. Yang, J. Emer, Efficient processing of deep neural networks: a tutorial and survey. Proc. IEEE **105**(12), 2295–2329 (2017). https://doi.org/10.1109/JPROC.2017.2761740
74. Y. Tian, Q. Zhang, T. Wang, F. Yuan, Q. Xu, ApproxMA: approximate memory access for dynamic precision scaling, in *Proceedings of 25th Edition on Great Lakes Symposium on VLSI (GLSVLSI)* (2015), pp. 337–342. http://doi.acm.org/10.1145/2742060.2743759
75. J. Tobisch, G.T. Becker, On the scaling of machine learning attacks on PUFs with application to noise bifurcation, in *Proceedings of International Workshop on Radio Frequency Identification: Security and Privacy Issues* (Springer, Berlin, 2015), pp. 17–31
76. TrustHub (Last accessed 12 January 2018) Trusthub.org. http://trust-hub.org/
77. S. Venkataramani, S.T. Chakradhar, K. Roy, A. Raghunathan, Approximate computing and the quest for computing efficiency, in *Proceedings of 52nd Annual Design Automation Conference (DAC)* (2015), pp. 120:1–120:6. http://doi.acm.org/10.1145/2744769.2751163
78. A. Verma, P. Brisk, P. Ienne, Variable latency speculative addition: a new paradigm for arithmetic circuit design, in *Proceedings of the Conference on Design, Automation and Test in Europe (DATE)*, pp. 1250–1255 (2008)
79. X. Xie, Y. Sun, H. Chen, Y. Ding, Hardware trojans classification based on controllability and observability in gate-level netlist. IEICE Electronics Express (2017). https://doi.org/10.1587/elex.14.20170682
80. Q. Xu, T. Mytkowicz, N. Kim Approximate computing: a survey. IEEE Design Test **33**(1), 8–22 (2016). https://doi.org/10.1109/MDAT.2015.2505723
81. Z. Yang, A. Jain, J. Liang, J. Han, F. Lombardi, Approximate XOR/XNOR-based adders for inexact computing, in *Proceedings of 13th IEEE Conference on Nanotechnology (IEEE-NANO)* (IEEE, New York, 2013), pp 690–693
82. A. Yazdanbakhsh, D. Mahajan, H. Esmaeilzadeh, P. Lotfi-Kamran AxBench: a multiplatform benchmark suite for approximate computing. IEEE Design Test **34**(2), 60–68 (2017). https://doi.org/10.1109/MDAT.2016.2630270
83. R. Ye, T. Wang, F. Yuan, R. Kumar, Q. Xu, On reconfiguration-oriented approximate adder design and its application, in *Proceedings of IEEE International Conference on Computer-Aided Design* (IEEE Press, New York, 2013), pp. 48–54
84. W.C. Yeh, C.W. Jen, High-speed booth encoded parallel multiplier design. IEEE Trans. Comput. **49**(7), 692–701 (2000). https://doi.org/10.1109/12.863039

85. T. Yeh, P. Faloutsos, M. Ercegovac, S. Patel, G. Reinman, The art of deception: adaptive precision reduction for area efficient physics acceleration, in *Proceedings of 40th Annual IEEE/ACM International Symposium on Microarchitecture* (IEEE, New York, 2007), pp. 394–406
86. G. Zervakis, K. Tsoumanis, S. Xydis, N. Axelos, K. Pekmestzi, Approximate multiplier architectures through partial product perforation: power-area tradeoffs analysis, in *Proceedings of 25th Edition on Great Lakes Symposium on VLSI (GLSVLSI)* (ACM, New York, 2015), pp. 229–232
87. A. Zhou, A. Yao, Y. Guo, L. Xu, Y. Chen, Incremental network quantization: towards lossless CNNs with low-precision weights. CoRR abs/1702.03044 (2017). http://arxiv.org/abs/1702.03044. 1702.03044
88. N. Zhu, W.L. Goh, K.S. Yeo, An enhanced low-power high-speed adder for error-tolerant application, in *Proceedings of 12th IEEE International Symposium on Integrated Circuits* (IEEE, New York, 2009), pp. 69–72

Mathematical Optimizations for Deep Learning

Sam Green, Craig M. Vineyard, and Çetin Kaya Koç

Abstract Deep neural networks are often computationally expensive, during both the training stage and inference stage. Training is always expensive, because back-propagation requires high-precision floating-point multiplication and addition. However, various mathematical optimizations may be employed to reduce the computational cost of inference. Optimized inference is important for reducing power consumption and latency and for increasing throughput. This chapter introduces the central approaches for optimizing deep neural network inference: pruning "unnecessary" weights, quantizing weights and inputs, sharing weights between layer units, compressing weights before transferring from main memory, distilling large high-performance models into smaller models, and decomposing convolutional filters to reduce multiply and accumulate operations. In this chapter, using a unified notation, we provide a mathematical and algorithmic description of the aforementioned deep neural network inference optimization methods.

1 Introduction

Deep neural networks (DNNs) are increasingly being incorporated into safety-critical cyber-physical systems. For example, Advanced Driver Assistance Systems use DNNs for autonomous avoidance of road hazards. Modern DNN architectures

S. Green (✉)
University of California Santa Barbara, Santa Barbara, CA, USA
e-mail: sam.green@cs.ucsb.edu

C. M. Vineyard
Sandia National Laboratories, Albuquerque, NM, USA
e-mail: cmviney@sandia.gov

Ç. K. Koç
İstinye University, İstanbul, Turkey

Nanjing University of Aeronautics and Astronautics, Nanjing, China

University of California Santa Barbara, Santa Barbara, CA, USA
e-mail: cetinkoc@ucsb.edu

Ç. K. Koç (ed.), *Cyber-Physical Systems Security*,
https://doi.org/10.1007/978-3-319-98935-8_4

require billions of floating-point multiplications and additions (MACs) for inference of a single input. Without careful design, this results in high power consumption. Fossil fuel-powered vehicles, for example, can support high energy demands, but efficient, battery-powered systems cannot. Additionally, modern large DNNs have high latency, but low latency is required for real-time cyber-physical applications. This chapter provides a unified view of the leading methods for mathematically optimized deep learning inference. The intended audience of this chapter are hardware and software researchers, as well as developers interested in efficient DNN inference. Depending on the context, "efficiency" may imply low-power or low-latency.

To motivate the need for optimizations, it is helpful to consider first-order power and silicon area requirements for DNN inference. Table 1 provides a list of energy and die area required for various operator and operand sizes. Observe that a single 32-bit floating-point multiplication (denoted "32b FP Mult") requires 20× more power and 12× more area than 8-bit integer multiplication ("8b Mult"). Also observe that the power cost of a 32-bit DRAM read is more than 100× the cost of floating-point multiplication. For this reason, efficient DNN implementations should prioritize the minimization of off-chip DRAM access first, followed by reducing operand and operator sizes. Naturally these two priorities complement one another.

DNN optimizations are useful only during the inference operation. Training a DNN requires labeled datasets and uses the back-propagation algorithm. The back-propagation algorithm uses gradient descent to make many small adjustments to the neural network weights, and these small values must be calculated and stored using full-precision accumulation. Therefore the optimizations discussed in this chapter are not primarily aimed at making training more efficient, but they are intended to make inference more efficient.

To further emphasize the need for inference efficiency, consider the number of operations required to evaluate various modern DNNs, given in Table 2. This table provides a first-order estimate for MAC and memory costs for popular

Table 1 Energy and die area costs for various operations [1]

Operation	Energy (pJ)	Area (μm)
8b Add	0.03	36
16b Add	0.05	67
32b Add	0.1	137
16b FP Add	0.4	1360
32 FP Add	0.9	4184
8b Mult	0.2	282
32b Mult	3.1	3495
16b FP Mult	1.1	1640
32b FP Mult	3.7	7700
32b SRAM Read (8KB)	5	N/A
32b DRAM Read	640	N/A

Quantized operators and operands are preferred for low-power and low-resource applications. FP stands for floating point

Table 2 Number and cost of weights and MACs for popular deep neural network architectures

Metrics	LeNet 5	AlexNet	Overfeat fast	VGG 16	GoogLeNet v1	ResNet 50
Weights	60k	61M	146M	138M	7M	25.5M
Read cost (8b)	10 μJ	10 mJ	23 mJ	22 mJ	1 mJ	4 mJ
Read cost (32b)	38 μJ	39 mJ	93 mJ	88 mJ	4 mJ	16 mJ
MACs	341k	724M	2.8G	15.5G	1.43G	3.9G
MAC cost (8b)	0.1 μJ	167 μJ	644 μJ	3565 μJ	329 μJ	897 μJ
MAC cost (32b)	2 μJ	3 mJ	13 mJ	71 mJ	7 mJ	18 mJ

Cost estimates are based on Table 1 and from architecture statistics provided in [1]. Note that memory costs are typically higher than MAC costs

DNN architectures. Power estimates assume 32-bit floating-point arithmetic and are derived from Table 1. MAC costs capture the power requirement for each network to perform the necessary operations for providing a single inference. The memory cost is best case and assumes weights are read from DRAM only once per inference; actual memory costs will be higher if intermediate results must be transferred back to DRAM during inference of the network. In Table 2 note that even though the number of MACs is much greater than the number of weights, the high DRAM read cost results in the power consumed between the two to be roughly equivalent.

The process of DNN training may be thought of as an exploration over a parameter space to find values which will solve an inference task. As will be expanded on, the weights found using standard training methods result in DNNs which are over-parameterized, which means they have redundancy. When the DNN performs satisfactorily during cross validation, back-propagation is no longer needed, and optimizations may be applied to decrease parameter redundancy. The goal of mathematical optimizations for deep learning is to find the most compact network which performs satisfactorily at its assigned real-world inference tasks.

DNN architectures are composed of various layer types: convolutional, fully connected, dropout, pooling, and others. Each layer type was developed to solve a particular weakness, and each classification problem is best solved by a different architecture, or combination of layers. Convolutional and fully connected layers represent the greatest computational expense in DNN inference, and optimizing these layer types is the focus of this chapter. Both convolutional and fully connected layers require repeated multiplication and addition, but they typically use different algorithmic steps. Adapting notation of [2], we represent an L-layer DNN as $\langle \mathrm{I}, \mathrm{W}, \mathrm{O} \rangle$, where:

- $\mathrm{I}_l \in \mathbb{R}^{c_{in} \times x \times y}$ and $\mathrm{W}_l \in \mathbb{R}^{c_{in} \times w \times h \times c_{out}}$ are layer l's input tensors and weight tensors, respectively. c_{in} represents the number of **input channels** and c_{out} represents the number of **output channels**.[1] x and y are the width and height of each input channel, and w and h are the width and height of each filter.

[1] Also called input filter maps (ifmaps) and output filter maps (ofmaps) in literature.

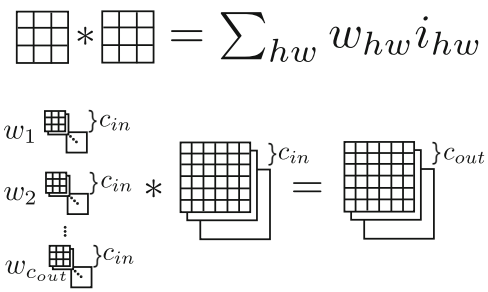


Fig. 1 Convolutional layers convolve a weight filter with an input. Filters are usually 5×5, 3×3, or 1×1. Each step of the convolution involves multiplying and accumulating elements of the weight filter with a **receptive field** of the input. The top illustration represents the basic convolution operation (∗). The lower illustration represents c_{out}, c_{in}-channel filters which are convolved with a c_{in}-channel input tensor, which results in an c_{out}-channel output tensor

- $O_l \in \{*, \cdot, \text{other}\}$ specifies whether the layer's operation type is convolution (∗), fully connected (·), or some other less computationally expensive type.

Convolutional layers convolve a $\mathbb{R}^{c_{in} \times w \times h \times c_{out}}$ weight filter tensor with a $\mathbb{R}^{c_{in} \times x \times y}$ input tensor, where (w,x) and (h,y) represent the widths and heights of the two respective tensors and may be different sizes and c_{in} and c_{out} represent the number of input and output channels. In particular the (w, h) for weight filters are often smaller than the (x, y) for inputs. c is the number of channels in the given layer; this value is equal for both the weight filter tensor and input tensor. As illustrated in Fig. 1 (top), each step in the convolution requires a sum of products between elements of the weight filter and elements of the receptive field of the input filter.

Note that what is shown in Fig. 1 (top) only depicts convolution of a single channel. If there are multiple channels, then the summation is also over all channels. Figure 1 (bottom) shows a higher-level view, where each c_{in}-channel weight filter is convolved with the c_{in}-channel input tensor. When multiple channels are included in the convolution, each output of the convolution becomes the triple sum across the channels. The number of weight filters in a layer equals the number of channels in the output tensor: if there are c_{out} weight filters, there will be c_{out} channels in the output tensor.

Computation for fully connected layers requires a single matrix-vector product. The input tensor $I_l \in \mathbb{R}^{c_{in} \times x \times y}$ is flattened to a vector $\in \mathbb{R}^{c_{in} \cdot x \cdot y}$. The weight tensor is denoted $W \in \mathbb{R}^{w \times h}$, where $w = c_{in} \cdot x \cdot y$ (from the input tensor dimensions) and h is equal to the number of desired output units from the fully connected layer. An illustration of a fully connected layer is given in Fig. 2.

After a weight filter W is convolved with an input I in a convolutional layer, or the matrix-vector product between weights and layer inputs is produced for a fully connected layer, the resulting matrix of vector entries is typically passed through a nonlinearity function $\sigma : \mathbb{R} \rightarrow \mathbb{R}$. A commonly used nonlinearity is the rectified

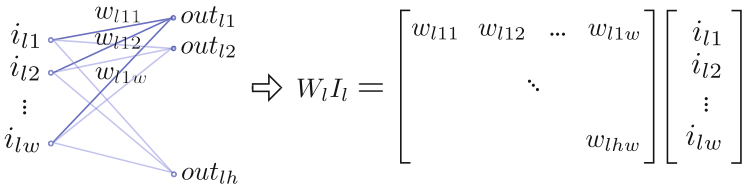


Fig. 2 Fully connected layers flatten the input tensor into a vector and multiply by a weight matrix with the same number of columns as the vector and as many rows as desired

linear unit (ReLU), which is defined as:

$$\sigma_{\text{ReLU}}(x) = \begin{cases} x & \text{if } x \geq 0, \\ 0 & \text{else.} \end{cases} \tag{1}$$

But more extreme nonlinearities exist, such as the binarized activation function which outputs only two values, −1 and 1:

$$\sigma_b(x) = \begin{cases} 1 & \text{if } x \geq 0, \\ -1 & \text{else.} \end{cases} \tag{2}$$

The choice of nonlinearity function influences the performance and computational cost of inference. Specifically, using the binarized activation function can lead to the elimination of floating-point and fixed-point arithmetic during inference, as detailed in Sect. 3.2.

Both convolutional and fully connected layers require many memory access and MAC operations, but a variety of numerical optimizations may be applied to DNN inference. Some optimizations reduce power and some optimizations reduce both power and latency. Furthermore, it is possible to optimize a DNN and maintain classification accuracy, but there also exist extreme optimization methods which result in unavoidable accuracy loss. Depending on the application, decreased accuracy may be worth the reduction in power and latency.

The remainder of this chapter provides an introduction to the common approaches of DNN mathematical optimization. The approaches are grouped by five primary strategies:

- **Pruning**: reduces the number of weights, which, in turn, reduces the total number of MAC operations, amount of traffic required to transfer weights, and storage requirements. This method applies to fully connected and convolutional layers.
- **Quantization**: lowers the number of bits of precision representing neural network inputs, weights, or activations, which lowers both memory requirements and silicon required for processing elements. This method applies to fully connected and convolutional layers.

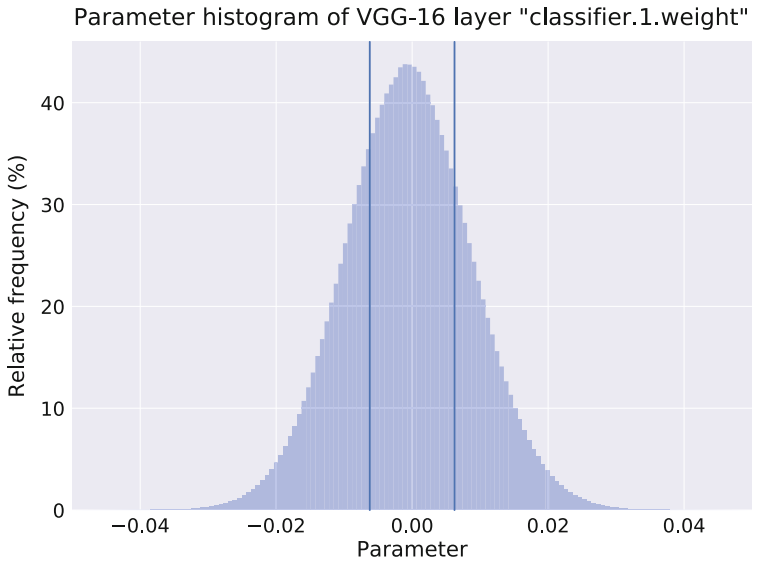


Fig. 3 Histogram of weights of the first fully connected layer in VGG-16. The name "classifier.1.weight" corresponds to the VGG-16 implementation found in torchvision [4]. The two vertical lines correspond to thresholds of values smaller than the 50th-percentile. These values may be pruned (permanently set to zero) and the remaining values fine-tuned with no loss in accuracy [3]. The same procedure may be applied to all other layers in the network

- **Weight Sharing and Compression**: forces weights to share values, thus decreasing memory storage and traffic. This method applies to fully connected and convolutional layers.
- **Model Distillation**: the training of a smaller network to mimic the behavior of larger network, reducing the number of weights and lowering latency. This method applies to fully connected and convolutional layers.
- **Filter Decomposition**: modifies convolutional filter designs such that the number of weights and latency is reduced. This method only applies to convolutional layers.

2 Pruning

Pruning applies to fully connected and convolutional layers and eliminates each layer's smallest weights, which has the consequence of reducing the number of MAC operations, the amount of traffic required to transfer weights, and storage requirements. The typical procedure is to train the network until the desired accuracy is reached and then to prune the smallest pth-percentile of weights by setting them to zero. Pruning is followed by fine-tuning the remaining weights, which can be accomplished using the same dataset as used during initial training.

In [3], the authors report 9× and 13× reduction in weights for AlexNet and VGG-16 with no impact on test accuracy. A histogram of the normalized frequency of weights is given in Fig. 3, where the smallest 50th-percentile is delineated with two vertical lines. In practice, one would pick the percentile threshold for each layer heuristically, that is, the percentile threshold would be a hyperparameter for each layer. This process is represented in Algorithm 1.

Algorithm 1 Pruning

Require: L-layer DNN $\langle \mathrm{I}, \mathrm{W}, \mathrm{O}, \mathrm{P} \rangle$, where I_l and W_l are layer l's input tensors and parameter tensors respectively, and O_l specifies whether the layer's type is convolutional, fully-connected (or some other type), and P is the pruning percentile for each layer.

Ensure: Pruned and fine-tuned network weights W.

1. Initial training:

Perform standard training of DNN until satisfactory performance is achieved.

2. Pruning:

For each layer l in $\langle \mathrm{I}, \mathrm{W}, \mathrm{O}, \mathrm{P} \rangle$, eliminate weights in W_l which are less than layer l's p^{th} percentile, where $p = \mathrm{P}_l$.

3. Fine-tuning:

Perform standard (re)training of remaining weights W, until maximum performance is achieved.

After pruning, the resulting DNN will be sparse, with many weights set to zero. Standard architectures, like GPUs, are currently not designed to take advantage of sparsity and will perform multiplication regardless if one of the operands is zero. In order to benefit from pruning, the architecture must be designed in such a way as to take advantage of sparsity. This will add edge cases to standard logic design. For example, consider a product summation tree, which can parallelize MAC operations. Even if the tree is designed to ignore products with a zero operand, it must still take into account that the zero product must be passed to the next tree level at the appropriate time. Recently, architectures for handling sparse dataflows have been developed. One such architecture reduces the amount of "wasted" logic required for ignoring zero products by only passing non-zero products to processing elements downstream [5].

3 Quantization

Before 2015, most DNNs were trained using 32-bit floating-point arithmetic. In this section we summarize approaches for using reduced precision, or quantized arithmetic, for DNN inference. Quantization reduces the amount of weight data that must be transferred from DRAM to processing elements. Additionally, quantized arithmetic is less expensive in terms of power and silicon area than full-precision arithmetic. Quantization may be applied to weights, activations, or both weights and activations. We emphasize that quantization techniques using <16 bits

currently only provide efficiency benefits during inference, because back-propagation requires accumulation of small values, and therefore $\geq$16 bits.

It appears that 8-bit or 16-bit quantization is adequate for most DNN inference tasks. For example, Google's DNN accelerator, the Tensor Processing Unit (TPU), exclusively uses 8-bit or 16-bit integer arithmetic [6]. The TPU (and the successor TPUv2) is becoming a critical component of Google's computing ecosystem. Additionally, NVIDIA's Pascal architecture was designed to support 16-bit floating-point and 8-bit integer arithmetic.

In this section we focus on extreme quantization methods which binarize weights and activations. Binarization usually has a large negative impact on performance, but we present techniques in Sects. 3.1 and 3.2 which reduce the impact.

Note that in this section, we will sometimes use a unified notation which applies to both convolutional and fully connected layers. In a convolutional layer, a c-channel weight filter $\mathrm{W} \in \mathbb{R}^{c \times w \times h}$ is convolved with an input $\mathrm{I} \in \mathbb{R}^{c \times w \times h}$. Convolution is performed by $\mathrm{W} * \mathrm{I}$. At a specific receptive field, the core operation may be interpreted as the inner products between vectors. In this section, we sometimes use the notation $\mathrm{W}^\top \mathrm{I}$ to denote the convolution of a filter with a specific receptive field. Simultaneously, the $\mathrm{W}^\top \mathrm{I}$ notation captures the partial calculation of a fully connected layer.

3.1 Binary Weights

In 2015, BinaryConnect [7] was an early DNN quantization method and exemplifies the field's approach to quantization. During inference, BinaryConnect quantizes full-precision DNN weights W to $\{-1, 1\}$, using the sign function:

$$w^{(b)} = \begin{cases} +1 & \text{if } w \geq 0, \\ -1 & \text{else.} \end{cases} \tag{3}$$

Equation (3) discards real-valued information, but, in doing so, it also eliminates the need for floating-point multiplication during inference. Instead, signed floating-point addition may be used for unit activation input calculations. During back-propagation, the error caused by quantization is used to update the real-valued Ws. After training is complete, full-precision weights and arithmetic are no longer required and may thereafter be discarded. From a hardware perspective, memory overhead is 32$\times$ less when using BinaryConnect-derived weights. However, this technique has an accuracy cost. When using the AlexNet DNN architecture, BinaryConnect achieves 61% top-5 accuracy on ImageNet, compared to 80.2% accuracy when using AlexNet with 32-bit full-precision accuracy [2].

In Algorithm 2 we outline the steps of BinaryConnect. Note here that we separate the bias terms from W, where normally it is included in that tensor for notation convenience. The reason here is that the bias is always added, even with

full-precision arithmetic, so there is no benefit to quantize it. Also note the `clip` function in Algorithm 2 limits the full-precision weights to between [−1, 1].

Algorithm 2 BinaryConnect [7]

Require: Inputs I, targets y, previous full-precision weights W, biases b, learning rate η, and objective function J.
Ensure: Updated $\{-1, 1\}$-valued weights $\mathrm{W}^{(b)}$ and real-valued bias b.

1. Forward propagation:
$\mathrm{A}_0 = \mathrm{I}$
for $l = 1$ to L
  for k^{th} filter in l^{th} layer
    $\mathrm{W}_{lk}^{(b)} \leftarrow \texttt{binarize}(\mathrm{W}_{lk})$ using Eq. (3)
    $\mathrm{A}_{lk} \leftarrow \mathrm{W}_{l}^{(b)} * \mathrm{A}_{(l-1)k} + b_{lk}$
2. Backward propagation:
Initialize output layer's activation gradient $\frac{\partial J}{\partial A_L}$ using y, A_L, and J
for $l = L$ to 2
  for k^{th} filter in l^{th} layer
    Compute $\frac{\partial J}{\partial A_{(l-1)k}}$ knowing $\frac{\partial J}{\partial A_{lk}}$ and $\mathrm{W}_{lk}^{(b)}$
3. Update weights:
Compute $\frac{\partial J}{\partial \mathrm{W}_{lk}^{(b)}}$ and $\frac{\partial J}{\partial b_{lk}}$, knowing $\frac{\partial J}{\partial A_{lk}}$ and $A_{(l-1)k}$
$\mathrm{W} \leftarrow \texttt{clip}(\mathrm{W} - \eta \frac{\partial J}{\partial \mathrm{W}})$
$b \leftarrow b - \eta \frac{\partial J}{\partial b}$

Not made explicit in Algorithm 2 is how the gradient signal passes through the binarization function given in Eq. (3). This is required for calculation of $\partial J / \partial \mathrm{W}_{lk}^{(b)}$. We cannot merely take the derivative of the binarization function, because it is 0 everywhere except at $\mathrm{W} = 0$, where the function is discontinuous. To handle this, the authors used a variant of the **Straight-Through Estimator** (STE) during back-propagation [8]. The modified STE is defined as:

$$\texttt{STE}(\text{pre-binarized value}) = \begin{cases} 0 & \text{if } x < -1, \\ 1 & \text{if } -1 \geq \text{pre-binarized value} \leq 1, \\ 0 & \text{if } x > 1. \end{cases} \tag{4}$$

During back-propagation, instead of flowing through the binarization function, the incoming gradient signal is multiplied by the value of `STE`, which is evaluated at the pre-binarized weight value (or pre-activation value, when using XNOR-Net, discussed below). Clipping caused by multiplying by the `STE` has the effect of canceling the gradient when the pre-binarized value is too large.

To summarize BinaryConnect, we take the sign of the real-valued weights during inference. During back-propagation, the errors caused by binarization may be very small (with significant changes accumulating over many inputs), and we track those small changes in full-precision versions of the weights. After training is complete, the full-precision weights may be discarded, only keeping their sign information.

XNOR-Net [2] introduced a method which is almost identical to BinaryConnect, but it performs binarization in way which achieves higher accuracy. As with BinaryConnect, weights are binarized during inference, but then they are also scaled by a factor which attempts to compensate for the binarization. Specifically, XNOR-Net introduced the following approximation for the inner product[2]:

$$\mathrm{W}^\top \mathrm{I} \approx \alpha \mathrm{W}^{(b)\top} \mathrm{I}, \tag{5}$$

where $\mathrm{W}^{(b)}$ is the binarized version of W using Eq. (3). This notation is slightly different than that used in Algorithm 2, where we are able to binarize the entire W tensor at once. But with XNOR-Net, each filter in each convolutional layer requires a separate α. To keep the notation simple, separate filters are not denoted.

To find the optimal scaling factor α, we solve the following optimization problem:

$$\begin{aligned} J(\alpha) &= \left\| \mathrm{W} - \alpha \mathrm{W}^{(b)} \right\|^2, \\ \alpha^* &= \arg\min_{\alpha} J(\alpha). \end{aligned} \tag{6}$$

That is, we are seeking an α which minimizes the distance between W and $\alpha \mathrm{W}^{(b)}$. For intuition, consider a scalar w and its binarized version $w^{(b)}$; in this case $\alpha = w/w^{(b)}$ perfectly minimizes the distance between w and $w^{(b)}$. Expanding the norm in Eq. (6) gives:

$$J(\alpha) = \alpha^2 \mathrm{W}^{(b)\top} \mathrm{W}^{(b)} - 2\alpha \mathrm{W}^\top \mathrm{W}^{(b)} + \mathrm{W}^\top \mathrm{W}. \tag{7}$$

We now take the derivative of $J(\alpha)$ with respect to α, set it to zero, and solve for α:

$$\frac{dJ(\alpha)}{d\alpha} = 2\alpha \mathrm{W}^{(b)\top} \mathrm{W}^{(b)} - 2\mathrm{W}^\top \mathrm{W}^{(b)}. \tag{8}$$

Let $n = \mathrm{W}^{(b)\top} \mathrm{W}^{(b)}$, which is also equal to the number of weights in the binarized filter. Substituting n into Eq. (8) and solving for α gives α^*:

$$\alpha^* = \frac{\mathrm{W}^{(b)\top} \mathrm{W}^{(b)}}{n} = \frac{\mathrm{W}^{(b)\top} \mathrm{sign}(\mathrm{W})}{n} = \frac{\sum |\mathrm{W}|}{n}. \tag{9}$$

New α^*s must be calculated every time W changes, i.e., each time back-propagation is used to update the weights, but, after the training is completed, α^* may be saved for use during inference.

[2]Note that we consider W and I to be flattened.

Table 3 The XNOR operation captures the behavior of signed multiplication

Signed multiplication		
Inputs		Output
I_i	W_i	$I_i \times W_i$
−1	−1	1
−1	1	−1
1	−1	−1
1	1	1

XNOR "multiplication"		
Inputs		Output
I_i	W_i	$\overline{I_i \oplus W_i}$
0	0	1
0	1	0
1	0	0
1	1	1

Using the weight binarization methods above, we may eliminate most multiplications from inference,[3] and instead we only need signed addition. If we assume 32-bit multiplication and addition, this results in 32× power reduction for weight transfer from DRAM and ∼3× power reduction for arithmetic. When using the AlexNet DNN architecture, XNOR-Net (binary weights, full-precision activations) achieves 79.4% top-5 accuracy on ImageNet, compared to 80.2% accuracy when using AlexNet with 32-bit full-precision accuracy [2]. We next consider operator optimizations which become available when both weights *and* inputs are binarized.

3.2 Binary Weights and Activations

If weights and activations are binarized, then we are able to eliminate almost all floating-point (and fixed-point) calculations, resulting in extreme energy savings. Specifically, when weights and inputs are binarized, the XNOR operation[4] may be used to calculate inner products during inference [9]. The XNOR logic truth table is given on the right in Table 3. The left-hand side provides the truth table for signed multiplication between scalar values $I_i \in I$ and $W_i \in W$. Note that by mapping −1 to 0, the two tables give identical output.

XNOR logic is simple and efficient to implement in hardware and may be used as the multiplication operator for the calculation of inner products during inference. To use the XNOR "product" between I and W for the input into a unit's nonlinearity function, we first map all −1s to 0s and then calculate the XNOR values for both vectors. The Hamming weight[5] (HW) of the XNOR vector result is then compared to $\#bits/2$, where $\#bits$ is the size of W and I. If the Hamming weight is greater than or equal to $\#bits/2$, then output 1, otherwise output 0. Note that after the initial mapping of −1 to 0, we no longer need to map back to −1 during the remainder of the inference procedure.

BinaryNet [9] operates similarly to BinaryConnect, with the addition that activations are also binarized. When using BinaryNet, the activation inputs are summed, as

[3] Multiplication by α is still necessary when using the weight binarization technique in XNOR-Net.

[4] Not to be confused with XNOR-Net [2]. Here we are referring to the exclusive-NOR operation.

[5] Hamming weight is defined as the number of 1s in a vector.

with BinaryConnect, and then the resulting sum is converted to [−1, 1] using the sign function. This optimization eliminates all full-precision calculations and replaces them with signed integer calculations. As with BinaryConnect, BinaryNet requires full-precision gradient updates during training, and during back-propagation the `STE` function (Eq. (4)) is used for both the activation and weights. BinaryNet achieves 50.42% top-5 accuracy on AlexNet, compared to 80.2% accuracy when using the same DNN topology and 32-bit full-precision accuracy [2].

XNOR-Net also has a version which binarizes both weights and activations. Similar to XNOR-Net's weight-only binarization presented above, there is a scaling factor α which may (optionally) be used to reduce the error between full-precision and binarized dot products:

$$\begin{aligned} J(\alpha) &= \left\| \mathrm{I}^{\top}\mathrm{W} - \alpha \mathrm{I}^{(b)\top}\mathrm{W}^{(b)} \right\|^2, \\ \alpha^* &= \arg\min_{\alpha} J(\alpha). \end{aligned} \tag{10}$$

This is solved in the same manner as Eq. (6), giving:

$$\alpha^* = \frac{\sum |\mathrm{I}^{(b)\top}\mathrm{W}^{(b)}|}{n} = \frac{\sum |\mathrm{I}||\mathrm{W}|}{n}. \tag{11}$$

Note that a separate scaling factor α^* must be solved for each receptive field and weight filter combination *both during training and when using the neural network after training*. This high computational overhead limits the use of vanilla XNOR-Net. Fortunately, in practice, the authors of BinaryNet found that the scaling factor for binarized weights was much more important than the scaling factor for binarized inputs and may therefore be ignored. We summarize the weight-scaled version of XNOR-Net with the following algorithm:

Similar to the calculation of $\partial J/\partial \mathrm{W}_{lk}^{(b)}$ in Algorithm 2, both partial derivatives $\partial J/\partial \mathrm{W}_{lk}^{(b)}$ and $\partial J/\partial \mathrm{A}_{lk}^{(b)}$ in Algorithm 3 are multiplied by the `STE` function in Eq. (4), where the inputs to `STE` are the real-valued weight and activation, respectively.

XNOR-Net using binarized inputs and weights achieves 69.2% accuracy on AlexNet, compared to BinaryNet's 50.42%, and full-precision accuracy of 80.2%. The XNOR-Net and BinaryNet papers introduce other training tips for improved performance. The aggregate contributions of the performance techniques introduced in XNOR-Net likely account for its significant gain over BinaryNet.

4 Weight Sharing and Compression

Top-performing neural networks use millions of weights which are typically transferred from DRAM to processing elements for inference (see Table 2). When these weights are transferred, DRAM energy cost can surpass arithmetic cost for

Algorithm 3 (Weight-scaled) XNOR-Net [9]

Require: Inputs I, targets y, previous full-precision weights W, biases $\boldsymbol{b}$, learning rate η, and objective function J.

Ensure: Updated $\{-1, 1\}$-valued weights $\mathrm{W}^{(b)}$, weight scaling factors $\boldsymbol{\alpha}$, and real-valued bias $\boldsymbol{b}$.

1. Forward propagation:

$\mathrm{A}_0 = \texttt{binarize}(\mathrm{I}_0)$

for $l = 1$ to L

  for k^{th} filter in l^{th} layer

    $\alpha_{lk} = \frac{1}{n}||\mathrm{W}_{lk}||_{\ell 1}$

    $\mathrm{W}_{lk}^{(b)} \leftarrow \texttt{binarize}(\mathrm{W}_{lk})$ using Eq. (3)

    $\mathrm{A}_{lk}^{(b)} \leftarrow \texttt{binarize}\big((\alpha_{lk}\mathrm{W}_{lk}^{(b)}) * \mathrm{A}_{(l-1)k}^{(b)} + b_{lk}\big)$ using Eq. (3)

2. Backward propagation:

Initialize output layer's activation gradient $\frac{\partial J}{\partial A_L}$ using y, A_L, and J

for $l = L$ to 2

  for k^{th} filter in l^{th} layer

    Compute $\frac{\partial J}{\partial A_{(l-1)k}^{(b)}}$ knowing $\frac{\partial J}{\partial A_{lk}^{(b)}}$ and W_{lk}

3. Update weights:

Compute $\frac{\partial J}{\partial \mathrm{W}_{lk}^{(b)}}$ and $\frac{\partial J}{\partial b_{lk}}$, knowing $\frac{\partial J}{\partial A_{lk}^{(b)}}$ and $A_{(l-1)k}$

$\mathrm{W} \leftarrow \texttt{clip}(\mathrm{W} - \eta \frac{\partial J}{\partial \mathrm{W}^{(b)}})$

$b \leftarrow b - \eta \frac{\partial J}{\partial b}$

performing a single inference. Weight sharing clusters weights into shared values and is applied after the network has reached peak performance. Once weights have been clustered, compression may be used to transmit cluster indices instead of full-precision values. Weight sharing coupled with compression is a method to retain the high performance typically provided by large full-precision neural networks while simultaneously reducing the amount of data sent over DRAM [3].

4.1 Weight Sharing

To apply weight sharing, first, the DNN is trained to maximum performance using standard training methods. After training, each layer's weights are grouped into clusters, where the number of weights in a layer is much greater than the number of clusters. After assigning weights to clusters, the network goes through a retraining phase.

For example, consider Fig. 4 which illustrates a 4×4 channel from some weight filter in W. Assume that the filter is part of a trained network. To apply weight sharing, we use k-means clustering [3], which assigns the weights $w \in \mathrm{W}$ to m cluster assignments $C^\star = \{c_1, c_2, \ldots, c_m\}$, such that the within-cluster sum of squares is minimized:

$$C^\star = \arg\min_{C} \sum_{i=1}^{m} \sum_{w \in c_i} |w - c_i|^2. \tag{12}$$

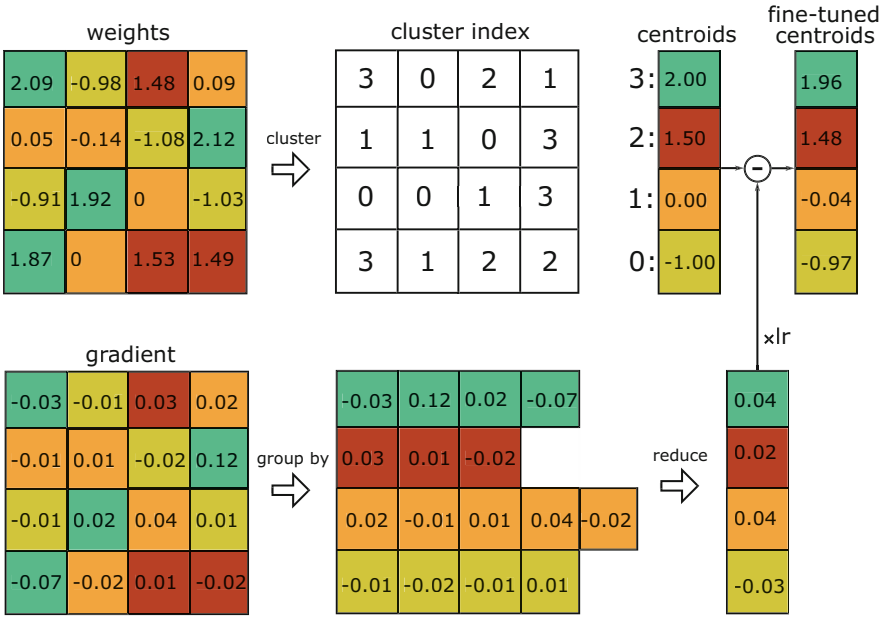


Fig. 4 After training, 16 weights have been clustered into 4 centroids. From that point on, clustered weights are equal to their centroid. Partial derivatives are calculated with respect to the weight values, as usual, but the gradients are accumulated and subtracted from the centroids [3]

After assignment to clusters, we calculate the centroids $\tilde{w}_i$ of each cluster c_i by taking the average value of each cluster:

$$\tilde{w}_i = \frac{1}{|c_i|} \sum_{w \in c_i} w. \tag{13}$$

In Fig. 4, $m = 4$, and the top portion of the plot illustrates 16 weights and their associated clusters and centroids.

After clustering, weights in the original filter are replaced by their centroid value (this is represented by the shading in Fig. 4). Next, the clustered weights are fine-tuned by reusing the original training data. The key difference between standard training and the fine-tuning phase is how the weights are updated during gradient descent (GD). In GD each weight is moved a small amount in the direction which will improve an objective function, e.g.:

$$\mathrm{W}_{l,w} = \mathrm{W}_{l,w} - \eta \frac{\partial J(\mathrm{W})}{\partial \mathrm{W}_{l,w}}. \tag{14}$$

However, after clustering, we apply GD to the centroid value of each weight cluster. For example, suppose the centroid $\tilde{w}_i$ of weight cluster c_i is to be updated using

GD. To update centroid $\tilde{w}_i$, we use the sum of partial derivatives with respect to weights assigned to that cluster:

$$\tilde{w}_i = \tilde{w}_i - \eta \sum_{w \in c_i} \frac{\partial J(\mathrm{W})}{\partial w}. \tag{15}$$

The lower portion and the subtraction in Fig. 4 illustrate the gradient descent step of back-propagation when using clustering. After the fine-tuned centroids have been calculated, they will replace the previous weight values in each cluster. The update given in Eq. (15) is repeated until maximum performance is attained.

Algorithm 4 Weight sharing

Require: Inputs I, previously *trained* full-precision weights W, number of clusters m, learning rate η, objective function J.
Ensure: Clustered and fine-tuned weights W
1. Cluster assignment:
for $l = 1$ to L
for k^{th} filter in l^{th} layer
Assign weights in filter k to m clusters using Eq. (12):
$C^\star \leftarrow \texttt{knn}(\mathrm{W}_{lk}, m)$
Replace weights in each cluster with centroid value using Eq. (13):
$\mathrm{W}_{lk} \leftarrow \texttt{centroid}(\mathrm{W}_{lk}, C^\star)$
2. Inference:
Perform standard inference using centroid-mapped weights.
3. Fine-tuning:
Calculate standard partial derivatives with respect to weights $\frac{\partial J(\mathrm{W})}{\partial w}$.
Update centroid values by summing partial derivatives in each cluster and using gradient decent: $\tilde{w}_i = \tilde{w}_i - \eta \sum_{w \in c_i} \frac{\partial J(\mathrm{W})}{\partial w}$
Replace weights in each cluster with updated centroid values.
4. Optionally repeat:
Repeat steps 2 and 3 until objective function is optimized.

The steps for weight sharing are provided in Algorithm 4. The algorithm is written from the perspective of CNNs, but adapting it for other DNN designs only requires clustering the appropriate values. For example, the values in fully connected layer could be clustered.

After weight values have been clustered and fine-tuned, there is an opportunity to decrease the storage and traffic requirements for loading the DNN weights from memory to an accelerator. This process is detailed in the following subsection.

4.2 Compression

Weight sharing reduces the amount of data transmitted over DRAM by intentionally creating redundancy in the form of a cluster index. For example, in Fig. 4 we see

that 16 original values are represented by four cluster values. Redundancy created by weight sharing is exploitable with compression methods [3].

If a network uses b bits of precision, then a full-precision network with n weights requires nb bits of transmission. After weight sharing, only a single full-precision value (the centroid) must be transmitted for each cluster; this results in mb bits. The indices for m clusters are represented with $\log_2(m)$ bits; therefore transmitting n indices requires $n\log_2(m)$ bits. In general, n weights clustered into m clusters compress the weights by a factor of:

$$\frac{nb}{n\log_2(m) + mb}. \tag{16}$$

For example, referring to Fig. 4, and assuming 32-bit floating-point weights, we see that $nb = 16 \cdot 32$ and $n\log_2(m) + mb = 16 \cdot 2 + 4 \cdot 32$. Therefore, by using weight sharing and compression, we reduce the traffic by a factor of 352.

5 Model Distillation

Large neural networks have a tendency to generalize better than smaller networks. Similarly, ensemble methods combine the predictions of multiple algorithms, e.g., DNNs, random forests, SVMs, logistic regression, etc., and almost always outperform the predictions from an individual algorithm. Both large networks and ensemble methods are attractive from an accuracy perspective, but many applications cannot support the time or energy it takes to perform inference using such approaches. Model distillation is the training of a smaller, more efficient, DNN to predict with the performance close to a larger DNN or ensemble [10, 11].

When training a multiclass network, first, the softmax of network logits a_i is used to calculate class probabilities:

$$\hat{y}_i = \frac{e^{a_i/T}}{\sum_{j=1}^{|C|} e^{a_j/T}}, \tag{17}$$

where C is the set of classes which the network can identify and T is the temperature and is usually set to 1. Class probabilities are then used in the cross-entropy error function:

$$J(y, \hat{y}) = -\sum_{i=1}^{|C|} y_i \log \hat{y}_i, \tag{18}$$

where y is the correct training label for a given input and $\hat{y}$ is the vector of class prediction probability output from the network. Using standard supervised training, y is a one-hot encoded vector, with 1 in the position of the correct label, and 0

everywhere else. Therefore, when the correct class is $i = k$, Eq. (18) simplifies to:

$$J(\hat{y}, k) = -\log \hat{y}_k. \tag{19}$$

Equation (19) contains the objective function typically differentiated during the training of a large neural network.

The output probabilities of a previously trained large network capture rich information not available in the original training set, which only contain input examples and the correct label for each input. For example, assume a classification dataset which includes cars, trucks, and other non-vehicle classes. During training, when learning instances of car classes, only a single correct label (y, which is one-hot encoded) will be used. Once trained, if presented with a previously unseen photo of a car, the car and truck class probabilities will most likely both contain significant information regarding the correct class, whereas the potato class probability would not contain as much information. Model distillation uses all of this information.

There are various techniques to implement distillation. Initially, assume a large network has been trained to high performance, and a smaller network is to be trained with distillation. Additionally, assume we do not have access to the correct training labels. In this case, we may input random images into the large network and use *all* of its prediction probabilities $\hat{y}$ as a **soft target** for the distilled network's output $\tilde{y}$:

$$J(\tilde{y}, \hat{y}) = -\sum_{i=1}^{|C|} \hat{y}_i \log \tilde{y}_i. \tag{20}$$

This is similar to Eq. (18), except $y = \hat{y}$, and we have class probabilities for each entry in $\hat{y}$, so it does not simplify to Eq. (19).

If training labels are also available, the objective function can be improved by summing Eqs. (18) and (20), giving:

$$J(y, \tilde{y}, \hat{y}) = -\sum_{i=1}^{|C|} \alpha \hat{y}_i \log \tilde{y}_i + \beta y_i \log \tilde{y}_i, \tag{21}$$

where α is a hyperparameter which sets the relative importance for matching soft targets and β sets the relative performance for selecting the correct class. In practice [11] found that α should be higher than β.

In addition to hyperparameters α and β, [11] also found that the temperature in Eq. (17) impacts distillation performance. Higher temperatures make "softer" probability distributions. To understand why this may be important, consider the logits $[1, 2, 10]$, which have a softmax with $T = 1$ of $[1 \times 10^{-4}, 3 \times 10^{-4}, 9.995 \times 10^{-1}]$. The small probabilities slow down learning during back-propagation. However, when $T = 10$, the softmax becomes $[0.22, 0.24, 0.54]$, which has ranges that will cause learning to occur more quickly with back-propagation. It can therefore be useful to use high T values for the softmax of both the large network and distilled

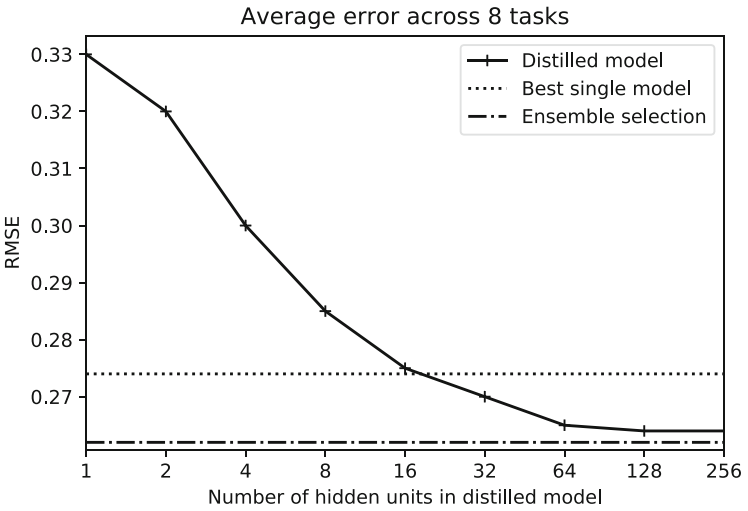


Fig. 5 An ensemble of models was trained for eight classification tasks. Distillation was then used to train a neural network to behave like each ensemble. The plot to the right compares average performance between the ensemble of classifiers, the best individual classifier in each ensemble, and the distilled classifiers. Once the distilled classifier has enough capacity, its average approaches the ensemble average [10]

network during the distillation phase.[6] After distillation is finished, T may be reset to 1.

Distillation is effective for transferring information from trained large networks to untrained smaller networks. In [11], a large DNN was trained to classify MNIST, resulting in 67 test errors. A smaller network, trained and tested with the same sets as the larger network, resulted in 146 errors. However, when the smaller network was trained with distillation, it only made 74 test errors.

Thus far we have discussed how to distill a DNN into a smaller network. Similar methods may be used to distill an ensemble of classifiers. In [10], eight binary classification problems were solved by an ensemble of methods, and then a neural network was trained by distillation to capture the behavior of the ensemble. The average performance of the small distilled model is given in Fig. 5. It can be seen that the average performance of the distilled model was similar to a giant ensemble prediction derived from SVMs, bagged trees, boosted trees, boosted stumps, simple decision trees, random forests, neural nets, logistic regression, k-nearest neighbor, and naive Bayes.

A smaller distilled model is obviously guaranteed to be more efficient than a large DNN or ensemble of models, and the distillation approaches presented in

[6]The softmax layer is at the output and has no trainable weights. It can therefore be replaced in the larger network with a separate temperature, with no need for retraining.

this section are a promising avenue to achieving adequate performance, given hard resource constraints. The steps for distillation are summarized in Algorithm 5.

Algorithm 5 Distillation

Require: Inputs I, optional targets y, previously *trained* high performance network $\langle \mathrm{W}, \mathrm{O} \rangle_{large}$, *untrained* distilled network $\langle \mathrm{W}, \mathrm{O} \rangle_{dist}$

Ensure: Trained distilled network $\langle \mathrm{W}, \mathrm{O} \rangle_{dist}$

1. Inference:

$\hat{y} \leftarrow$ output probabilities of $\langle \mathrm{I}, \mathrm{W}, \mathrm{O} \rangle_{large}$

$\tilde{y} \leftarrow$ output probabilities of $\langle \mathrm{I}, \mathrm{W}, \mathrm{O} \rangle_{dist}$

2. Calculate loss:

if targets y are available

$J(y, \tilde{y}, \hat{y}) = -\sum_{i=1}^{|C|} \hat{y}_i \log \tilde{y}_i + y_i \log \tilde{y}_i$

else

$J(\tilde{y}, \hat{y}) = -\sum_{i=1}^{|C|} \hat{y}_i \log \tilde{y}_i$

3. Update distilled model weights:

$\mathrm{W}_{dist} \leftarrow \mathrm{W}_{dist} - \eta \nabla_{\mathrm{W}_{dist}} J$

4. Optionally repeat:

Repeat steps 2 and 3 until objective function is optimized.

6 Filter Decomposition

AlexNet introduced the first popular high-performance convolutional neural network (CNN) architecture, which has since been widely adopted and modified [12]. The AlexNet architecture won fame by winning the 2012 ImageNet Challenge, which required classification across 1000 categories. AlexNet uses five convolutional layers, three fully connected layers, and other less computationally expensive layers. Modern CNNs use even more convolutional layers, for example, Google's GoogLeNet-v1 CNN architecture uses 57 convolutional layers, but only one fully connected layer.

Fully connected layers are expensive from a bandwidth perspective, because they perform only one multiply-accumulate operation (MAC) per byte transferred over memory. Convolutional layers, however, are efficient from a bandwidth perspective, but they are expensive computationally. For example, AlexNet's three fully connected layers require 58.6M MAC operations and 58.6M weights, whereas AlexNet's six convolutional layers require 666M MAC operations and only 2.3M weights. The total cost of a fully connected layer or convolutional layer is the total number of MACs plus the total number of bytes required for the layer.[7] The choice of filter sizes in convolutional layers has a large impact on the bandwidth and computational costs of a CNN. In this section we analyze the bandwidth and computational impacts of different convolutional filter designs.

[7] First-order estimates of power costs can be calculated using Table 1.

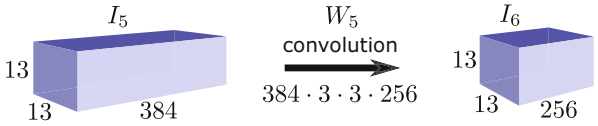


Fig. 6 Example calculation of `MAC cost` of the fifth convolution in AlexNet. For intuition in understanding `MAC cost`, consider that each point in I_6 is the result of applying a $384 \times 3 \times 3$ filter tensor to I_5. Therefore the total number of MACs needed to calculate I_6 is $256 \times 13 \times 13 \times 384 \times 3 \times 3$. This is a different perspective on the calculation than given in the main text

We loosely base our discussion on AlexNet, because it is well understood and the foundation of modern CNN designs. AlexNet convolutional layers use three filter shapes, 11×11, 5×5, or 3×3, and four channel depths, 96, 256, or 384. The shape of convolution filters has a significant impact on computational cost. To calculate the MAC cost for layer l's convolution operations, we first recall the notation introduced in Sect. 1, where layer l's filter tensor is denoted $\mathrm{W}_l \in \mathbb{R}^{c_{in} \times w \times h \times c_{out}}$ and layer l's input tensor is denoted $\mathrm{I}_l \in \mathbb{R}^{c_{in} \times x \times y}$. The number of MAC operations in a convolutional layer is found by[8]:

$$\texttt{MAC cost} = \texttt{cardinality}(\mathrm{I}_l) \times \frac{\texttt{cardinality}(\mathrm{W}_l)}{c_{in} \text{ from } \texttt{cardinality}(\mathrm{W}_l)}, \tag{22}$$

where `cardinality()` returns the number of elements in the input tensor. The bandwidth required for a filter, assuming 32-bit floating-point weights, is calculated as:

$$\texttt{Byte cost} = c_{in} \times w \times h \times c_{out} \times 4 \text{ bytes}. \tag{23}$$

The goal of efficient CNN design is to obtain the highest classification performance, using the fewest number of MACs and weights. Therefore from an efficiency perspective, the cost of CNN inference is:

$$\texttt{COST}() = c_1 \texttt{MAC cost} + c_2 \texttt{Byte cost} + c_3 \texttt{CNN errors}, \tag{24}$$

where the coefficients c depend on the priorities and budget of the CNN's designer.

To better understand Eq. (22), consider the calculation of the number of MACs in the fifth convolutional layer of AlexNet, illustrated in Fig. 6. In this case $\texttt{cardinality}(\mathrm{I}_5) = 384 \times 13 \times 13$ and $\texttt{cardinality}(\mathrm{W}_5) = 384 \times 3 \times 3 \times 256$. So the total number of MAC operations for $\mathrm{I}_5 * \mathrm{W}_5$ is $384 \times 13 \times 13 \times 3 \times 3 \times 256 = 150\text{M}$. Additionally, the size of W_5 is $384 \times 3 \times 3 \times 256 = 885\text{k}$ weights.

[8] Our calculations assume there is no pooling layer after convolution, which is now commonly the case.

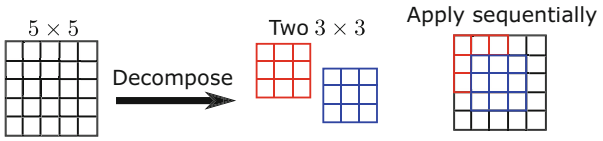


Fig. 7 A "large" convolutional filter may be separated into two smaller filters, which retain the feature detection capabilities of the larger filter. The outputs of the smaller filters are summed. This approach is used to reduce the number of bytes required to represent filters and to reduce the number of MAC operations

As another example, assume that instead of 3×3 filters, 5×5 filters were used in AlexNet's fifth convolutional layer. 5×5 filters cause the number of MAC operations to increase to 415M and byte cost to increase to 2.5 MB. A larger filter can capture more detail, and suppose that switching to a 5×5 filter increased classification accuracy, but caused the total cost to exceed the time and energy budget allotted to the CNN. Perhaps surprisingly, there are techniques to extract the benefit of 5×5 filters without using 5×5 filters.

The concept of filter decomposition was introduced in [13], where two smaller filters $\mathrm{W}_{l,1}$ and $\mathrm{W}_{l,2}$ were applied to the input tensor I_l and then added (prior to the nonlinearity), giving $\mathrm{I}_{l+1} = \mathrm{I}_l * \mathrm{W}_{l,1} + \mathrm{I}_l * \mathrm{W}_{l,2}$. As shown in Fig. 7, instead of using a single 5×5 filter tensor in the previous case, two 3×3 tensors can be used. Specifically, instead of performing $256 \times 13 \times 13 \times 5 \times 5 \times 384 = 415\text{M}$ MAC operations (requiring 2.5M weights), $256 \times 13 \times 13 \times 3 \times 3 \times 384 \times 2 = 300\text{M}$ MACs are performed (requiring 1.8M weights). Similarly, a 5×1 and 1×5 filter may be used, requiring $256 \times 13 \times 13 \times 5 \times 2 \times 384 = 166\text{M}$ MAC operations (requiring 1M weights), which is close to the original 150M MACs and 885k weights required when using a single 3×3 filter tensor.

Going even further, [14] introduced 1×1 convolutions, which are used to create **bottleneck layers**, because they can shrink an input tensor. 1×1 filters detect correlation between corresponding weights in each channel, which may be seen when considering their full notation: $c_{in} \times 1 \times 1 \times c_{out}$. For example, suppose we are given input $\mathrm{I}_l \in \mathbb{R}^{c_{in} \times x \times y}$, then a filter $\mathrm{W}_l \in \mathbb{R}^{c_{in} \times 1 \times 1 \times c_{out}}$ may be chosen such that $c_{out} \ll c_{in}$. Convolving W_l with I_l gives $\mathrm{I}_{l+1} \in \mathbb{R}^{c_{out} \times x \times y}$. The information from I_l is not lost, even though I_{l+1} now has fewer channels than I_l. 1×1 convolutions capture channel correlations, compared to larger filters which capture channel and spatial correlations.

Various filter schemes can be combined. For example, a 1×1 convolution may be followed by a 3×3 or 5×5 convolution. The goal here is to extract channel correlations using the 1×1 convolution and to extract spatial (and channel) correlations using the 3×3 or 5×5 filter. Going back to our original AlexNet example, we calculated the number of MACs used for the convolution of I_5 and W_5 as $256 \times 13 \times 13 \times 3 \times 3 \times 384 = 150\text{M}$ MACs and 885k weights. We can reduce this by picking a smaller c_{out} size for W'_5, e.g., 64, giving $256 \times 13 \times 13 \times 1 \times 1 \times 64 = 2.8\text{M}$ MACs and 16k weights. We may then add another convolution layer, using a $384 \times 3 \times 3$ filter W'_6, and return to the original shape of I_6 using

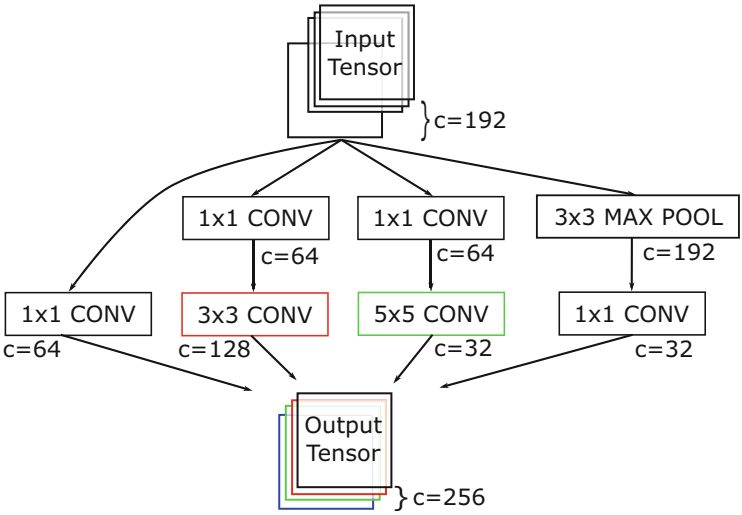


Fig. 8 Diagram of an Inception module. Layer inputs are passed through separate 1×1 bottleneck layers and then through standard convolutional layers. This technique allows for the use of different filter sizes, without paying the computational or bandwidth cost of normal convolutional layer implementations [15]

$384 \times 13 \times 13 \times 3 \times 3 \times 64 = 37$M MACs and 221k weights. We now have extracted both channel and spatial correlations, using 1×1 and 3×3 filters and a total of 39.8M MACs and 237k weights, much fewer than the original example which used 150M MACs and 885k weights. Bottleneck layers followed by convolution have proven to be an effective way to increase efficiency without sacrificing accuracy.

Filter decomposition represents a fundamentally different way to improve DNN inference efficiency, compared to earlier sections. Specifically, by making careful architectural choices, high performance can be maintained and fewer weights and MAC operations can be used. The methods introduced here may also be combined. For example, Inception is a modern CNN architecture, which combines bottleneck layers and various filter shapes to capture the benefits of every possible combination. Figure 8 illustrates an Inception module, which combines many convolutional layers and outputs each combination as stacks of sub-channels. Without the 1×1 bottleneck layers, such an architecture would be much more expensive.

7 Conclusion

Deep neural networks are increasingly being integrated into cyber-physical systems, which have power, silicon area, latency, and accuracy budgets. This chapter introduced various mathematical and algorithmic methods for optimized DNN inference:

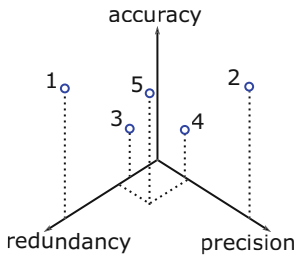


Fig. 9 The notional trade-off between accuracy, redundancy, and precision. In general, one may prioritize any two at the expense of the third [16]. There is currently no formal proof for this plot, but most of the optimization papers referenced in this chapter report metrics across the different axes and seem to generally follow the trend of this plot

- Eliminating "small" weights via pruning, which reduces the required number of multiply-accumulate operations
- Quantization, or reducing the precision, of layer inputs and/or weights to reduce computation and data transfer costs
- Sharing weights between layer units and therefore enabling data transmission compression
- Training small models to mimic larger models by distilling the information from the larger models into the smaller models
- Separating larger convolutional filters into smaller filters while retaining the performance of the larger filters

These optimization methods may be used individually or may be combined for greater optimization. Note that the methods are not equivalent and should be expected to affect performance metrics in different ways.

Unfortunately most of the optimizations introduced here will result in an accuracy loss when compared to a high-performance model which was designed with no regard to computational efficiency. The trade-off between accuracy, redundancy, and precision is depicted in Fig. 9 [16]. In general, one may expect to obtain high accuracy when using high-precision (e.g., floating-point) arithmetic (Pt. 2 in Fig. 9) and lower accuracy when using low-precision arithmetic (Pt. 4). But low-precision arithmetic may be offset with redundancy (e.g., larger models) (Pt. 1). Likewise the errors caused by using low-redundancy (few weights) models may be offset, to some extent, with high-precision arithmetic.

Ultimately, it is the DNN architect's task to find a design which achieves minimum acceptable performance, given a particular resource (e.g., latency, silicon area, power) budget. The methods introduced in this chapter facilitate this task.

Acknowledgements Sandia National Laboratories is a multimission laboratory managed and operated by the National Technology & Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International Inc., for the US Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

References

1. D. William, High-performance hardware for machine learning, in *Conference on Neural Information Processing Systems* (2015)
2. M. Rastegari, V. Ordonez, J. Redmon, A. Farhadi, Xnor-net: Imagenet classification using binary convolutional neural networks, in *European Conference on Computer Vision* (2016)
3. S. Han, H. Mao, W.J. Dally, Deep compression: compressing deep neural networks with pruning, trained quantization and huffman coding, in *International Conference on Learning Representations* (2016)
4. S. Marcel, Y. Rodriguez, Torchvision the machine-vision package of torch, in *International Conference on Multimedia* (ACM, New York, 2010), pp. 1485–1488
5. A. Parashar, M. Rhu, A. Mukkara, A. Puglielli, R. Venkatesan, B. Khailany, J. Emer, S.W. Keckler, W.J. Dally, Scnn: an accelerator for compressed-sparse convolutional neural networks, in *International Symposium on Computer Architecture* (ACM, New York, 2017), pp. 27–40
6. N.P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers, R. Boyle, P.-l. Cantin, C. Chao, C. Clark, J. Coriell, M. Daley, M. Dau, J. Dean, B. Gelb, T.V. Ghaemmaghami, R. Gottipati, W. Gulland, R. Hagmann, C.R. Ho, D. Hogberg, J. Hu, R. Hundt, D. Hurt, J. Ibarz, A. Jaffey, A. Jaworski, A. Kaplan, H. Khaitan, D. Killebrew, A. Koch, N. Kumar, S. Lacy, J. Laudon, J. Law, D. Le, C. Leary, Z. Liu, K. Lucke, A. Lundin, G. MacKean, A. Maggiore, M. Mahony, K. Miller, R. Nagarajan, R. Narayanaswami, R. Ni, K. Nix, T. Norrie, M. Omernick, N. Penukonda, A. Phelps, J. Ross, M. Ross, A. Salek, E. Samadiani, C. Severn, G. Sizikov, M. Snelham, J. Souter, D. Steinberg, A. Swing, M. Tan, G. Thorson, B. Tian, H. Toma, E. Tuttle, V. Vasudevan, R. Walter, W. Wang, E. Wilcox, D.H. Yoon, In-datacenter performance analysis of a tensor processing unit, in *International Symposium on Computer Architecture* (ACM, New York, 2017), pp. 1–12
7. M. Courbariaux, Y. Bengio, J.-P. David, Binaryconnect: training deep neural networks with binary weights during propagations, in *Conference on Neural Information Processing Systems* (2015)
8. G. Hinton, Neural networks for machine learning. https://www.coursera.org/learn/neural-networks (2012). Accessed 03/14/18
9. I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, Y. Bengio, Binarized neural networks, in *Conference on Neural Information Processing Systems* (2016), pp. 4107–4115
10. C. Buciluǎ, R. Caruana, A. Niculescu-Mizil, Model compression, in *International Conference on Knowledge Discovery and Data Mining* (ACM, New York, 2006), pp. 535–541
11. G. Hinton, O. Vinyals, J. Dean, Distilling the knowledge in a neural network (2015). Preprint. arXiv:1503.02531
12. A. Krizhevsky, I. Sutskever, G.E. Hinton, Imagenet classification with deep convolutional neural networks, in *Advances in Neural Information Processing Systems* (2012), pp. 1097–1105
13. R. Rigamonti, A. Sironi, V. Lepetit, P. Fua, Learning separable filters, *Conference on Computer Vision and Pattern Recognition* (IEEE, New York, 2013), pp. 2754–2761
14. M. Lin, Q. Chen, S. Yan, Network in network, *International Conference on Learning Representations* (2014)
15. C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, A. Rabinovich, Going deeper with convolutions, in *Conference on Computer Vision and Pattern Recognition* (IEEE, New York, 2015)
16. D. Strukov, ECE594BB neuromorphic engineering, University of California, Santa Barbara, March (2018)

A Zero-Entry Cyber Range Environment for Future Learning Ecosystems

Elaine M. Raybourn, Michael Kunz, David Fritz, and Vince Urias

Abstract Sandia National Laboratories performed a 6-month effort to stand up a "zero-entry" cyber range environment for the purpose of providing self-directed practice to augment transmedia learning across diverse media and/or devices that may be part of a loosely coupled, distributed ecosystem. This 6-month effort leveraged Minimega, an open-source Emulytics™ (emulation + analytics) tool for launching and managing virtual machines in a cyber range. The proof of concept addressed a set of learning objectives for cybersecurity operations by providing three, short "zero-entry" exercises for beginner, intermediate, and advanced levels in network forensics, social engineering, penetration testing, and reverse engineering. Learners provided answers to problems they explored in networked virtual machines. The hands-on environment, Cyber Scorpion, participated in a preliminary demonstration in April 2017 at Ft. Bragg, NC. The present chapter describes the learning experience research and software development effort for a cybersecurity use case and subsequent lessons learned. It offers general recommendations for challenges which may be present in future learning ecosystems.

1 Introduction

Most technology-mediated learning interventions, instructions, and assessments are intended for use in blended (instructor-led) or formal (schoolhouse-based) learning contexts. Most cybersecurity education geared for adult learners is delivered online via e-learning slide presentations, webinars, video lectures (see Federal Virtual Training Environment https://niccs.us-cert.gov/training/federal-virtual-training-environment-fedvte), or face-to-face consisting primarily of lecture,

E. M. Raybourn (✉) · M. Kunz · V. Urias
Sandia National Laboratories, Albuquerque, NM, USA
e-mail: emraybo@sandia.gov; mkunz@sandia.gov; veuria@sandia.gov

D. Fritz
Sandia National Laboratories, Livermore, CA, USA
e-mail: djfritz@sandia.gov

Ç. K. Koç (ed.), *Cyber-Physical Systems Security*,
https://doi.org/10.1007/978-3-319-98935-8_5

hands-on individual practice, team practice, and/or Capture the Flag (CTF) exercises. While cybersecurity education may require self-motivation for student success, few curricula are actually designed to facilitate *informed self-directed learning* such as that found in an apprenticeship, internship, or mentorship program (see Sandia National Laboratories Technical Internships to Advance National Security [TITANS] http://www.sandia.gov/titans/), and even fewer still offer *connected* learning experiences such as those supported by storytelling that comes to life in and across different media—offering off-ramps to auxiliary resources and activities intended to incentivize rich on-demand, self-directed, informal learning. Connected learning experiences are facilitated by transmedia learning. *Transmedia learning* is defined as the scalable system of messages representing a narrative or core experience that unfolds from the use of multiple media, emotionally engaging learners by involving them personally in the story [1].

A science and technology (S&T) goal for many is to enable personalized, data-driven, and lifelong technology-enabled learning. The long-term goal is that ecosystems of connected, transmedia systems will provide adaptive, personalized learning that is facilitated by data shared among technologies in the ecosystems. While transmedia learning is a goal of future learning ecosystems, near-term emphasis is usually placed on supporting instructor-led, blended, linearly sequenced, or stand-alone learning via different web-based technologies. In that respect, future transmedia learning ecosystems—those that offer self-directed, connected, story-driven learning experiences—must not only engage the learner personally but also provide authentic experiences for learners of all levels. The S&T challenge is to create rapidly configurable environments and learning pathways flexible enough to support a learner's unique and authentic journey across multiple media and modalities, and designed to promote self-directed exploration over time.

Our team developed *Cyber Scorpion*, a Capture the Flag cyber range lab environment to address this gap. Cyber Scorpion can share exercise completion data through a web interface using the xAPI specification [2] saving to a Learning Record Store (LRS). Subsequent sections describe the research and development effort of Cyber Scorpion for the specific use case of offering off-ramps to auxiliary cybersecurity resources, activities, and lessons learned. Cyber Scorpion logged when learning started, when milestones were being attempted, and when milestones were completed. While Minimega, the technology underlying the virtual lab in Cyber Scorpion, is capable of stealth assessment (see [3]), learner actions and behaviors in the emulated environment were not logged or recorded in the present effort.

2 Limitations of Current Practice

As described in the previous section, the current practice of cybersecurity training for adult learners is largely delivered online via e-learning slide presentations, webinars, video lectures, slide presentations, or face-to-face consisting primarily of lecture, hands-on individual practice, team practice, and/or Capture the Flag

(CTF) exercises. However, the current practice of pass/fail assessment results in insufficient fidelity to reveal useful detail about whether/how/when "learning" is occurring. Although adult active duty military or reserve learners may engage in highly orchestrated exercises distributed across installations in the United States or around the globe, these exercises, such as Cyber Flag and Cyber Guard, may only occur once a year. Additionally, many ranges used for these exercises are "heavy"—being bound to a physical infrastructure, particular vendor software stack, databases, and large maintenance staff.

Military-grade cybersecurity game-based training is gaining popularity, but these games are usually not lightweight and rapidly configurable, nor rapidly scalable. Other options for training are more "academic" and often do not reflect real-world network or adversary behavior. Modernization will require much more realistic scenarios utilizing robust models, simulations, and emulations, with adaptive, persistent, and blended live, virtual, constructive, and gaming environments [4]. According to retired CYBERCOM Chief of Staff, Air Force Major General Jim Keffer, "We don't have—but we need—an exercise environment where you do rehearsals, go against adversary networks, and figure out ways to better protect your own . . . the team training, the force-on-force training, that is primarily limited by a lack of a persistent training environment" [5].

Another limitation to the current practice is the lack of congruency with respect to language and meaning leading to misunderstandings. For example, the definitions for emulation and simulation are often confused, especially when referring to cyber training environments. *Emulation* has been defined as a reproduction or replica of the function or action of a particular system, whether it is software or hardware. An emulation replicates a system as specifically and exactly as possible. *Simulation*, on the other hand, models the internal state of a system and is an abstraction rather than an exact replica. Emulation may or may not model the internal state of a system. In cybersecurity operations training, it is important to "train as you fight, fight as you train." To do this, training should be executed in *emulated* systems and networks.

Finally, cyber operations training is not nearly as on-demand as required, so learners do not have the opportunity to continually train as much as they should. Therefore, much of the current practice is either limited to training that is not rapidly configurable, unengaging, and stale or highly engaging training that is executed by face-to-face teams or via logistically complicated, over-orchestrated distributed exercises.

To summarize, there are at least three limitations to the current practice that Sandia Cyber Scorpion sought to address:

- Not rapidly configurable or scalable
- Not persistent or on-demand
- Not sufficiently realistic

2.1 Specific Problem Being Solved

Currently, cyber defender training is performed on (1) operational systems, (2) a limited testbed, or (3) simulated models of the system of interest. Each of these has inherent limitations that can be addressed by using an emulated system.

According to Urias et al. [6], "Analysis and training on operational systems is usually limited to the most benign levels since any disruption to the operational system has potentially severe consequences. Testbeds for analysis and training are typically expensive and time-consuming to construct and deploy, single-purpose, and difficult to maintain. Another option for cyber defender training would be a simulated environment. In many cases however, the simulation program code needs to be developed to simulate the system and devices in question or extensions need to be made to answer specific questions. These (sometimes buggy) simulation codes typically do not depict an accurate picture of the system. To increase simulation result accuracy, models have to be extended and validated."

These processes can be time-consuming and inefficient. Thus there is a need for the ability to rapidly create, tear down, recombine, and reuse high-fidelity replications, or emulations, of information systems for cybersecurity training. Our team has participated on multi-year research projects in the development of new strategies and methodologies that enable researchers to quickly and accurately model information systems hosts and networks of interest for cyber analysis and training. Therefore an early determination was made that Cyber Scorpion should serve as a "zero-entry" practice environment.

The specific problem being solved by Cyber Scorpion is the demonstration of the ability to offer a cyber range capability with a government-owned, open-source virtual machine (VM) management tool called Minimega via a web interface with the ability to share learner data. Cyber Scorpion, by virtue of being an emulated environment, is rapidly configurable and able to support persistent training that allows learners to "train as they fight, fight as they train."

3 Research

3.1 Research Question

The authors of the present chapter addressed the following research question, "What are the learning experience challenges associated with bringing a zero-entry, cyber range environment to future learning ecosystems that allow learners and instructors to transition among learning activities, devices, and modalities?"

Subsequent sections detail the approach taken by our team for design and software development, followed by the lessons learned.

3.2 Learning Science Approach

Learning experience design (LX) is a subset of user experience design (UX) that addresses the synthesis of learning sciences, human-computer interaction, and design thinking. Learning experience design puts the learner at the center of the product or service design process. As more immersive simulations and persistent transmedia learning [1] experiences are developed with distributed exercise environments, games, and virtual/mixed/augmented realities, it can be useful to ground approaches in theory such as Distributed Cognition (see Sect. 3.2.2) and design methods such as the Simulation Experience Design Method [4]. The Simulation Experience Design Method specifically aims to bring to the fore initial assumptions, biases, or notions of expectations that inform the decisions shaping the design of learning experiences. Sections 3.2.1 and 3.2.2 further discuss the method used to design the learning experience and its theoretical underpinning.

3.2.1 Simulation Experience Design Method

The Simulation Experience Design Method and Framework [4] is a process that addresses the design of learning as a system of experiences that exists within an emergent, adaptive cultural context that the designer strives to engender throughout engagement, as well as before, between, and after formal learning has concluded.

The word simulation in the name of the method refers to an experience in which the role of a human, environment, or both can be simulated. The Simulation Experience Design Method, briefly described in this section, has been applied by the author and others to serious game design [4, 7] and transmedia learning [1]. Whether UX or LX, experience design solutions require that designers understand what makes a good experience first and then translate these principles, as efficiently as possible, into the desired medium without the technology dictating the form of the experience. In simulated environments in which learners are creatively problem solving together, one's experience may be unpredictable, may not have a right or wrong approach, or may not be what the designer intended. The Simulation Experience Design Method can be helpful in framing the co-creation of problem-solving opportunities as an open-ended, rich *system of experiences* that fosters learning (Fig. 1).

The Simulation Experience Design Method suggests that supporting equitable intercultural communication and learning is comprised of several salient elements, among them (1) the *interactions* or type of communication (interpersonal, group, etc.); (2) the *narratives* that are co-created by interlocutors; (3) the *place*, or context, in which narratives occur; and (4) the *culture that emerges* from the social construction of experience [8]. Following the circular framework from upper left to upper right, design tasks may then be considered as facilitating a journey or connected learning experience from interactions to emergent culture that iteratively lead to new interactions spawned by the emergent culture. Use of the framework

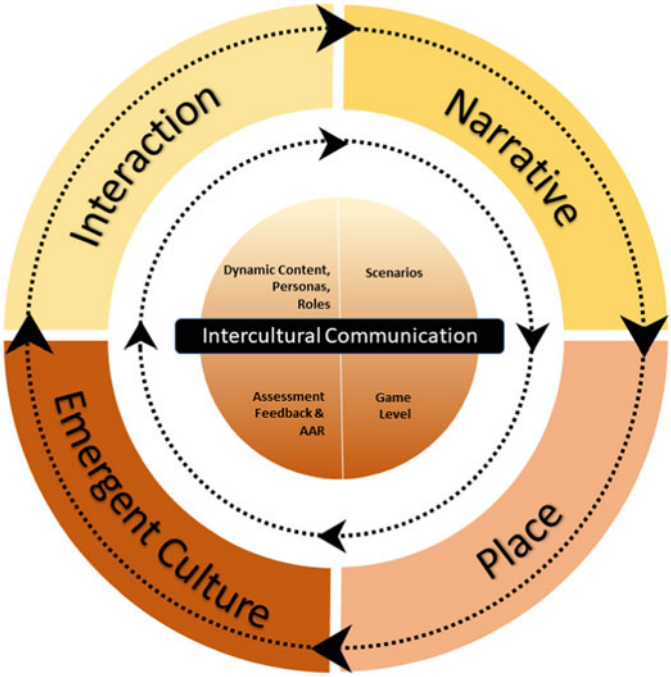


Fig. 1 Simulation Experience Design Method and Framework [4]

is intended to improve the quality of equitable learning in collaborative, immersive environments such as serious games, simulations, and transmedia storytelling and learning ecosystems [1, 4, 7].

Finally, by treating intercultural communication as a *core value*, the individual cultural backgrounds the players bring to their experiences are considered strengths, not design liabilities. As we strive to create engaging immersive experiences, differing cultural values of designers, developers, stakeholders, and players can create a myriad of complications and competing desires or expectations. The Simulation Experience Design Method can serve to socially construct narratives and establish a shared understanding for thoughtful analysis from which to better ground assessment and evaluation of human performance, creativity, and expertise [9].

3.2.2 Distributed Cognition Theory

The theory of Distributed Cognition, advanced by Edwin Hutchins [10], provides a framework from which cognition can be viewed as an ecosystem involving people, artifacts, tools, and environments. The Simulation Experience Design Method and Framework applies Distributed Cognition and the notion of "cognition in the wild" to LX design. Cognition in the wild refers to human cognition as it naturally

occurs and adapts in the everyday world—situated in culturally constituted human activity [10]. Distributed Cognition is a distributed, social process in which human knowledge and cognition are not confined to the individual and often reside in other people, tools, or artifacts. As we move along the Simulation Experience Design Method and Framework to supporting learning with technology, we can see the application of Distributed Cognition. For example, learners may work in face-to-face teams, but they also use media—papers, pencils, tablets, collaborative tools, computers, etc. They may engage in dialogue as they walk through Cyber Scorpion exercises, but they may also take notes or save data to extend their working memory. Their cognition may be characterized as distributed among artifacts and people. After the 10-hour test and demonstration, they may have engaged in online discussions and/or crowdsourced information via social media. Their cognition is also distributed among persistent artifacts such as digital messages that facilitate asynchronous collaboration and help them remember, understand, and connect with others. In this sense, their working knowledge exists among diverse systems and cannot truly be understood without taking this socially distributed sense-making into account [11]. Distributed Cognition underpins distributed learning science and particularly transmedia learning, by providing a grounding theory for the learning experience facilitated by transmedia ecosystems utilizing diverse media, devices, and modalities.

3.3 Learner Analysis

The John F. Kennedy Special Warfare Center and School (JFKSWCS) is the United States Army's school for professional training of Army special operations forces personnel. The United States Army JFKSWCS (USAJFKSWCS) is also responsible for training US Army Reserve and National Guard Civil Affairs and Psychological Operations conventional forces. As a component subordinate command of the United States Army Special Operations Command, JFKSWCS enables the Army Special Operations Force (ARSOF) force modernization and conducts institutional training through a headquarter, center, and school. There are two formal distance learning (DL) instances in JFKSWCS training—(1) the Reserve Component Civil Affairs (CA) Phases 1 and 3 of Captains Career Course and (2) the Psychological Operations (PSYOP) Reserve Officer Qualification Course Phase 1.

Little was known about the individual learner's job/role at the time of Cyber Scorpion learning experience development. However, we assumed that since learners could potentially be from CA, Military Information Support Operations Command (MISOC) formerly PSYOP, and SF (Special Forces), we could apply this general information to the design of a zero-entry environment that would engage beginners, intermediate, and advanced learners.

We also observed that the learners could be highly motivated individuals who are risk-takers, adaptive, and competitive. They could speak a second or third language or be in the process of learning one. Trained in understanding the human

Table 1 Terminal learning objectives

1.	Describe network architecture, browser configuration, and hardware/software for secure Internet browsing
2.	Understand social engineering concepts in cyberattacks (e.g., phishing, waterhole attacks, lures)
3.	Run penetration tests to locate potential security threats (network enumeration using NMAP, Wireshark)
4.	Know how to analyze a network packet capture file
5.	Recognize, document, and analyze a successful attack from point of entry, pivoting, and systems controlled
6.	Successfully execute the steps in creating a software exploit

domain, these individuals could have prior knowledge that is relevant to the topic of social engineering, but not likely network forensics, penetration testing, or reverse engineering. As they may already be skilled social engineers, their expectations could be high regarding this topic, and it is likely that this topic will be easiest for them and of interest. In either case, they would not like to waste their free time, since they do not normally get a lot of it. Given the background information above, we concluded that the learning experience be designed to address the variance in prior knowledge, interest, and familiarity with the technical topics.

3.4 Cybersecurity Terminal Learning Objectives

Informed by the learner analysis, we identified six terminal learning objectives (TLOs) for cybersecurity that were based on the National Cybersecurity Workforce Framework [12]. The terminal learning objectives are documented in Table 1.

Given the tactical emphasis of the JFKSWCS, we decided to address terminal and enabling learning objectives by focusing on providing "hands-on practice" in the emulated Cyber Scorpion environment.

4 Cyber Scorpion Design

Based on the observations from the learner analysis, our technical approach for the preliminary user experience demonstration was to design an experience that facilitated a "zero-entry" mindset intended to incentivize learners' curiosity to dig deeper and explore transmedia content. Using The Simulation Experience Design Method [4], a hypothetical narrative was generated to identify potential challenges and opportunities that might arise during the learning experience demonstration [13]. This narrative, or learner sketch, also informed the design of Cyber Scorpion exercises and its interaction experience approach.

4.1 Cyber Scorpion Learner Sketch

An excerpt of the full learner sketch is provided in the present section to orient the reader toward the intended use of Cyber Scorpion in the context of the transmedia learning demonstration.

> US Army Captain Angela Diablo has volunteered for a 4-day cybersecurity transmedia learning experience. She has volunteered because in her support role as a PSYOP specialist she needs to leverage cyber technologies to conduct research for the development of PSYOP campaigns. She's interested in learning about offensive and defensive techniques. She arrives at the computer lab and sits in front of an open work area.
>
> The facilitator directs her attention to the devices in front of her: a laptop, a smart phone, ear buds, and a tablet. She picks up the tablet and notices content on the basics of social engineering. There is a flashcard game that is fun for a while. She watches a video on the smart phone. She looks at the clock and 20 minutes have gone by. This background information is okay, but she wants to test her skills. She logs into the laptop and sees several applications: a lecture-based cyber security course, some penetration testing tools with associated exercises, videos and PowerPoint slides, read me files on how to access a cyber range called Cyber Scorpion, and a game. She starts playing the education game. It starts out easy, but she soon realizes she doesn't understand how to solve the challenges.
>
> She decides to brush up her skills with Cyber Scorpion, where she can learn pentesting and gain hands-on experience. While getting hands-on training in the Cyber Scorpion environment, she proceeds at her own pace through the scenarios and puzzles by watching Cyber Scorpion step-by-step video tutorials and accessing other resources available on the laptop or from the Internet. Cyber Scorpion allows her to use virtual machines from the web interface without downloading a plug-in. The zero-entry approach gradually increases the difficulty without overwhelming her confidence. After a few hours of hands-on training, she's ready to try the game or go straight to the Cyber Scorpion "capstone" final assessment exercise.

We anticipated an interaction experience during the demonstration for motivated, self-directed learning. Cyber Scorpion was designed to increase learner familiarity and boost confidence. We believed this approach to LX could also engender curiosity and encourage learners to engage other content longer.

Subsequent sections describe the software development approach and underlying technology for Cyber Scorpion.

5 Software Environment

5.1 Cyber Scorpion Underlying Technology: Minimega

Cyber Scorpion leverages a distributed virtual machine (VM) lab environment that is managed by using an open-source tool called Minimega (see http://Minimega.org, https://github.com/sandia-Minimega/Minimega). Cyber Scorpion reflects a low cost of entry by being accessible from a modern browser which allows learners to train from whichever platform they chose, whether that be desktops, laptops, tablets, or phone. The virtual machine state was synchronized across all their platforms

facilitating multitasking and allowing seamless transition from platform to platform. Thus, Cyber Scorpion was able to provide off-ramps for more connected learning experiences such as those supported by storytelling across different media and on-demand, self-directed learning.

Minimega was designed to easily integrate VMs into other systems, training toolkits, and front-ends through a simple scripted interface.

The Minimega platform can be installed on both commodity desktop Linux environments for individual training and on clusters of machines, which allow for large-scale team training/experimentation. Sandia National Laboratories leverages decades of supercomputing and high-performance computing (HPC) expertise to provide scale to networks. In extreme scenarios, Minimega has been able to launch experiments with over 4 million endpoints.

Between 2012 and mid 2013, Minimega supported more than 12 active projects for university and private industry use that were not involved with Sandia National Laboratories. Minimega is also used by over a dozen government sponsors for test and evaluation of hardware and software stacks in representative environments.

5.2 Software Development Approach

Cyber Scorpion focused on providing approximately 10 hours of digital content (hands-on training environment, video walk-through, and exercises) to introduce and coach learners on topics consistent with the learning objectives. Straightforward exercises were developed to incentivize learning in an immersive, realistic cyber environment. Examples of the analysis techniques, procedures used to perform offensive and defensive maneuvers, and exposure to common tools were provided.

Micro scenario exercises were designed to "get familiar with tools and techniques" while distilling "key nuggets" of information for micro training sessions, etc. Videos of "what right looks like" provided lead-ins to problem-based learning. Cyber Scorpion environment served as a "capstone" final assessment exercise for the test and demonstration.

Cyber Scorpion is a Jeopardy-style Capture the Flag interface based on the Open-Source Technology CTFd (see https://github.com/CTFd/CTFd). Identity management was handled by the Open-Source Project Keycloak, and when people logged into their machines at the start of their day for training, they did not have to log in again. Actions learners performed in the web interface, such as starting or completing a challenge, were reported to a repository. From whatever platform users chose to use (tablet, mobile, desktop, etc.), they would be presented with the responsive web page in Fig. 2.

When a challenge tile was clicked on, the learner was presented with a question and a link to open a new tab that instantly bridged users into a virtual lab where they had control of their own remote machines. These machines were required to solve the challenge and provided real-world hands-on cybersecurity experience. The following screenshot shows the virtual machine integration in the browser (Fig. 3).

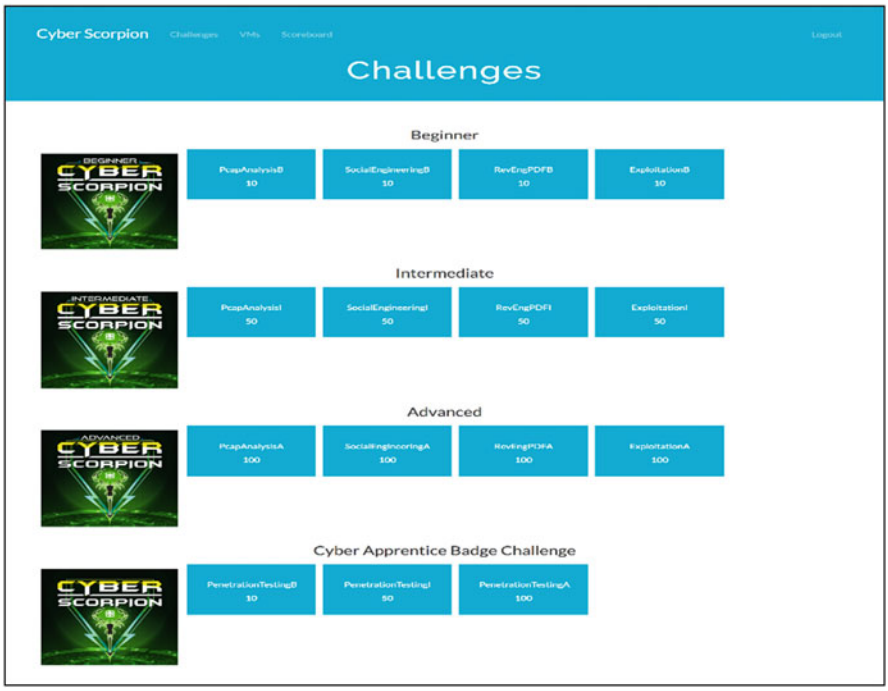


Fig. 2 Cyber Scorpion jeopardy framework (CTFd)

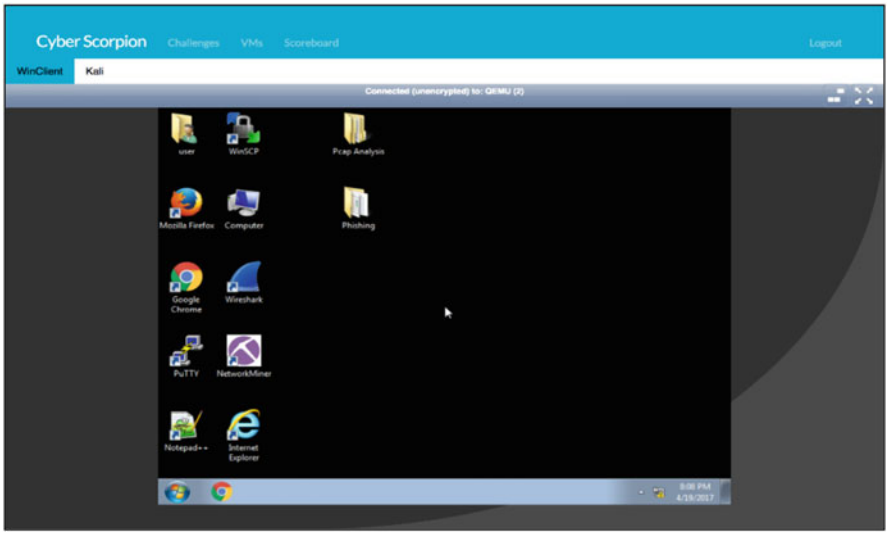


Fig. 3 Cyber Scorpion jeopardy framework (Minimega)

Fig. 4 Cyber Scorpion Jeopardy scoreboard

This session, where learners could control the keyboard and mouse, was shared and synced live between whatever platform learners chose to use, including a mobile phone or tablet. The remote machines had training modules with objectives that when completed netted a secret phrase, commonly referred to as a flag, hence "Capture the Flag."

This flag could then be entered by the learner into Cyber Scorpion's Jeopardy board, and when submitted, the framework would send predefined progress statements to a repository.

The CTFd Jeopardy framework also has a built-in scoreboard, and during the competition, administrators could follow what challenges were being completed and could track the progress of the learners (Fig. 4).

A pilot test was conducted a month prior to the demonstration at Ft. Bragg. During the pilot test, Cyber Scorpion was able to prove scalability, demonstrating 50 simulated learners heavily using the application. It was concluded that Cyber Scorpion would demonstrate well as the exercise environment elicited positive feedback and was proposed by the SME assisting the research team to be the

culminating learning assessment environment for learners who would later pursue the network forensics training track.

6 Learner Experience Demonstration and Lessons Learned

A preliminary learning experience demonstration of Cyber Scorpion (and other technology resources) was conducted at the JFKSWCS, Ft. Bragg, NC, in the spring of 2017 during a 10-hour period of spaced exposure (2 hours per day over 4-day period). Sixty-seven learners from ages 18–30, volunteered to use the resources during the 10-hour period to increase their understanding of cybersecurity in two different areas: social engineering and network forensics. Participants used resources at their own discretion. All had prior experience with mobile phones and computer-based e-learning, but fewer were familiar with tablets, simulations, and games [14]. Some completed certain activities to achieve badges in "social engineering" and "cyber apprentice." To achieve the badge for "cyber apprentice," participants completed the pentest exercises in Cyber Scorpion as their culminating "competency" learning assessment.

Learner anecdotal feedback indicated that the ability to move from an interactive multimedia instructional (IMI) resource (i.e., game) and Cyber Scorpion (real-world hands-on cyber range lab) worked well. Participants explored a topic in the game and then tested their knowledge using Cyber Scorpion exercises.

User feedback was captured during the exercise and in one-on-one interviews conducted at the end of the demonstration to better gauge whether learners thought Cyber Scorpion could be better tailored in the future. The learners who volunteered for the test and demonstration sessions were mostly nontechnical. Teaching cybersecurity topics to people of such varying backgrounds is very difficult, and the feedback illuminated this.

For some, the exercises were too easy, while for others (without command line experience) they found the exercises appropriate. Without the ability to conduct a user needs analysis, Cyber Scorpion content was developed to be very direct and capable of being learned without much prior experience. With more information about the learners in advance, Cyber Scorpion could have been better tailored to be more in line with users' existing knowledge, skills, and abilities (KSAs). It's interesting to note that, as predicted, other learners focused on completion badges and a chance to take home a coin as a prize. These learners achieved their own training goals by being self-directed and actively sought out exercises that were interesting to them, which they felt incentivized to complete.

In general, the preliminary learning experience demonstration illuminated valuable lessons learned for future development. During this 6-month effort, we addressed the following research question, "What are the challenges associated with bringing a zero-entry, cyber range environment to future learning ecosystems that allow learners and instructors to transition among learning activities, devices, and modalities?"

A number of key observations were made during this preliminary development and demonstration cycle. First, a few factors contributed to Cyber Scorpion's successful demonstration:

1. *Leveraging open-source frameworks*. Cyber Scorpion itself is an open-source framework, and additional tools such as the scoreboard system were quickly integrated into its platform.
2. *Not attempting to be everything to everyone*. Focusing the exercises on tasks that supported learning objectives is consistent with modern cyber-education pedagogies. While there is an opportunity to iterate and improve pedagogical approaches, the "zero-entry" approach appeared to work well given the learners, the difficulty level of the content, and amount of time on task, and spaced exposure.
3. *Using contemporary technologies*. Many educational frameworks are outdated soon after deployment, and this is especially true with cyber-education frameworks. The use of a highly configurable, emulated practice environment contributed to its success.

In addition to these success indicators, we observed several challenges or areas where improvements could be made in future phases. They include:

4. *Scalability*. We provided small-scale hosting for the Ft. Bragg demonstration via an ad hoc network connection and development hardware. While successful, the need for locally hosted infrastructure, scaled to the expected audience, is critical for continued success.
5. *Atypical interfaces*. While an S&T goal is to enable personalized, data-driven, and lifelong technology-enabled learning, many of the envisioned interfaces are atypical of those most cybersecurity professionals expect to use. By providing a more generic (workstation, tablet, Android mobile phone, or other OS) accessible interface, several technical limitations could be avoided.
6. *Cybersecurity*. Due to the potential sensitivity of learner privacy and data security, a design assurance perspective (designing cybersecurity into the prototypes and subsequent iterations) is recommended when designing new systems for training and education [15]. Also see Sandia National Laboratories IDART—Information Design Assurance Red Team (http://www.idart.sandia.gov/).

7 Limitations and Future Work

Cybersecurity training is not nearly as on-demand as required, so learners do not have the opportunity to continually train as much as they should. Much of the current practice is either limited to training that is not rapidly configurable, unengaging, and stale *or* highly engaging training that is executed by face-to-face teams or via logistically complicated, over-orchestrated distributed exercises. Sandia intended to address this gap with Cyber Scorpion, a zero-entry cyber range environment

offering off-ramps to auxiliary resources and activities intended to incentivize rich on-demand, self-directed, transmedia learning.

As noted previously, Capture the Flag (CTF) exercises may not provide adequate, real-time assessments of recorded events or facilitate observation of human performance without introducing artifacts into the system. Additionally, there is rarely a standardized concept or methods for offering and assessing the efficacy of cyber training, even though multiple recommendations to develop and implement standards have been made [6].

Few CTF exercises are supported with auxiliary material to enable on-demand, informal learning. Future development could better address this gap through the application of design principles and methods toward transmedia learning [1].

CTF-style measurement, in the long run, does not have sufficient fidelity to reveal much about learning that is occurring. Therefore, future research should focus on capturing and interpreting learner activity in these self-directed exercise environments.

Future work may consist the following:

- Instrument Cyber Scorpion to share (output) and leverage (ingest) learner data and/or analytics generated by other systems.
- Design and test human performance-based assessment for immersive environments.
- Develop more training modules (with varying degrees of complexity and story-driven off-ramps) for Cyber Scorpion.

8 Conclusion

In the highly VUCA environment that is cyber operations, training is obsolete as soon as it is deployed. A survey of service strategy documents highlights the shared belief of the need for training and education modernization and some congruence on how to achieve it. Modernization will require much more realistic scenarios utilizing robust models, simulations, and emulations, with adaptive, persistent, and blended live, virtual, constructive, and gaming environments. According to retired CYBERCOM Chief of Staff, Air Force Major General Jim Keffer, "We don't have—but we need—an exercise environment where you do rehearsals, go against adversary networks, and figure out ways to better protect your own . . . the team training, the force-on-force training, that is primarily limited by a lack of a persistent training environment" [5].

This 6-month effort leveraged an existing technology (Minimega) utilized by CPT (cyber protection teams) and DOD agencies to facilitate cyber operator mission rehearsal. The resulting exercise environment, Cyber Scorpion, is a zero-entry practice/competency mastery environment.

Minimega is used by over a dozen government sponsors for test and evaluation of hardware and software stacks in representative environments. Because of this, Cyber

Scorpion can be used for training, mission rehearsal, experimentation, or testing theories and hypotheses related to training efficacy, human systems integration, learning science, visualization, and development of data-driven, learner behavior analytics.

Our technical approach to use Minimega to manage the distributed VMs for Cyber Scorpion resulted in a robust, stable software environment. Our learning science approach was successful because the exercises are purposely kept simple, approachable, and doable—to offset a potentially, unnecessarily complicated learning experience that may have otherwise introduced increased cognitive load.

In future phases, we anticipate employing better understanding of capturing learner activity in constructivist environments such as scenario-based simulations and emulated practice exercises for cybersecurity training and testing.

Acknowledgment Sandia National Laboratories is a multimission laboratory managed and operated by the National Technology and Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International, Inc., for the US Department of Energy's National Nuclear Security Administration under contract DE-NA-0003525. This work was supported by the ADL Initiative (Contract QL6H5R66F007MP-0). The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the ADL Initiative.

References

1. E.M. Raybourn, A new paradigm for serious games: Transmedia learning for more effective training & education. J. Comput. Sci. **5**(3), 471–481 (2014)
2. ADL Initiative, *xAPI-Spec* (2016). https://github.com/adlnet/xAPI-Spec/blob/1.0.1/xAPI.md. Accessed 28 Feb 2018
3. V. Shute, M. Ventura, *Stealth Assessment: Measuring and Supporting Learning in Video Games*, The John D. and Catherine T. MacArthur Foundation Reports on Digital Media and Learning (MIT Press, Cambridge, MA, 2013)
4. E.M. Raybourn, Applying simulation experience design methods to creating serious game-based adaptive training systems. Interact. Comput **19**, 207–214 (2007)
5. A. Tilghman, Without solid training options, mysterious cyber training command remains a work in progress. Military Times. (2016, 5 June), http://www.militarytimes.com/story/military/tech/2016/06/05/military-cyber-offensive-defensive-weapons-training/85033862
6. V. Urias, B. Van Leeuwen, B. Wright W. Stout, in *Emulytics™ at Sandia National Laboratories.* Proceedings of MODSIM (NTSA, Arlington, VA, 2015)
7. T. Bergin-Hill, R. Creekmore, J. Bornman, *Designing a Serious Game for Eliciting and Measuring Simulated Taxpayer Behavior* (The MITRE Corporation, McLean, VA, 2014)
8. B. Laurel, *Computers as Theater* (Addison-Wesley, Reading, MA, 1991)
9. E.M. Raybourn, in *A Metaphor for Immersive Environments: Learning Experience Design Challenges and Opportunities*. Proceedings of MODSIM (NTSA, Arlington, VA, 2016)
10. E. Hutchins, *Cognition in the Wild* (The MIT Press, Cambridge, MA, 1995)
11. J. Bruner, The narrative construction of reality. Crit. Inq. **18**(1), 1–21 (1991)
12. W. Newhouse, S. Keith, B. Scribner, G. Witte, *National Initiative for Cybersecurity Education (NICE) Cybersecurity Workforce Framework*. NIST Special Publication 800-181 (2017). https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-181.pdf. Accessed 28 Feb 2018

13. Folsom-Kovarik, J.T. & Raybourn, E.M., in *Total Learning Architecture (TLA) Enables Next-Generation Learning Via Meta-Adaptation*. Proceedings of the I/ITSEC (NTSA, Arlington, VA, 2016)
14. P.S. Gallagher, J.T. Folsom-Kovarik, S. Schatz, A. Barr, S. Turkaly, in *Total Learning Architecture Development: A Design-Based Research Approach*. Proceedings of the I/ITSEC (NTSA, Arlington, VA, 2017)
15. Raybourn, E.M., Fabian, N., Davis, W., Parks, R.C., McClain, J., Trumbo, D., Regan, D., Durlach, P., in *Data Privacy and Security Considerations for Personal Assistants for Learning (PAL)*. Proceedings of the 20th International Conference on Intelligent User Interfaces Companion (2015), pp. 69–72

Parallel Programming in Cyber-Physical Systems

Sandro Bartolini and Biagio Peccerillo

Abstract The growing diffusion of heterogeneous Cyber-Physical Systems (CPSs) poses a problem of security. The employment of cryptographic strategies and techniques is a fundamental part in the attempt of finding a solution to it. Cryptographic algorithms, however, need to increase their security level due to the growing computational power in the hands of potential attackers. To avoid a consequent performance worsening and keep CPSs functioning and secure, these cryptographic techniques must be implemented so to exploit the aggregate computational power that modern parallel architectures provide. In this chapter we investigate the possibility to parallelize two very common basic operations in cryptography: modular exponentiation and Karatsuba multiplication. For the former, we propose two different techniques (*m*-ary and *exponent slicing*) that reduce calculation time of 30/40%. For the latter, we show various implementations of a three-thread parallelization scheme that provides up to 60% better performance with respect to a sequential implementation.

1 Introduction

Cyber-physical system, or CPS, is the term used to denote a variety of systems in which the components are electronic devices that form a network and interact with the environment via physical inputs/outputs.

They are already shaping the world we live in and are expected to play a strategical role in the upcoming society and world in general. Their growth and diffusion are comparable with the "Internet of Things" (IoT) trend, and furthermore these two concepts share a similar architecture. There are already many examples in the real world of CPSs, and other examples are going to make their appearance in near future. For instance, CPSs operating physical plants and production systems,

S. Bartolini (✉) · B. Peccerillo
Department of Information Engineering and Mathematical Sciences, Università degli Studi di Siena, Siena, Italy
e-mail: bartolini@dii.unisi.it; peccerillo@dii.unisi.it

Ç. K. Koç (ed.), *Cyber-Physical Systems Security*,
https://doi.org/10.1007/978-3-319-98935-8_6

monitoring and tracking shipments for optimal logistics, monitoring passengers and/or vehicles in public transport systems, controlling irrigation in fields depending on the recorded humidity levels, controlling various aspects and services in smarthomes and smart cities, smart vehicles and integrated transportation, and so on.

In these connected, distributed, and (semi-)autonomous systems, various facets of safety and security have a crucial importance [19], specifically the security of some involved data (e.g., privacy of personal data or safe management of industrial sensitive information) and, most importantly, the operational security of the systems/services themselves. It is *fundamental* to avoid malicious intrusion into the systems and to avoid data tampering even from sensors and peripheral devices with reduced computational capabilities. In fact, also this *peripheral* kind of attack could hamper the rated functioning of systems up to the risk of possibly inducing big direct and indirect economical losses to companies and people, severe threats for human lives, and even public safety of countries [11].

Cryptographic protocols, algorithms, and related techniques are key ingredients, though not the only ones needed, for securing cyber-physical systems. However, CPSs pose specific challenges to cryptographic strategies for reaching the overall security and safety. In fact, CPSs are, first of all, very heterogeneous systems in their composition and in their distribution both physical logical and also related to the network connectivity of the various subsystems.

The deployment of security features into systems in general, and into cyber-physical ones in particular, requires highly efficient implementations from both performance and energy standpoints. In fact, as security requirements need to increase over time due to the growing computational power easily available to possible attackers, cryptographic techniques need to increase their security level accordingly (e.g., key size). At the same time, the presence of a variety of simple embedded devices into current and future CPSs introduces a great pressure in running complex cryptographic calculations fast, reliably, and efficiently even with scarce computational resources.

1.1 Parallel Architectures

About 15 years ago, *uniprocessor* design began to approach its limits: the end of frequency scaling due to heat dissipation and power consumption reaching a critical threshold [16], the end of ILP (*Instruction-Level Parallelism*) improvements due to power growing faster than performance [4], and the emerging of wire-delay issues which caused communication energy cost to be greater than computation energy cost [2]. In an attempt to overcome these difficulties and overtake the *impasse*, processor manufacturers embraced the replication of components as the main form of technological evolution. Mainly, multiple *cores* of execution have been shipped into processors, which are now *multi-cores* by default. A core works like a traditional processor and can have specific connectivity, and resource sharing (e.g., caches), with other cores in the same processor chip. *Parallelism* became the

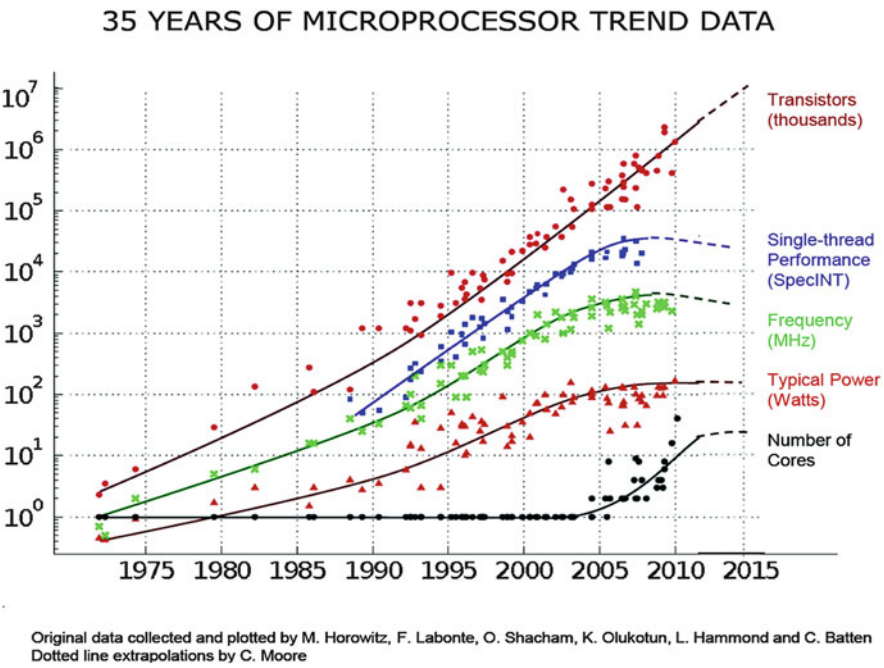


Fig. 1 Historical trend of microprocessors' technological evolution. Single-core frequency has stopped in 2004, and single-core performance has obtained only marginal improvement since then

driving force in the industry, and all the signals currently suggest that there will be no inversion of the trend anytime soon, as depicted by Fig. 1.

From the end user point of view, the parallel revolution did not always bring him/her the performance wonders promised. On the contrary, in some cases, a simplified design of the parallel cores with respect to the highly complex single-core processors of a previous generation caused users' programs to perform even slightly worse than before. In general, in order to exploit the aggregate computational power of multi-cores, a program needs to be reorganized in concurrent, possibly independent units of execution.

As confirmed from the data in Table 1, nowadays, processors are more and more parallel. Apart from multiprocessors (desktop and server) having reached about tens of cores, GPUs and so-called *many-core* devices can count on even thousands of cores (e.g., 5120 for the NVIDIA Tesla V100 board), thus pushing the hardware parallelism further. The tendency, however, is not limited to the *desktop* world, and also mobile and embedded devices are evolving in the direction of an ever-increasing parallelism. It is now common to have smart-phone SoCs with eight cores, plus a powerful GPU, DSPs, and even some purpose-specific processing unit (e.g., for deep learning applications). So, it is not an isolated phenomenon, but rather a general trend in the industry and the main source of innovation in hardware

Table 1 General features of some Intel processors, NVIDIA CUDA-enabled GPUs, and Snapdragon System on Chip (SoC) over time

Model	Launch year	Cores (threads) [CPUs] SM (CUDA cores) [GPUs]	Frequencies GHz (turbo)	Notes
Intel Pentium-4	2000	1 (1)	1.3–3.8 (–)	Desktop CPU
Intel Pentium-4 NW	2002	1 (2)	3.2–3.8 (–)	Desktop CPU
Intel Core 2 Duo	2007	2 (2)	1.86–3.33 (–)	Desktop CPU
Intel i7 Bloomfield	2008	4 (8)	2.13–2.66 (3.3)	Desktop CPU
Intel i7 Coffee Lake	2017	6 (12)	3.2–3.7 (4.3)	Desktop CPU
Intel Xeon E7-v4	2016	24 (48)	2.0–2.4 (3.4)	Server CPU
NVIDIA GeForce 6xx	2012	16 (512)	0.5–1.0	PCIe GPU (4 GB RAM)
NVIDIA GeForce 9xx	2015	Up to 96 (3072)	0.9–1.17	PCIe GPU (12 GB RAM)
NVIDIA Tesla V100	2017	160 (5120)[a]	1.38	PCIe GPU (16 GB RAM)
Snapdragon 820E	2018	4+4[b]+DSP+GPU(256)	2.2 (0.7 GPU)	Mobile SoC

The increase of core number is sustaining Moore's law. Clock frequency has substantially saturated more than 10 years ago. GPUs are increasing their memory capacity to exploit local and massively parallel computations

[a]Plus 640 Tensor Cores for AI deep learning and inference

[b]Customizable cores

manufacturing at the present day. For this reason, it is completely reasonable to think that also the upcoming IoT devices will join this trend.

2 Parallel Programming in CPSs

As specified before, the only way to harness the power of the modern, parallel architectures is to explicitly design applications as composed by concurrent units that can execute in parallel. For this reason, parallel programming techniques should be central in education so to give the programmers of tomorrow a fundamental tool that is gaining even more importance during the years as CPSs spread across the world and the demand for reliable, efficient, cryptography-based security grows.

Identifying concurrent activities in applications and programming their execution in parallel are however not trivial because it requires an unprecedented multidisciplinary approach encompassing also operating system interaction, synchronization between parallel activities, non-determinism in the execution interleaving of the different cores, etc. Since years also other technology-induced issues must be taken into account for effective parallel programming. In fact, Fig. 2, taken from a famous Bill Dally's Keynote in 2010, witnesses a paradigm change in computer engineering. Specifically, it shows that the energy to perform a double-precision operation in a processor (20 pJ) is 10× smaller than the energy needed to move the operands from halfway the typical size of the chip (256 pJ) and 50× smaller than moving them

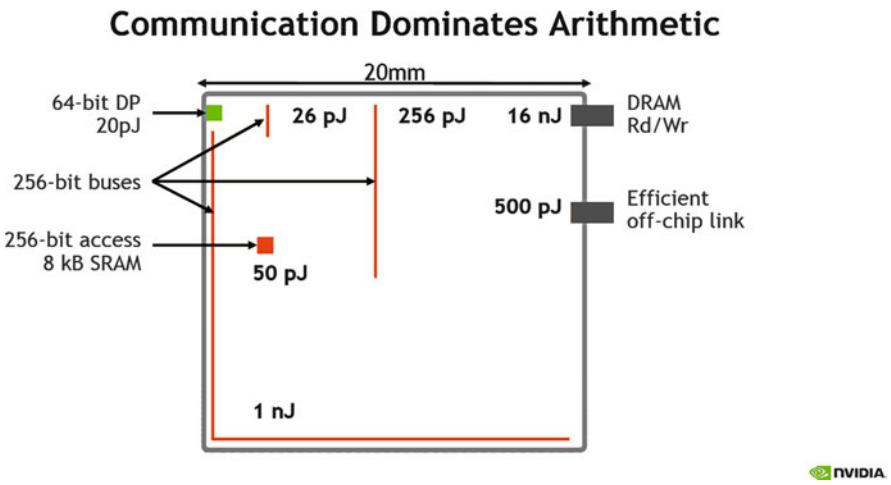


Fig. 2 Comparison between the energy needed to perform a 64-bit double-precision operation (20 pJ) and the energy for moving the operands at various distances on-chip and off-chip (circa year 2010). Since years, communication energy cost exceeds by far the computation cost [2]

from the opposite chip corner (1 nJ). Therefore, nowadays we need far more energy to bring operands to the cores (across the chip) than to perform the operation on them. As energy is tightly linked to time at the technological level, this means that an increasing amount of time is needed to bring data around the chip compared to traditional processors. This time is far bigger than the time to perform many operations: in fact around 30 cycles can be observed to communicate corner-to-corner on-chip and only one cycle to perform an integer addition in the functional unit. In parallel programming this fact hampers also the possibility to quickly coordinate processing cores on-chip within a parallel computation. Essentially, for efficiency and performance scalability in nowadays computing systems, *elaboration must be as local and parallel as possible*.

In this chapter we will go through the analysis of two cryptographic algorithms in order to devise possible parallelization strategies that can improve their performance on parallel architectures. As we will see, the intrinsic sequential nature of these algorithms makes this approach quite challenging, but nonetheless we will highlight interesting performance margins to be exploited.

Specifically we will focus on parallel approaches to:

- Modular exponentiation: m-ary and exponent slicing approaches
- Karatsuba multiplication

These cases are significant in cryptography because Karatsuba multiplication [7] and derived approaches are typically used to efficiently multiply multi-precision numbers (hundreds/thousands bits long). Then, modular exponentiation is the operation on which RSA [15] public-key cryptographic protocol is based.

We will adopt an educational approach to present, highlight, and discuss such case studies. This will allow us to underline the key features, along with the opportunities and limitations, of different parallel programming approaches in this specific application domain. As a consequence, we will favor clarity and focus on principles without looking for the ultimate performance/optimizations in the algorithms and in the parallelization strategies. For the same reason, we are using algorithms that are known to be vulnerable to both cache and timing side-channel attacks [6, 9, 14], but their simplicity allows us to explain some parallel programming concepts more easily. For sure, the highlighted principles will be applicable in general and not limited to the cases used for their explanation.

2.1 Experimental Methodology

We have implemented the described algorithms in modern (i.e., post 2011) C++ programming language and compiled the programs with g++ 6.3 compiler with -O3 optimization flag and selecting C++14 version. We relied on GMP [17] library for the implementation of arbitrary precision numbers and for measuring some reference performance of the considered algorithms.

Performance was evaluated considering repeated executions with enough iterations to amortize warmup effects in processor internals and caches. Moreover, the execution time of each experiment was tuned to be exceeding some hundreds of seconds so that operating system time quantum and (re-)scheduling would not significantly affect results.

Experiments were performed on machines running Linux Debian versions 8 and 9.

2.2 Modular Exponentiation

Modular exponentiation (see Eq. (1)) is the key mathematical operation that allows the implementation of RSA cryptographic protocol for both encryption and decryption operations. Exponentiation applies to numbers (base and exponent) big as the key size, which depends on the desired level of security. Even if typically the public exponent is small (e.g., 65,535), bases and private exponents are as big as the key size. Currently key sizes deemed secure are at least 2048/4096 bits long, and in near future such sizes are expected to grow [3]:

$$M^e \bmod n \tag{1}$$

The square-and-multiply, or binary method, allows the implementation of modular exponentiation through repeated modular squares and multiplications of the partial results with the base. The method relies on the binary expansion of the

Algorithm 1: Square-and-multiply modular exponentiation algorithm

Input: M, e, n
Output: $C = M^e \bmod n$
begin
  if $e_{k-1} == 1$ **then** $C := M$ **else** $C := 1$;
  for $i = k - 2$ ***downto*** 0 **do**
    $C := C \cdot C \bmod n$;
    if $e_i == 1$ **then** $C := C \cdot M \bmod n$;
  end
  return C ;
end

exponent $e = (e_{k-1}, e_{k-2}, \ldots, e_1, e_0)$, having k bits, and on the observation that $e = \sum_{i=0}^{k-1} e_i 2^i$. Therefore, the exponentiation can be calculated according to Algorithm 1 [8].

Essentially, the core part of the algorithm scans exponent bits from the most significant one, and it performs a *square* on each partial result. Then, depending on the bit being "1" or "0," it performs or not, respectively, an additional multiplication with the base M.

These experiments were run on a dual-Xeon machine with 64 GB RAM, evenly split between the sockets close to each processor. Processors were Xeon E5-2650 v2 operating at 2.60 GHz (3.4 GHz turbo) and featuring 8 cores (16 threads, thanks to Hyperthreading SMT[1] [18]). Each core has 32 KB, separated instruction and data, private level-1 (L1) caches, and a private 256 KB level-2 (L2). Each processor has a 20 MB level-3 (L3) cache shared between all the cores, 64 GB RAM machine.

2.2.1 *m*-ary Approach

The m-ary method is the generalization of the binary one. While in the latter the scanning of the exponent is done one bit at a time, in the former the bits are visited in groups of $\log_2(m)$ at a time. So, for $m = 2$, the two methods coincide.

The method is described in Algorithm 2 [8].

Here, the k bits of the exponent are decomposed in s groups of r bits each so that $r = \log_2(m)$ and $k = sr$. A padding of r—$(k \bmod r)$ zeroes may be necessary if r does not divide k. For each group i of bits, F_i represents the binary value of the group. For each possible value of F_i, i.e., $w = 0, 1, \ldots, m - 1$, excluded the trivial ones 0 and 1, the w power of M is pre-computed and stored. Then the k bits are scanned in groups of r from the most significant to the least significant. For each group, the partial result is raised to the $2^r = m$ power and multiplied by M^{F_i} modulo n.

[1] Simultaneous multi-threading.

Algorithm 2: m-ary modular exponentiation algorithm

Input: M, e, n
Output: $C = M^e \bmod n$
begin
  Compute and store $M^w \bmod n$ for $w = 2, 3, \ldots, m-1$;
  Decompose e into r-bit words F_i for $i = 0, 1, \ldots, s-1$;
  $C := M^{F_{s-1}} \bmod n$;
  for $i = s-2$ ***downto*** 0 **do**
    $C := C^{2^r} \bmod n$;
    if $F_i \neq 0$ **then** $C := C \cdot M^{F_i} \bmod n$;
  end
  return C ;
end

The main advantage of this method over the binary method is that it can reduce the number of multiplications performed. By increasing m, the number of steps necessary to scan the exponent decreases, and so does the number of multiplications, since there is at most one for each step. However, when increasing m, the number of pre-computed powers of M grows accordingly, and thus it is not trivial to guess the optimal value of m.

An important source of performance comes from the possibility to calculate the exponentiation by employing N_t-independent threads of execution. This can be done by expressing the exponent e as a sum of exponents $\widetilde{e}_j$ for $j = 0, 1, \ldots, N_t$, calculating the single exponentiations independently, and then multiplying the partial results together as Eq. (2) shows:

$$e = \sum_{i=0}^{N_t} \widetilde{e}_i \Rightarrow M^e \bmod n = M^{\sum_{i=0}^{N_t} \widetilde{e}_i} \bmod n = \prod_{i=0}^{N_t} M^{\widetilde{e}_i} \bmod n \tag{2}$$

The easiest way to construct N_t exponents $\widetilde{e}_j$ that sum to e is by dividing the bits in N_t different sets that cover the whole exponent, copying the bits of each set j in the same positions in exponent $\widetilde{e}_j$, and zeroing the other bits.

This is exemplified in Fig. 3 where the bits of the original exponent e are split into four exponents e_i using a *comb*-like approach. Each colored zone of the exponents has width $r = \log_2(m)$ bits. In this way four threads can calculate in parallel, and independently, $R_i = M^{e_i}$, $i = 0, 1, 2, 3$.

We consider a definition of the j-th set as composed by the bits in the i-th group of $r = \log_2(m)$ bits with index $i = j + hN_t$, where $h = 0, 1, \ldots$. Or, in other words, the groups of bits whose index i is such that $i \bmod N_t = j$. So, the j-th exponent $\widetilde{e}_j$ is defined as shown in Eq. (3):

$$\widetilde{e}_j = \sum_{i=0}^{s-1} \widetilde{F}_i 2^{ir}, \quad \widetilde{F}_i = \begin{cases} F_i & \text{if } i \bmod N_t = j \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

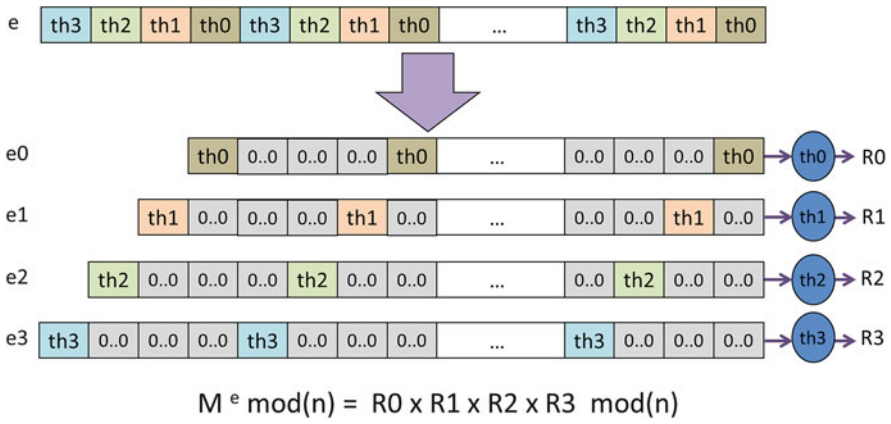


Fig. 3 Sample parallelization scheme for m-ary modular exponentiation with four threads

This algorithmic approach requires a limited computational overhead both (a) in the setup of the parallel computation and (b) after the parallel execution to obtain the overall result R. In fact, the former requires to allocate space and fill the $\widetilde{e}_j$ exponents for the threads, which is relatively fast being $O(k)$. Then the final step is performed by the main thread, once all parallel threads have finished their work, and consists in three modular multiplications between the R_j partial results to obtain R. Both these conceptual steps require a *small* time compared to the modular exponentiations $M^{\widetilde{e}_j}$ performed in parallel. It is interesting to underline that m-ariety and parallelization degree are independent design variables of the algorithm implementation and can be tuned independently to maximize performance on a specific platform.

From the performance point of view, however, the expected speedup of the parallel m-ary exponentiation compared to the serial one cannot be expected to be huge. In fact, Fig. 3 highlights that the *work* to be performed by each thread appears very similar to the one of a serial implementation, i.e., one thread doing the whole M^e computation. And, in particular, the third thread dealing with the most significant bits of e actually manages an exponent which is as long as e itself. The performance improvement can derive from the fact that each thread, by construction, deals with exponents $\widetilde{e}_j$ which exposes fewer 1 bits than e: statistically, half of the bits of the colored parts. Therefore, despite performing a modular squaring for each bit in $\widetilde{e}_j$, the number of multiplications performed by each thread is about $\frac{k}{N_t \cdot 2}$ instead of $\frac{k}{2}$.

Then, m-ary method itself requires a pre-computation step for calculating the m $(2^r - 2)$ nontrivial[2] multiples of base M, which need to be conceptually available before the exponent scan from the threads. For this reason, pre-computation

[2] $M^0 = 1$ for every non-zero M, and $M^1 = M$.

overhead exponentially increases with m, so we have explored the possibility to work in parallel also on the pre-computation of the m powers of M. Basically, we split such work between some *pre-computation threads* and calculated via GMP/MPIR functions.

Figure 4 shows the achieved performance of various implementations of a parallel m-ary modular exponentiation for key sizes of 2018, 4096, and 8192. Performance is measured in elapsed time to complete a computation, so less is better. The green area is bounded by square-and-multiply performance, so every sample in this area performs better than the basic implementation. For each curve, the number of threads employed in the main computation is highlighted (thr) along with the threads employed in the pre-computation of the powers of M (thPr). These threads are started together with the others and are used to calculate the factors necessary in the multiplications. GMP/MPIR indicates the achieved performance by the reference highly optimized, sequential, GMP-library implementation.

In Fig. 4a, relative to 2048-bit operands, the best configuration is four threads, four pre-computation threads, and $m = 2^2$. The corresponding performance improvement is 26% with respect to square-and-multiply. As a side observation, we can see that threading, especially in pre-computation, allows the scheme to be more performance-stable when m-ariety is varied. Specifically, pre-computation parallelism is required to support exponentiation parallelism.

In Fig. 4b 4096-bit performance is shown. In this case the best configuration is eight threads, four pre-computation threads, and $m = 2^3$, for a performance improvement of 26%, again, with respect to binary method. However, also four threads perform very close to such one.

Finally, Fig. 4c corresponds to the biggest key size measured (8192) and shows that the performance gain is up to 23.5% with respect to square-and-multiply when eight threads, four pre-computation threads, and $m = 2^2$ are employed. Again, also four threads can reach almost the very same performance.

All these figures show that elapsed time grows exponentially when *big* values of m are used. This is due to the fact that the powers of M that need to be pre-computed grow exponentially with r, which increases linearly on the x-axis in the figures. So, for big ms, pre-computation dominates the whole calculation time.

A possible solution is to avoid calculating the powers of M in advance. In fact, as m grows, the probability of needing each power of M in the exponent decreases, and thus there is no guarantee that all of them are actually needed. We thus explore a variation of the illustrated method in which there are no pre-computation threads. When a thread needs a power of M, it accesses the shared pre-computation table and uses the required entry, if present; otherwise it calculates it *on-demand* and put it in the table for future uses by itself or other threads.

Figure 5 shows that on-demand calculation generally improves performance compared to preliminary computation of all powers, shown in Fig. 4b. In this case the best configuration delivers a 27% improvement with only three threads and $m = 2^6$ or four threads and $m = 2^4$. As expected, the pre-computation overhead is far better amortized even for bigger m-ariety. Interestingly, performance becomes less sensitive than before to thread count, making this approach more flexible to

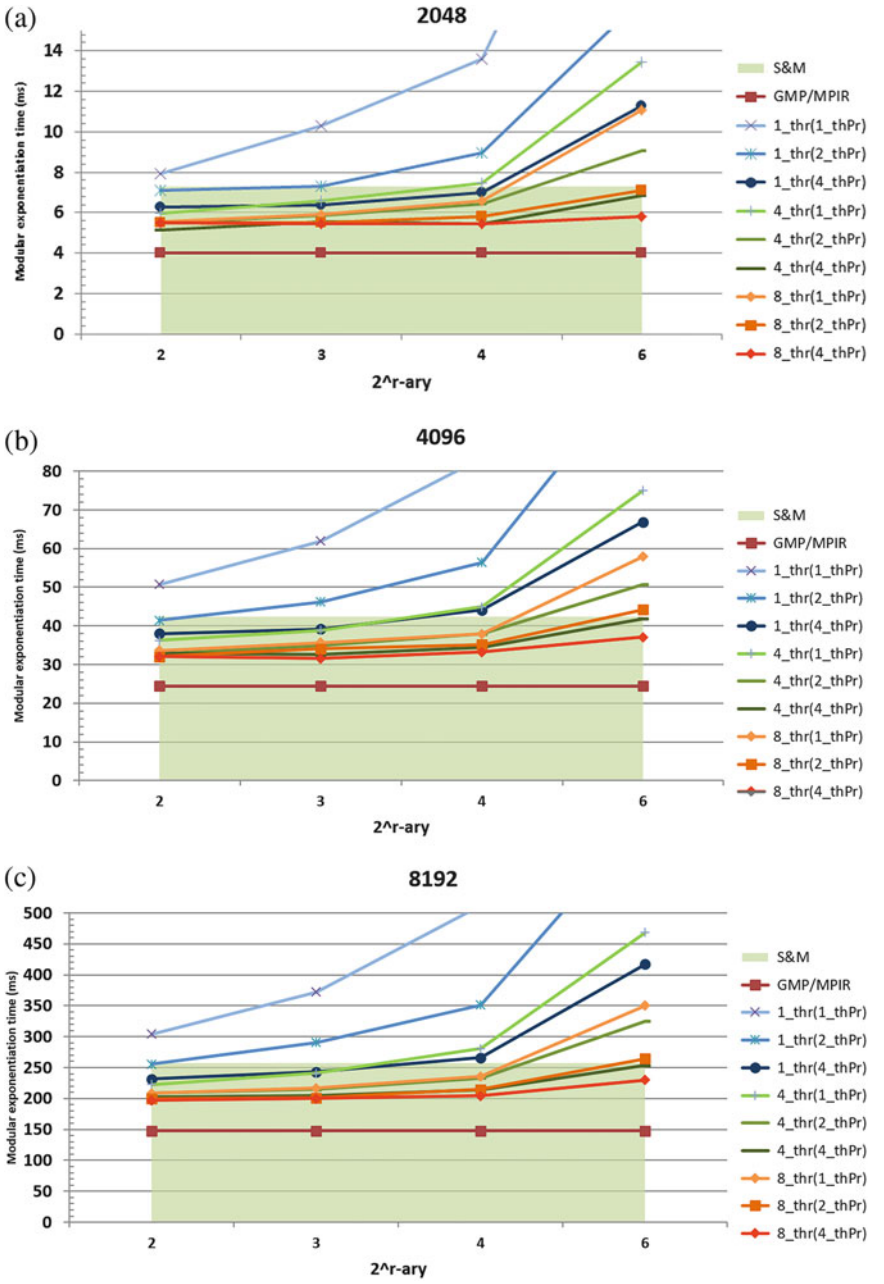


Fig. 4 Performance of m-ary modular exponentiation for different thread count and m-ariety. 2048-, 4096-, and 8192-bit key sizes. The top limit of the green zone represents the speed of the plain sequential *square-and-multiply*, while X_thr(Y_thPr) data represents a version with X computation threads with Y threads dedicated to pre-calculate the required M multiples. GMP/MPIR is the highly optimized sequential modular exponentiation in GMP/MPIR libraries. (**a**) 2048-bit. (**b**) 4096-bit. (**c**) 8192-bit

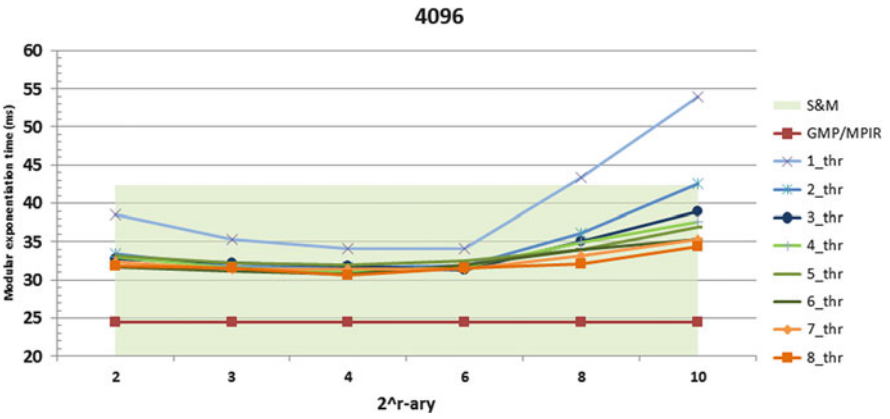


Fig. 5 Performance of parallel m-ary modular exponentiation with 4096-bit operands and on-demand calculation of the required powers of M

accommodate on processors with different number of cores. From two threads onward, performance increases significantly with thread count only for m-ariety bigger than 2^6.

From the parallel programming standpoint, this implementation needs to impose synchronization between the working threads only when they attempt to pre-calculate the same multiple of M. In fact, if one thread needs a specific multiple M^u when another thread is already pre-calculating it, we need to stop it and let it wait the result from the other one for both consistency of the table entries and to avoid performing redundant work. Paradoxically, it could be interesting to analyze an additional variation of the scheme, where the second thread calculates the same power even if the first is already calculating it. In fact, due to wire-delay issues on-chip and inter-thread synchronization delays, it may be that this approach can be slightly faster because the threads never stop. Obviously only the first who computes the given power must update the shared table. The other(s) must not update the table and simply discard the power that they used *locally*. We leave the evaluation of this variant to the interested reader.

2.2.2 Slicing Approach

Exponent "slicing" is a pretty intuitive way for approaching work splitting in modular exponentiation so that parallel threads can work on independent exponent *slices* in parallel. The binary expansion of the exponent is divided in N_p partitions so that $e = (e_{k-1}, e_{k-2}, \ldots, e_1, e_0) = (p_{N_p-1}, p_{N_p-2}, \ldots, p_1, p_0)$. By denoting with B_j the number of bits of the j-th partition, the following equation holds:

$$e = \sum_{i=0}^{k-1} e_i 2^i = \sum_{j=0}^{N_p-1} \sum_{i=0}^{B_j-1} e_{i+s_j} 2^{i+s_j} = \sum_{j=0}^{N_p-1} \widetilde{e}_j \tag{4}$$

where s_j is the index of the least significant bit of the j-th partition or expressed in formula:

$$s_j = \sum_{h=0}^{j-1} B_h \tag{5}$$

So, the exponent can be expressed as a sum of addends $\widetilde{e}_j$, each one determined by a single partition, and thus the slicing method is eligible for parallelization as done before. It's also important to note that the binary expansion of each addend is just the *few* contiguous bits of the partition followed by a number of zeroes that amounts to the sum of all the bits of all the previous partitions. When applying the square-and-multiply approach to each exponent $\widetilde{e}_j$, modular multiplications are performed in the initial non-zero portion only, while the subsequent stride of contiguous zeroes induces only repeated modular squarings.

The number of non-zero bits of each partition B_j can be chosen in various ways. A naïve but very intuitive approach is to divide the exponents in chunks of the same size, so $B_j = k/N_p$ for $j = 0, 1, \ldots, N_p - 1$, as exemplified in Fig. 6. This leads to a quite unbalanced distribution of operations among the threads. In fact, the exponent is bigger when hosting the bit partitions on the most significant side of the original exponent. All exponents $\widetilde{e}_j$ would expose the same average number of multiplications but with a very different number of squarings. A more balanced approach would aim to reduce B_j as j approaches $N_p - 1$ so that it would require less multiplications to compensate for the increased number of squarings.

In this sense, Lara et al. [10] present an optimal method to calculate the partition sizes depending on N_p and considering "0" and "1" values as having the same probability to appear in the binary expansion of the exponent. Furthermore, they propose also a way to determine the optimal number of partitions N_p.

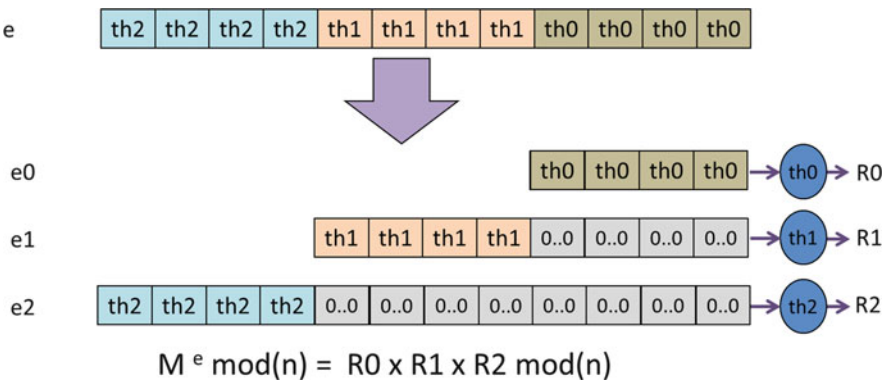


Fig. 6 Sample parallelization scheme exploiting exponent *slicing* in modular exponentiation with three threads

From the parallel programming viewpoint, the preparation of the exponents $\widetilde{e}_j$ to enable the parallel working threads has a complexity which is linear with the number of bits of the exponent and with the number of threads to activate, as in the m-ary approach described in Sect. 2.2.1. In particular, the exact overhead can be expected to be even slightly smaller because (a) some exponents are fairly smaller as their average size is $k/2$ and (b) the non-zero part of each exponent can be filled from the original exponent with a contiguous copy of processor words and bits, exploiting more likely its cache locality. The final *reduction* of the partial results into the final result $R = M^e$ works exactly as in the m-ary method, i.e., requires $N_p - 1$ modular multiplications. For these reasons, the measured performance differences can be reliably attributed to the core part of the parallel exponentiation algorithms.

Performance-wise we can expect a speedup over the serial version similar, if not smaller, than in the m-ary case, especially for evenly split exponents. In fact the overall result can be computed only when the thread dealing with the longest exponent has finished. Such exponent has the same length as the original one and features the same average number of 1 bits, which require multiplications, as in the longest exponent in the m-ary approach. The main structural difference between such exponents is that in the *slicing* method, all ones appear at the start of the exponent scan, while in the m-ary they appear periodically interleaved with some groups of zeroes.

Figure 7 shows the achieved performance of parallel modular exponentiation with different exponent partitionings and operand sizes. All the partitionings have been obtained by varying N_p and by determining each B_j as explained in [10], so to balance optimally the work between threads.

Slicing into three or four parts (threads) is typically enough to get the maximum benefit. For 1024- and 2048-bit number size, the *slicing* approach reaches up to

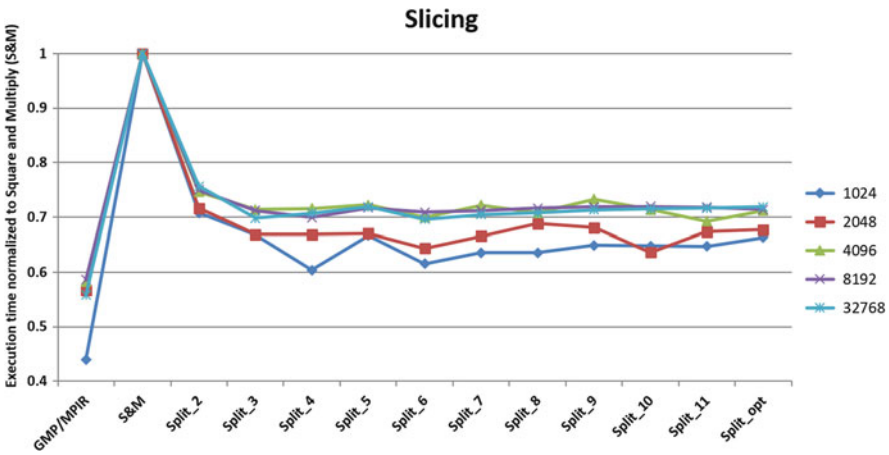


Fig. 7 Performance of parallel modular exponentiation with slicing method in correspondence of different exponent partitionings and operand sizes

35–40%, while it reliably reaches 30% for bigger number sizes. Then, in our platforms, we observe that the optimum splitting does not deliver the maximum performance improvement.

Comparing the performance of the *slicing* approach to the *m*-ary method, we can observe that, counterintuitively, the first is marginally faster than the latter and that it is far more robust in the speedup over the serial version when considering different numbers of threads and for the various key sizes.

At the same number of threads, given that the performance of each of the two methods is limited by the execution time of the thread that is assigned with the longest partial exponent and that the two exponents differ only in the localization of the non-zero bits, the performance differences between the two approaches are due to two combined effects: (a) in the *m*-ary case, the pre-computation of the needed powers can bias the execution of *m*-ary, even if it is then amortized when reusing the same power, and (b) the sequence of successive zeroes is only one, and far longer, in the *slicing* method. The net effect of (a) can be considered in favor of *m*-ary especially in the on-demand variation because only the required powers are computed and potentially reused along the exponent. So the reason why the *slicing* method is quicker can be attributed to the faster execution of the repeated modular squarings. This can be justified considering how modern processors work. In fact, a repeated small sequence of instructions executed many times in a row (i.e., many successive squarings in *slicing* method) performs faster than the same number of squarings interleaved with modular multiplications in the *m*-ary approach, mainly due to better cache locality and branch predictor accuracy.

2.2.3 Conclusions: Modular Exponentiation

Overall we have illustrated two parallelization approaches for the modular exponentiation algorithm, *m*-ary and exponent *slicing*, with also an *on-demand* variation in case of the *m*-ary one. We have discussed some aspects related to the interaction between the parallelization in itself and the specific parallelized algorithms. Furthermore, we can conclude that the *m*-ary approach can achieve up to about 26% speedup over the serial version using four threads for 2048- and 4096-bit numbers, while about 23% for eight threads in the 8192-bit case. A variation that allows to pre-compute powers only when needed, and once for all threads, shows to be very robust against thread count and *m*-ariety and reaching reliably around 27%. Then the *slicing* method appears the most robust and the fastest among the considered ones, reaching 35–40% speedup for 1024- and 2048-bit key sizes and reliably 30% for bigger ones and using small number of threads.

Considering that modular exponentiation is not trivially parallelizable and that, conversely, it has sequential semantics intrinsically in the algorithm, the achieved speedup can be regarded as interesting for speeding up modular exponentiation in devices that, after all, typically have multiple cores available. In fact, if we are dealing with a datacenter implementing RSA algorithm, we can reduce by 30% the number of machines required for the number of concurrent connections. At the

other extreme of the architecture spectrum, one can think of a smartphone that can perform a session-key exchange 30% faster, or in the same time but requiring 30% less clock rate.

2.3 Karatsuba Multiplication

Many cryptographic algorithms rely on the modular multiplication of *big* numbers ($x \cdot y \bmod n$) and in turn on multiplication itself. For instance, as seen in the previous section, it constitutes the core part of modular exponentiation.

For this reason, multiplication is typically used repeatedly in cryptographic routines, so its efficient implementation is of the utmost importance. A number of research studies and implementations have been proposed during the years [1, 13] for high-performance multiplication of *big*, multi-precision numbers. Here in the following, we present a parallel implementation of the Karatsuba approach [7].

When multiplying two k-bit numbers x and y, the result $x \cdot y$ is a $2k$-bit number, which is reduced back to k bit after modulo calculation. The complexity of the pencil-and-paper algorithm for multiplication grows quadratically with k, $O(k^2)$ asymptotic complexity. Conversely, Karatsuba method reduces the big-O complexity down to $O(k^{1.583})$. The scheme is based on a *divide-et-impera* strategy along with the reuse of some partial results.

Specifically, if m is a number such that $0 < m < k$, the two operands can be expressed as:

$$x = 2^m x_1 + x_0 \tag{6}$$

$$y = 2^m y_1 + y_0 \tag{7}$$

And their product can be expressed as:

$$x \cdot y = (2^m x_1 + x_0)(2^m y_1 + y_0) = 2^{2m} x_1 y_1 + 2^m (x_1 y_0 + x_0 y_1) + x_0 y_0 \tag{8}$$

So, it is possible to express the product of two numbers in terms of multiplications and additions of smaller elements. By choosing $m = k/2$, the operation is balanced and the four products are k-bit terms each. Also, the multiplications for powers of two can be obtained as simple left-shift operations in binary arithmetic.

However, from the complexity point of view, this is not an improvement as the asymptotic complexity to perform a k-bit multiplication ($\approx k^2$) is the same as the one required for performing four $k/2$-bit operations ($\approx 4 \cdot (k/2)^2 = k^2$).

So, Karatsuba algorithm introduces a clever rewriting of the $m = k/2$ multiplications, as highlighted in Eq. (9):

$$\begin{aligned} t_0 &= x_0 y_0 \\ t_2 &= x_1 y_1 \\ t_1 &= (x_0 + x_1)(y_0 + y_1) - t_0 - t_2 = x_1 y_0 + x_0 y_1 \\ x \cdot y &= 2^{2m} t_2 + 2^m t_1 + t_0 \end{aligned} \tag{9}$$

The number of multiplications between m-bit operands is reduced from four to three, but the number of additions is increased from three to five. Since the complexity of additions grows linearly with m, the whole operation gains in efficiency both in practice and, asymptotically, when m grows.

Coming to possible parallelization strategies, we observe that the calculation of t_i terms is independent of each other, and so a natural way to express Karatsuba in parallel is by using three threads, the main one with two other helper threads. Once all the t_i terms have been calculated, they can be properly shifted and added together to obtain the final result, as visualized in Fig. 8.

As in the modular exponentiation case, let's analyze whether the parallelization scheme can be reasonable or not. Essentially we must evaluate the overhead of activating, i.e., spawning, the working threads and of composing the final result out of the partial results of each thread.

Each thread performs a multiplication, and two of the three can immediately use the original operands, the high and low part of x and y, respectively. The operands of the third thread must instead be prepared. So, memory space for the additional numbers must be allocated and the numbers filled according to Eq. (9). In this, the most time-consuming operation is the memory allocation, if such temporary space is not maintained available across successive multiplications. The required subtractions are the remaining operations to be done and, being linear in the number size, they are far faster than the multiplication itself. When the threads complete (see Fig. 8), the three partial results need to be merged through scaling, i.e., bit

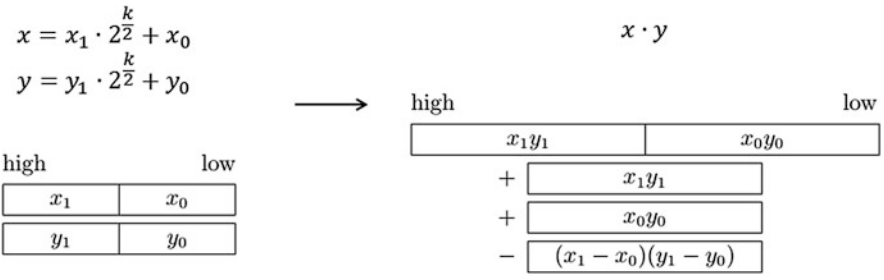


Fig. 8 Karatsuba multiplication computes a k-bit multiplication using only three $k/2$-bit ones, reducing asymptotic complexity and execution time

shifting and actually almost zero-complexity word-level shifting, and additions. All such operations are fast and linear with k. Therefore we can conclude that, in principle and considering the asymptotic complexity, this Karatsuba parallelization is promising.

Then, in order to achieve a speedup in practical cases, that is, considering interesting number sizes (thousands bits), we have to consider such overheads also in terms of time. There is another implicit overhead that must be considered: how fast the operating system allows the program to spawn new threads compared to the multiplication time. In fact such time is typically in the critical path of the parallel processing and could potentially decrease the theoretical speedup.

We implemented the sequential Karatsuba algorithm, where all three multiplications are performed in the same main thread, as a reference for the parallel versions. Table 2 shows the execution time of the sequential Karatsuba algorithm (Kara_seq), as well as the one of the highly optimized sequential multiplication of GMP library (GMP_mul), for various bit sizes on an Intel E5-2650 v2 machine. These values need to be compared to the spawn and join time of a thread. Incidentally, we observe that the GMP_mul algorithm is almost three× faster than our simple implementation of Karatsuba for 1024-bit numbers, but only 13.7% faster in case of very big numbers (65,536-bit). This is probably due to some optimizations that we are missing in our purposely simplistic code, but the overall asymptotic complexity appears to be similar.

Table 3 shows the measured spawn and join times of a thread on three Linux machines. Specifically, we have repeatedly generated and joined an *empty* thread from the main one and measured the required average elapsed time. We have also evaluated the quotes for thread spawning and joining time, separately.

Table 2 Execution time (µs) of sequential GMP-library and Karatsuba multiplications for various bit sizes

Bits	GMP_mul (µs)	Kara_seq (µs)
1024	0.616	1.727
2048	0.874	2.291
4096	2.579	4.070
8192	7.865	9.558
16,384	21.161	26.396
32,768	55.902	67.858
65,536	153.327	174.204

The simple handcrafted Karatsuba version is 2.7× slower than GMP one for 1024-bit numbers but only 13.7% slower for 65,536-bit

Table 3 List of the times needed to spawn and join a thread on three different Linux machines

Processor	Spawn (µs)	Join (µs)	Total (µs)
i7 2600	6.5	9.5	16.0
Xeon E5-2650 v2	4.3	5.8	10.1
i7 6800K	4.4	6.0	10.4

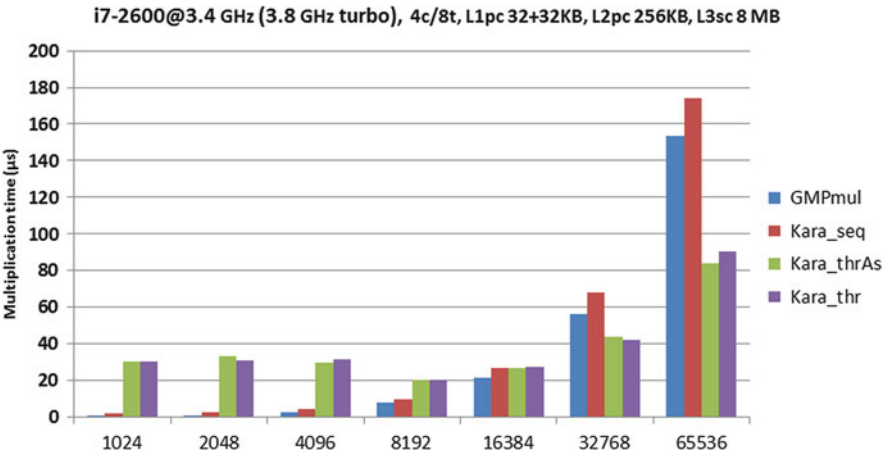


Fig. 9 Execution time (µs) of various parallelization approaches of the Karatsuba multiplication algorithm on an Intel i7-2600 machine. Karatsuba sequential (*Kara_seq*) and two variations of the parallel Karatsuba (*Kara_thrAs* and *Kara_thr*) are shown. Furthermore, the sequential highly optimized GMP multiplication (*GMPmul*) is also indicated for overall reference

From the comparison between Tables 2 and 3, we can see that the thread management overhead alone is not only comparable with the expected execution time of each of them, but most importantly, it is even dominant for number sizes smaller or equal to 8192 bits.

This observation is confirmed in Fig. 9 which highlights that our simple parallel Karatsuba approach (*Kara_thrAs* and *Kara_thr* series) improves over the serial version (*Kara_thr*) by about 40% and 50% only for 32,768-bit and 65,536-bit numbers, respectively, on an Intel i7 2600 machine. For 16,384-bit ones, performance is the same as in the sequential version. For smaller numbers, parallel Karatsuba exhibits a slowdown as the execution is dominated by the unavoidable thread spawn-join overhead. In fact, the latter last about 4×–6× more than the parallel multiplications performed. From a practical perspective, the speedup for so big numbers, and thus key sizes, is quite beyond the current and short-term security requirements, and so it is not exploitable. On the other reference machines, results are slightly different numerically, but the trend is exactly the same; therefore we are not showing them for space reasons. The core part of the source code of *Kara_seq* and *Kara_thrAs* versions is shown in "Appendix: Sequential and Basic Parallel Code for Karatsuba" at the end of the chapter.

As a sidenote, *Kara_thrAs* and *Kara_thr* represent the same conceptual parallelization approach where the parallel activities are implemented via `std::async` and `std::thread` [5] C++11 classes, respectively. The outcoming performances are very similar as both are implemented in terms of the same operating system threads.

Based on these results, we can conclude that a *standard* parallelization strategy that generates the required threads on-demand does not work very well on reasonably sized keys because of the intrinsic time needed to generate and deallocate such

threads even if, from the application point of view, both work splitting and partial result recomposition steps are negligible compared to the parallel execution and would be compatible with an effective parallelization.

Given this, we have then tried to apply more advanced parallel programming approaches for reducing the thread management overhead, and we have done it in incremental complexity steps.

The first approach is based on a thread pool that works as follows:

- The so-called worker threads are always active and wait on a condition variable in an *infinite loop*.
- The main thread fills the input structures of the worker threads and triggers them.
- They compute the partial multiplications, store the results in a shared data structure, and block again.
- The main thread retrieves the partial results or block until they are produced.

The blocking/triggering of the worker threads is performed using condition variables and mutual exclusion (mutex) thread synchronization constructs, within C++11 standard, and is implemented interacting with the operating system scheduler and user-space semaphores.

We can expect this approach to improve over the previous one if the thread synchronizations are significantly faster than thread spawn/join operations. We measured that the time to unblock a thread in this scheme, and on the same considered machines, is within 2.3–4.2 µs. Therefore, it is about 2.5×–4× faster, but it is still comparable with the execution time of multiplications on reasonably sized numbers (2048–4096-bit). So we can expect to, at most, break even with the sequential Karatsuba only when multiplying 8192-bit numbers and achieve speedups only for bigger numbers.

This trend is confirmed by the *Kara_InfThr* bars shown in Fig. 10 as values normalized to the sequential Karatsuba version for our Xeon machine. Interestingly, beyond 16,384 bits, the parallel version outperforms even the highly optimized sequential GMP multiplication.

Lastly, Fig. 10 shows an additional bar, *Kara_infThrLF*, which corresponds to a further refinement of the *Kara_infThr* implementation. Specifically, it reduces the mutex and condition variable wake-up costs by using lock-free [12] data structures for inter-thread synchronization. Here the programming effort is higher than in the previous cases, but the investment pays off as it allows a performance improvement of 17%, 33%, and 41% for 4096-, 8192- and 16,384-bit keys, respectively.

In conclusion, thanks to these advanced parallel programming techniques, these results demonstrate significant acceleration of Karatsuba multiplication for security levels which are very interesting nowadays and in near future applications (i.e., 4096 bits and soon 8192 bits).

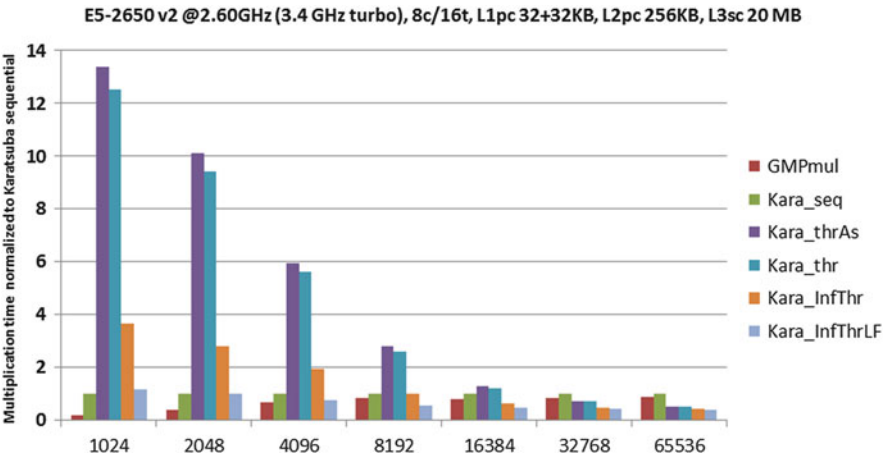


Fig. 10 Execution time of various parallelization approaches of the Karatsuba multiplication algorithm on an Intel Xeon E5-2650 v2 machine, normalized to sequential Karatsuba (*Kara_seq*) performance. Four variations are considered: *Kara_thrAs*, *Kara_thr*, which use the same approach and differ only in the C++ parallelization construct, `std::async` vs. `std::thread`. *Kara_InfThr* employs worker threads in an infinite loop for reduced start-up time, and *Kara_InfThrLF* further reduces runtime overhead using lock-free thread synchronization. The sequential highly optimized GMP multiplication, *GMPmul*, is also indicated as an overall reference implementation

3 Conclusions

Cyber-Physical Systems are emerging as very crucial assets of our future society in order to achieve highly efficient and effective services and systems within production, transportation, energy management, homes, social life, and business in general. These systems will be distributed, heterogeneous, and connected to the Internet and toward a multitude of application-level protocols to promote integration. Securing these systems will be of the utmost importance and will require to operate on different kinds of devices and, ultimately, processing elements, within desktop and server computers down to embedded, personal, wearable, and IoT devices. In this chapter we have highlighted the evolution of processors toward parallel architectures composed of an increasing number of cores, which contribute to the overall aggregate computational potential of the chip. However, to harness such increasing aggregate processor speed, applications must expose independent activities to be executed as much as possible in parallel by the available cores. Unfortunately, cryptographic algorithms, and the underlying mathematical primitives, typically expose a sequential nature as the final result is obtained in steps where each step depends on the previous one like in the sequence of multiplications and squarings in the square-and-multiply modular exponentiation. We have discussed and analyzed the possibility to parallelize a pair of mathematical operations which are very relevant in cryptography: modular exponentiation (e.g., used in RSA) and Karatsuba multiplication of multi-precision numbers. We have compared two

possible approaches for parallelizing the *m*-ary and *exponent slicing* approaches for modular exponentiation, and we demonstrate to reach up to −30/−40% across various key sizes and for 3/4 threads.

Then we have investigated various parallel programming techniques, with increasing coding complexity, for multi-precision Karatsuba multiplication. We have shown improvements of −17% to −60% compared to the sequential version and, most importantly, in key size ranges which are interesting and useful for nowadays and near future security levels.

Overall, we hope to have made an interesting point for the promotion of education into parallel programming in general and, specifically, for the cryptographic algorithms needed to secure the services offered by Cyber-Physical Systems.

Appendix: Sequential and Basic Parallel Code for Karatsuba

In the following we show the base sequential Karatsuba code (*Kara_seq*) and the parallel version based on C++11 `std::async` asynchronous thread invocation that we used in our experiments. The purpose is to highlight the changes that need to be done for enabling a multi-threaded computation.

In Fig. 11 we show the base code that implements a Karatsuba step on the multi-precision numbers and then delegates the multiplication of the $k/2$ split numbers to the native GMP multiplication algorithm, here accessed through overloaded *

```
mpz_class karaMul(mpz_class const& x1, mpz_class const& x2)
{
    assert( x1.get_mpz_t()->_mp_size == x1.get_mpz_t()->_mp_size ); // runtime check

    auto const part1 = splitBigNum_limb(x1);    // part1 is a std::pair{ high_bits1, low_bits1}
    auto const part2 = splitBigNum_limb(x2);    // part2 is a std::pair{ high_bits2, low_bits2}

        //multiplication 1
    mpz_class x1Lx2L= part1.second * part2.second;
        //multiplication 2
    mpz_class x1Hx2H= part1.first  * part2.first;
        //multiplication 3
    mpz_class midTerm= x1Hx2H + x1Lx2L -
            (part1.first-part1.second) * (part2.first-part2.second);

    mp_bitcnt_t halfBits= (x1.get_mpz_t()->_mp_size/2) * sizeof(mp_limb_t) * 8;

        //computing final result
    mpz_class ret= x1Lx2L + (midTerm << halfBits) + (x1Hx2H << (2*halfBits));

    return ret;
}
```

Fig. 11 Basic sequential code implementing the Karatsuba step used as a reference for the discussed parallelized versions. We use multi-precision numbers from the GMPXX (GMP for C++) library (`mpz_class`), and the three multiplications on $k/2$-bit numbers are performed by the main thread using GMP standard multiplication. The code for splitting the numbers into most and least significant part is omitted

```
mpz_class karaMulThrAs(mpz_class const& x1, mpz_class const& x2)
{
    assert( x1.get_mpz_t()->_mp_size == x1.get_mpz_t()->_mp_size ); // runtime check

    mp_bitcnt_t halfBits= (x1.get_mpz_t()->_mp_size/2) * sizeof(mp_limb_t) * 8;

    auto const part1 = splitBigNum_limb(x1);    // part1 is a std::pair{ high_bits1, low_bits1}
    auto const part2 = splitBigNum_limb(x2);    // part2 is a std::pair{ high_bits2, low_bits2}

        // concurrent multiplication 1
    auto retLL = std::async(std::launch::async, standardMul, part1.second, part2.second);
        // concurrent multiplication 2
    auto retHH = std::async(std::launch::async, standardMul, part1.first, part2.first);

        // multiplication 3 (main thread)
    mpz_class midTerm= - (part1.first-part1.second) * (part2.first-part2.second);

        // retrieves partial result of first multiplication or block till ready
    mpz_class x1Lx2L= retLL.get();
        // ditto, for second multiplication
    mpz_class x1Hx2H= retHH.get();
    midTerm += x1Lx2L + x1Hx2H;

    //computing final result
    mpz_class ret= x1Lx2L + (midTerm << halfBits) + (x1Hx2H << (2*halfBits));

    return ret;
}
```

Fig. 12 Parallel code implementing the Karatsuba step through C++ `std::async` concurrency construct. Each of the three multiplications on $k/2$-bit numbers is performed by a different thread: the first two by additional threads, each operating inside the `std::async`, and the third directly performed in the main thread. The main thread then waits the completion of the other ones (e.g., `retLL.get()`) before calculating the final result

operator. The two operands are split, and the three multiplications are performed in sequence by the same main thread, which eventually calculates the final result from the three partial ones. `splitBigNum_limb` procedure, not shown, splits a multi-precision number into its most and least significant parts using the *limb*, i.e., processor word, granularity.

Then, Fig. 12 shows the version (*Kara_thrAs*) in which two additional threads are spawned through the `std::async` C++ construct to calculate the first two multiplication concurrently and, in parallel with the third one, computed in the main thread. Then the main thread retrieves the partial results calculated by the helper ones or blocks until they are ready (e.g., `retLL.get()` call).

Threads are spawned in the points indicated by the green arrows and are *joined* to the main execution in the points indicated by the red arrows.

Kara_thr version is very similar to the *Kara_thrAs* with only two differences. Firstly, `std::thread` is used in place of `std::async`, and the retrieval of partial results requires to preliminarily join the threads via `thread::join()` method, i.e., possibly waiting for their completion.

The other more advanced versions, *kara_infThr* and *kara_infThrLF*, are not shown as the detailed analysis of their code would require too many advanced concepts of parallel programming for the scope of this chapter. However, their main

features and operational principles are summarized without approximations in the overall discussion of Sect. 2.3.

References

1. G.R. Blakley, A computer algorithm for the product AB modulo M. IEEE Trans. Comput. **32**(5), 497–500 (1983)
2. B. Dally, Efficiency and parallelism: the challenges of future computing. Tech. rep., Nvidia Research, Stanford University, 2014
3. Q.D. Elaine Barker, Recommendation for key management, part 3: application-specific key management guidance. Tech. rep., National Institute of Standards and Technology (NIST), 2015
4. J.L. Hennessy, D.A. Patterson, *Computer Architecture, Fifth Edition: A Quantitative Approach*, 5th edn. (Morgan Kaufmann Publishers Inc., San Francisco, 2011)
5. ISO, *ISO/IEC 14882:2011 – Information technology – Programming languages – C++* (Standard, International Organization for Standardization, Geneva, 2011)
6. J. Kelsey, B. Schneier, D. Wagner, C. Hall, Side channel cryptanalysis of product ciphers, in *Proceedings of the 5th European Symposium on Research in Computer Security, ESORICS '98* (Springer, London, 1998), pp 97–110. http://dl.acm.org/citation.cfm?id=646647.699203
7. D.E. Knuth, *The Art of Computer Programming: Seminumerical Algorithms*, vol. 2, 3rd edn. (Addison-Wesley Longman Publishing Co., Inc., Boston, 1997)
8. Ç.K. Koç, High-speed RSA implementation. Tech. rep., RSA Laboratories, 1994
9. 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, CRYPTO '96* (Springer, London, 1996), pp. 104–113. http://dl.acm.org/citation.cfm?id=646761.706156
10. P. Lara, F. Borges, R. Portugal, N. Nedjah, Parallel modular exponentiation using load balancing without precomputation. J. Comput. Syst. Sci. **78**(2), 575–582 (2012)
11. G. Loukas, *Cyber-Physical Attacks: A Growing Invisible Threat*, 1st edn. (Butterworth-Heinemann, Newton, 2015)
12. M.M. Michael, M.L. Scott, Simple, fast, and practical non-blocking and blocking concurrent queue algorithms, in *Proceedings of the Fifteenth Annual ACM Symposium on Principles of Distributed Computing, PODC '96* (ACM, New York, 1996), pp. 267–275
13. P.L. Montgomery, Modular multiplication without trial division. Math. Comput. **44**(170), 519–521 (1985)
14. D. Page, Theoretical use of cache memory as a cryptanalytic side-channel, 2002
15. R.L. Rivest, A. Shamir, L. Adleman, A method for obtaining digital signatures and public-key cryptosystems. Commun. ACM **21**(2), 120–126 (1978)
16. H. Sutter, The free lunch is over: a fundamental turn toward concurrency in software. Dr Dobb's J. **30**(3), 202–210 (2005)
17. G. Torbjörn, GNU MP - the GNU multiple precision arithmetic library (2016). https://gmplib.org/gmp-man-6.1.2.pdf. Accessed 28 Feb 2018
18. D.M. Tullsen, S.J. Eggers, H.M. Levy, Simultaneous multithreading: maximizing on-chip parallelism, in *Proceedings of the 22nd Annual International Symposium on Computer Architecture, ISCA '95* (ACM, New York, 1995), pp. 392–403
19. M. Wolf, D. Serpanos, Safety and security in cyber-physical systems and internet-of-things systems. Proc. IEEE **106**(1), 9–20 (2018)

Automatic Application of Software Countermeasures Against Physical Attacks

Nicolas Belleville, Karine Heydemann, Damien Couroussé, Thierno Barry, Bruno Robisson, Abderrahmane Seriai, and Henri-Pierre Charles

Abstract While the number of embedded systems is continuously increasing, securing software against physical attacks is costly and error-prone. Several works proposed solutions that automatically insert protections against these attacks in order to reduce this cost and this risk of error. In this chapter, we present a survey of existing approaches and classify them by the level at which they apply the countermeasure. We consider three different levels: the source code level, the compilation level, and the assembly/binary level. We explain the advantages and disadvantages of each level considering different criteria. Finally, we encourage future works to take compilation into account when designing tools, to consider the problem of combining countermeasures, as well as the interactions between countermeasures and compiler optimisations. Going one step further, we encourage future works to imagine how compilation could be modified or redesigned to optimise both performance and security.

1 Introduction

Nowadays, embedded systems have become integral part of our daily life and are of the largest consumer electronics market segment. The number of embedded systems a person manipulates every day is expected to rise massively due to the Internet of

N. Belleville (✉) · D. Couroussé · T. Barry · A. Seriai · H.-P. Charles
Univ Grenoble Alpes, CEA, List, Grenoble, France
e-mail: nicolas.belleville@cea.fr; damien.courousse@cea.fr; thierno.barry@cea.fr; abderrahmane.seriai@cea.fr; henri-pierre.charles@cea.fr

K. Heydemann
Sorbonne University, CNRS, Laboratoire d'Informatique de Paris 6, LIP6, Paris, France
e-mail: karine.heydemann@lip6.fr

B. Robisson
CEA/EMSE, Secure Architectures and Systems Laboratory CMP, Gardanne, France
e-mail: bruno.robisson@cea.fr

Ç. K. Koç (ed.), *Cyber-Physical Systems Security*,
https://doi.org/10.1007/978-3-319-98935-8_7

Things. Back in 2008, this number was already huge as a person used about 230 embedded chips every day [73].

These embedded systems often manipulate sensitive data. For instance, privacy-critical data are handled every day by payment cards, transport cards, smartphones, GPS, etc. Therefore, the security of these systems reveals itself as a major concern for both industrials and state organisations.

Secure devices rely on cryptography to protect sensitive data. While they use cryptographic algorithms that are robust against cryptanalysis, attackers can exploit a physical access to a device either to extract sensitive data such as a cryptographic key, or to bypass authentication, or in certain cases to reverse engineer intellectual properties. These attacks, known as physical attacks, are of two categories. (1) Side channel attacks, introduced in 1996 by Kocher et al. [48], exploit the correlation between the data being processed inside the device and a set of physical quantities that can be measured from outside the device. These physical quantities can be the power consumption of the device [23, 49, 54, 63, 78], the electromagnetic radiation [6, 42], the acoustic emissions [43], the execution time [36, 48], etc. (2) Fault injection attacks, introduced in 1997 by Boneh et al. [21], exploit the effect of a deliberate perturbation of a system during its operation. Fault injection attacks can be carried out by means of laser/light beam [39, 75], electromagnetic injection [35, 62, 65], variation of the supply voltage [12, 24], clock glitch [5], temperature [46, 74], etc.

Several protections to thwart physical attacks have been proposed at software and hardware levels. There are also some mixed hardware-software approaches [9, 18, 31]. In practice, secure elements rely both on hardware and software countermeasures. Moreover, hardware-based solutions are considered as too expensive for IoT devices that face strong cost requirements. The current software hardening process is most often manual and so costly as well as error-prone and tedious. Automating the deployment of software countermeasures is becoming paramount in order to reduce the overall cost and also to offer code hardening solutions for IoT devices.

In this survey, we present how automatic application of software countermeasures has been carried out in the literature by categorising approaches by the level where the countermeasure is applied, either on source code, or on assembly, or within the compilation process. We begin by a brief background (Sect. 2) about side-channel attacks, fault injection attacks, their countermeasures, and the issues related to the compilation of secured code as well as usual ways to circumvent them. Then, we present the approaches that propose an automated application of a countermeasure (Sect. 3) at source code level, compilation level, and assembly level, and we point out their pros and cons. Then we take a step back to compare the different levels (Sect. 4.1). Finally we discuss the important remaining challenges (Sect. 4.2) before concluding (Sect. 5).

2 Background

2.1 Side-Channel Attacks

Instructions and data manipulated by a processor during a program execution affect the processor's power consumption, electromagnetic emissions, and execution time. Side-channel attacks exploit this correlation. Many side-channel attacks proposed in the literature [23, 49, 54, 63, 78] exploit the power consumption of a chip, [36, 48] the execution time of the implementation, and [6, 42] the electromagnetic radiation of a chip.

During an attack, the attacker makes measurements of a physical quantity, while the processor executes the targeted program. She then retrieves the data manipulated by the processor from these measurements, by statistically comparing the measurements with a behavioural model. In this survey, we focus on side-channel attacks that exploit the power consumption or the electromagnetic emissions.

In the case of a correlation power analysis (CPA), the attacker chooses the data she provides as an input to the program, or reads the output of the program (e.g. the encrypted texts). To find the encryption key of an AES, she proceeds byte by byte. Each byte is found as follows: the attacker places an electromagnetic probe on the processor or directly measures its electrical consumption with an oscilloscope. She carries out electrical consumption measurements during several AES executions. For each new run, she gives a random clear text to the program. She calculates theoretical consumptions for each value of the key byte that she is attacking using a consumption model (e. g. the Hamming weight of the value returned by the SBox of the first round). She compares the measurements obtained on several executions with the theoretical consumptions using a statistical operator, here the Pearson correlation. The byte hypothesis that gives the strongest correlation between theoretical and measured consumptions corresponds to the true value of the key byte if enough measurements have been taken.

2.2 Fault Injection Attacks

Processors are designed to work under certain conditions. By using a processor outside these conditions, for example, at a high temperature, faults appear in the calculations [46, 74]. Fault injection attacks exploit this principle. They can use various physical means to provoke faults: light [39, 75], electromagnetic injection [35, 62, 65], temperature [46, 74], etc.

The effects of faults are manifold:

- Bit flips in a register or a memory cell [14, 20, 22, 38, 60, 61]
- Random modification of a value in a register
- Random modification of a value while it is transferred between the CPU and dynamic or non-volatile memory [37, 56]
- Instruction replacement when the instruction fetch gets corrupted [56]

Fault injection attacks can then be used to hijack the execution flow of a program (e.g. to bypass a password verification of a VerifyPIN) or to retrieve information about data manipulated by the program (e.g. finding a cryptographic key). To retrieve a secret data, the attacker analyses the erroneous output that results from these faults, or even the absence of an error on the output, and compares this information using a fault attack model that makes the link between the expected output and the possible outputs in the presence of faults.

2.3 Combined Attacks

Combined attacks are physical attacks that combine side-channel analysis and fault injection.

Currently, all fault attacks are combined with a side-channel observation in practice, in order to monitor the injection of the fault, i.e. to (1) find a suitable moment for the fault injection and (2) precisely control the moment when the fault is injected [77].

Second, some attacks use side-channel analysis and fault injection attack as steps of a wider attack [10]. Several approaches showed that these attacks can break implementations that were protected against both side-channel attacks and fault injection attacks, for example, on an AES implementation [32, 71] or an ECC implementation [41].

2.4 Countermeasures

This section presents the main categories of countermeasures against side-channel attacks and fault injection attacks.

For side-channel attacks, we focus on side channels related to power consumption or electromagnetic emissions and on approaches that were evaluated on these side channels.

Software countermeasures against side-channel attacks can be of two different natures: hiding and masking.

A hiding countermeasure is designed to make attacker's measurements too noisy to be exploitable [26]. For example, one can use dummy rounds or random delays, so that the measurements gathered in two different executions are no more aligned. The link between the measurements and the targeted information is not removed, but the exploitation of the measurements becomes more complicated. There are several types of hiding countermeasures: dummy rounds, random delays, static multiversionning, polymorphism, dual rail, etc. [8, 26–28]. Static multiversionning consists in generating statically several different equivalent execution paths and choosing between them randomly at runtime. Polymorphism consists in dynamically changing the binary code in memory, so that the code is renewed regularly. It was introduced by Amarilli et al. who indicated that it was possible

to automatically implement such countermeasure [8]. Both static multiversionning and runtime polymorphism can use random delay insertion or instruction shuffling, for example, to make the code vary. As complementary approaches, the dual-rail and random precharging countermeasures are sometimes used. Dual rail with precharge logic consists in changing the value encoding so that the Hamming weight of the manipulated values becomes a constant value, and precharging destination registers with the value 0 so that the Hamming distance becomes constant too. Random precharging consists in putting a random value in a register before loading a sensitive value into it in order to prevent transition-based leakages. Note that hiding has been also used outside the scope of power consumption and electromagnetic emission side channels [30, 45, 66].

A masking countermeasure is designed to remove the direct link between the measurements and sensitive data manipulated by the processor [44]. For this purpose, the algorithm of the target program is modified so that all intermediate results that depend on the secret data are separated into several shares, where all the shares are needed to reconstruct the results. For example, first-order Boolean masking consists in performing an "exclusive or" between the secret data and a random number and then carrying out all calculations with this masked data. The masked data and the random number are the two shares here. The random number is changed at each execution, so that the values of the shares change randomly from one execution to another. In practice, hiding and masking countermeasures need to be combined [70]. Indeed, masking needs a certain amount of noise to be effective [47], and hiding can increase the noise.

Software countermeasures against fault attacks can be of three different types: fault tolerance, fault detection, or infective.

A fault tolerance countermeasure aims to ensure that a fault does not alter the output of a program. For example, an instruction duplication countermeasure can be used to tolerate a fault of the type "replacement of an instruction by a nop" [57]. A fault detection countermeasure is intended to detect an attack and then allows to adapt the response to produce (e.g. destroying the system). Control flow integrity countermeasures are fault detection countermeasures that detect a change in control flow [33]. One can also duplicate instructions in order to compare the results to detect a fault. An infective countermeasure aims to make the result of a fault more difficult for an attacker to exploit [67]. The goal is that the attacker does not derive information from the program result when a fault occurred. It can be used as a reaction to a fault detection.

2.5 Compilation of Secured Code

In this section, we give a brief background about compilation and the problem that can arise when compiling secured applications.

Compilation is the process of translating a source code into a binary program for a target architecture [11, 59, 76]. Compilers are usually divided into three parts.

- The front end is in charge of parsing the source code and generating an intermediate representation (IR).
- The middle end is responsible for target-independent optimisations. It is composed of a sequence of analysis and transformation passes that optimise the IR code.
- The back end is responsible for target-dependant optimisations, as well as instruction selection and register allocation, and finally emits the binary program.

Several works have shown that the compiler can alter countermeasures against both side-channel and fault injection attacks when these countermeasures are applied on the source code [13, 15, 72].

Countermeasures can be threatened by various passes. In the case of masking, the passes that simplify arithmetic operations, the instruction scheduling and register allocation passes may alter the countermeasure. For example, the compiler could invert the order of two xors, revealing a secret data. In the case of addition of noise instructions or of instruction redundancy, all the passes that suppress dead code may threaten the countermeasure. In the case of instruction shuffling, the instruction scheduling pass may also alter the countermeasure. Please note that this list is not an exhaustive list of passes that could threaten the countermeasures. Such a list depends on the compiler, its version, the target architecture, etc.

In order to circumvent this problem, one can use various ways:

- One can compile code using the -O0 optimisation flag so that few optimisations remain enabled. Yet, the compilation process remains risky: the code still goes through instruction selection, register allocation, and instruction scheduling, for example, each of these passes being able to alter some countermeasures. In addition, it increases the code surface available for an attack, and there are a lot of register spilling and filling, which increase the information available via side channel.
- One can use the `volatile` keyword in C/C++ source code to force the compiler to not perform memory-access optimisations on some selected variable.
- One can disable some specific passes by using the compiler command line options.
- One can inline assembly code in its source code. However, this solution leads to complex implementations, as developers have to make the link between the C/C++ variables and physical registers. Moreover, the source code is no more portable and becomes harder to maintain.
- One can apply directly the countermeasure on assembly, so that the compilation problem is bypassed. We will see later however that this solution has drawbacks too.

3 Automatic Application of Software Countermeasures

This section presents several state-of-the-art approaches proposed for automatic application of software protections. Table 1 shows an overview of the approaches presented in this section. We present the different approaches, both in this section and in Table 1, gathered according to the code level on which the automatic application is carried out.

The different levels of application for automated approaches will be mainly compared on both their ease of use and the complexity of their implementation. The pros and cons of each level of application will also be presented.

We consider as a usage constraint either the replacement of programming language or the replacement of tools in the developer's usual production chain, such as the compiler. Indeed, changing the programming language can prevent from reusing reference implementations. Also, replacing one of the tools in a production toolchain may not be possible: as an example, closed source software components do not offer the ability to modify the source code or some components may have been certified and any modification would require a new certification process.

While comparing the security level achieved for a specific approach as well as the impact of a protection on performance and code size would be of high interest, it is quite impossible to achieve. Evaluations carried out in the literature vary with the target platform, the considered benchmarks, and the attacks or tests performed. To fairly compare all the approaches, it would then require to dispose of all approaches, to choose a common target of evaluation, and to mount realistic security evaluation scenarios. Hence, we only report fair performance comparison of approaches available in the literature (between approaches [57] and [16]).

3.1 At Source Code Level

Several approaches are proposed to automatically apply countermeasures at source code level.

3.1.1 Side-Channel Attack Countermeasures

Luo et al. proposed an automated hiding countermeasure where independent C operations are shuffled [51]. The associated tool takes C code as input. It gathers statements by group of independent statements, and shuffling is performed at runtime inside each group. It adds dummy statements when too few independent statements have been found for a group in order to increase shuffling effect. It assumes that the code does not contain any loop or branch.

Couroussé et al. proposed an approach to deploy a hiding countermeasure based on runtime polymorphic code generation [29]. Their approach requires to use a

Table 1 Overview of existing automated approaches for side-channel attacks or fault injection attacks gathered by the code level at which they are deployed

Approach	Countermeasure principle		Requirements or constraints
	Side-channel attacks	Fault injection attacks	
Source level			
[51]	Static multiversioning (hiding)	None	Straight-line code
[29]	Polymorphism with runtime code generation (hiding)	None	Domain-specific language
[40]	Masking	None	No input-dependent control flow
[50]	None	Control flow integrity	–
[7]	None	Control flow integrity	–
Compiler level			
[53]	Static multiversioning (hiding)	None	–
[1]	Polymorphism with runtime code modification (hiding)	None	–
[4]	Static multiversionning (hiding) and partial masking	None	–
[3]	Other	None	–
[58]	Masking	None	Domain-specific language
[2]	Masking	None	–
[19]	Random precharging and masking	None	Measurements (optional)
[52]	Threshold implementation (masking)	None	–
[16]	None	Instruction duplication (fault tolerance)	–
[69]	None	Instruction duplication (fault detection) and control flow integrity	–
[64]	None	Instruction duplication on loop exits (fault detection)	–
[25]	None	Instruction and data redundancy (fault detection)	Availability of SIMD instructions

(continued)

Table 1 (continued)

Approach	Countermeasure principle		Requirements or constraints
	Side-channel attacks	Fault injection attacks	
Assembly level			
[17]	Random precharging	None	Measurements to set up the protection
[68]	Dual rail with precharge logic	None	Bitsliced input code
[57]	None	Instruction duplication (fault tolerance)	–
[34]	None	Various fault detection and fault tolerance countermeasures	–

Few approaches consider several different countermeasures, and none of them considers countermeasures for both families of attacks simultaneously

domain-specific language (DSL). The written code is translated by a tool that produces the C code of a specialised polymorphic code generator. The generator regularly produces new versions of the machine code at runtime using semantic variants at machine instruction level, instructions and registers shuffling, and insertion of noise instructions.

Eldib et al. proposed an approach to automatically find and apply a masking countermeasure, with the help of a SMT solver [40]. They assume that the program has an input-independent control flow. The program is parsed and transformed into LLVM's intermediate representation (LLVM IR) by clang. The code in LLVM IR format is then transformed into a Boolean program. Then, each operation of the program is masked, directly if it is a linear operation, by finding a sequence of equivalent masked instructions found out by a SMT solver. Then, the secured code is emitted as C++ code and compiled in -O0 (this information comes from a discussion with authors).

3.1.2 Fault Injection Countermeasures

Lalande et al. proposed to apply a control flow integrity countermeasure based on counters and additional variables at the source code level [50]. The countermeasure is applied in two phases; first, all vulnerabilities of the original code are searched for by simulating control flow hijacking faults at the source code level; then the countermeasure is applied to vulnerable points. Jump attacks larger than two C statements are systematically detected. However, smaller faults, for example that only affect one assembly instruction, are not always detected.

Akkar et al. also presented an automated application of a control flow integrity countermeasure [7]. The developer must annotate his code beforehand using pragmas to indicate the areas to secure. The application is done by a tool that comes in the form of a preprocessor.

3.1.3 Pros and Cons of Source Code Level

The source code level has the advantage of being compatible with the use of proprietary compilers and even of allowing the use of several different toolchains without any compatibility concerns.

In addition, a substantial amount of information is available at this level, such as variable typing information.

This level of application also enables to be independent of the target architecture. Thus, the development of a tool may be easier at this level if a lot of architectures have to be supported.

However, the countermeasures may be altered by compilation. This is not always the case, for example, in the COGITO approach [29], the countermeasure is applied at runtime by a dedicated generator, and therefore there is no risk that it will be altered by the compilation. Approaches [50] and [40] suggest to compile the secure parts without compiler optimisations to circumvent this problem, which does not remove completely the risk as discussed in Sect. 2.5. Thus, developers will have to check for each hardened application at source code level that the countermeasures are still present and correct after compilation. This typically involves reviewing the assembly code produced by the compiler, which is a tedious and error-prone task.

3.2 During Compilation

Several approaches are proposed to apply countermeasures during compilation. Table 2 summarises the level of application inside the compiler and the passes that have been modified for each approach.

3.2.1 Side-Channel Attack Countermeasures

Malagón et al. proposed to deploy a hiding countermeasure based on static generation of several variants of a function [53]. This countermeasure consists in randomly choosing between different versions of the same code at runtime. The source code must be annotated using pragmas by developers to indicate functions where sensitive data are being manipulated. The compiler then generates several different versions of the function code by changing optimisation configuration parameters, for example, using the loop unwinding pass. It also inserts the code that is in charge of randomly selecting at runtime the version of the code to be executed.

Agosta et al. proposed another hiding countermeasure based on dynamic modification of code [1]. The code is modified at runtime using semantic equivalence at instruction level, randomisation of table accesses, and mixed instructions. The countermeasure is automatically applied by a compiler: some transformation passes have been added in LLVM in order to statically prepare the transformations made at runtime.

Table 2 Level of application and modified passes within the compiler for compiler-level approaches

Approach	Level of application	Modified passes
Malagón et al. [53]	Middle end	Loop unwinding pass
Agosta et al. [1]	Unknown	–
Agosta et al. [4]	Middle end and back end	Several (unknown) passes in middle end and back end
Agosta et al. [3]	Middle end and back end	Instruction selection
Moss et al. [58]	Middle end	–
Agosta et al. [2]	Middle end	–
Bayrak et al. [19]	Middle end and back end	–
Eldib et al. [40]	Middle end	–
Barry et al. [16]	Back end	Instruction selection and register allocation
Reis et al. [69]	Unknown	–
Proy et al. [64]	Middle end and back end	Branch folding and register allocation
Chen et al. [25]	Middle end	–
Luo et al. [52]	Middle end	–

Agosta et al. also proposed a hiding countermeasure based on static generation of several variants [4]. The authors propose to generate automatically a code containing multiple execution paths, with choice between the different paths at runtime, which is also a hiding countermeasure. This approach also incorporates some masking elements, since the SBox accesses are masked. In addition, the process of saving registers on the stack is modified: one register is dedicated to hold a random value used to mask any register value stored in the stack. When the content of the register is restored, it is also unmasked so that it can be used again. All these transformations are handled by new transformation passes in LLVM. Some existing passes have also been modified. Among other things, modifications to existing passes are intended to ensure that an instruction that was in an area to be protected cannot leave this area because of optimisations. The developer must provide a C file annotated so as to specify the code regions to protect and the SBox. In addition to the source file, the compiler takes an input file that specifies the equivalent instructions to be used.

Agosta et al. also proposed a new countermeasure against side-channel attacks that aims to bring out several key hypotheses instead of one during an attack so that the attacker cannot know which one is the right hypothesis [3]. This countermeasure is entirely applied during compilation, in several steps. Several passes have been added in the middle end and back end; also the instruction selection pass has been changed. The compiler takes an input file annotated by the developer that specifies the parts of the code to be protected.

Moss et al. proposed to automatically apply a Boolean masking countermeasure during compilation [58]. The developer must write his program in a domain-specific language (DSL). This DSL allows to express with predefined types the level of confidentiality of variables, for example, to indicate that a variable is secret. The

compiler then uses this information to determine which intermediate values are to be masked and thus masks these values.

Agosta et al. also proposed an approach for the application of a masking countermeasure. Their approach allows to generate higher-order masked code [2]. The compiler calculates for each key-dependent value the number of key bits on which the value depends. This analysis enables to apply the countermeasure only to intermediate values that depend on a small number of key bits. For example, intermediate values dependent on all bits of the key are not masked. This principle reduces the overhead of the countermeasure.

Bayrak et al. also proposed a compilation approach to apply Boolean masking to a program [19]. An important difference with the other approaches is that they use the compiler to decompile a binary program to a higher-level representation and then recompile the program while applying the protection. To find out where to apply the countermeasure, they suggest to start by identifying instructions that may reveal sensitive data through a side channel. This analysis is either done using measurements provided by the user or statically. The countermeasure is then applied to all instructions that were found to be critical compared to a predefined threshold. This enables to partially apply the countermeasure and to reduce the performance overheads. In addition, the compiler can also apply a random precharging countermeasure.

Luo et al. proposed a similar approach to generate a threshold implementation automatically on LLVM IR [52]. Threshold implementation is a countermeasure close to the masking countermeasure, as the secret is split into shares. Yet, in threshold implementation, every function is independent from at least one of the shares, which is not the case for masking. They use a SAT solver along with a transformation step in order to find suitable solution. Every function is split into a succession of smaller functions so that the SAT solver can find solutions effectively.

3.2.2 Fault Injection Countermeasures

Barry et al. used the compiler to automatically apply a fault tolerance countermeasure [16]. They duplicate assembly instructions to tolerate the skip of one instruction. The use of the compiler is twofold compared to a lower-level approach: it favours the selection of instructions compliant with the duplication scheme, increasing the number of idempotent instructions, and takes advantage of optimisations to gain performance. To this end, several passes have been added to LLVM, and the instruction selection and register allocation passes have been modified. The overheads obtained are lower than those obtained by applying this countermeasure at the assembly level.

Reis et al. proposed to deploy a fault detection countermeasure during compilation [69]. Instructions are duplicated so that their results are compared in order to detect faults. In addition, additional checks are added to ensure that the control flow is not hijacked. The authors indicate that the approach could be easily extended to include fault tolerance.

Proy et al. proposed to use the compiler to apply a countermeasure to secure loops against fault injection attacks [64]. The instructions involved in the computation of conditions for exiting the loop are duplicated to add checking blocks in charge of detecting an early exit or an extra iteration. This transformation is applied at IR level. They explain that some compiler passes had to be modified to keep the countermeasure correctly applied until the code is emitted.

Finally, Chen et al. proposed to achieve operation redundancy by using SIMD instructions [25]. Their compiler vectorises some instructions in order to have instruction redundancy and adds error-checking codes. All the code transformations are performed at the IR level, and the approach is architecture-independent. It only requires the target architecture to have support for SIMD instructions. The use of SIMD instructions allows to obtain a smaller performance overhead compared to classic instruction duplication approaches.

3.2.3 Pros and Cons of Compiler Level

The compiler level is interesting if several source languages need to be supported, as the front end usually supports various languages.

Moreover, the back end must most often be modified and therefore the approach depends on the architecture. However, some elements applied in the middle end are common for all architectures, so adding support for an architecture is done without starting from scratch.

What is more, the application of countermeasures during the compilation process makes it possible to finely control the transformations carried out in the compiler and to choose when to apply the countermeasure to avoid the risk that it will be altered by the compilation. The compiler allows to have both high-level information such as the types of variables and low-level information that depends on the target architecture. Thus, countermeasure can be applied in several transformation passes, strategically placed in the compilation process. As an example, Reis et al. [69] and Barry et al. [16] exploit the scheduling instruction pass to reduce the countermeasure overhead by creating parallelism at the instruction level (depending on the latency of the instructions). In addition, several approaches modify compiler transformation passes such as instruction selection or register allocation to prepare the countermeasure application in order to produce a more efficient code.

Moreover, if developers manage to propagate the necessary information throughout the compilation process, developers can add a check pass before issuing instructions to confirm that the countermeasure has been correctly applied and that it has not been altered by possible downstream optimisations.

The engineering effort deployed to implement such approaches is important; nevertheless, the control offered by this level of application makes it possible to obtain an important confidence in the produced code. In case a checking pass is added before code emission, it is not necessary to manually check the presence and the effectiveness of the countermeasures in the produced assembly code for each

hardened application. In that case, the developer does not need to be an expert in security to be able to effectively secure his applications.

This level of application requires to have access to the compiler source code. If the developer uses a closed-source compiler, using a compiler approach would imply to use an open-source one to disassemble a file, reconstruct an intermediate representation, apply the countermeasure to the code, and recompile it, which is a tough process.

3.3 At Link Time/At Assembly Level

This section presents approaches that apply countermeasures directly on an assembly file, during or before the linking phase.

3.3.1 Side-Channel Attack Countermeasures

Bayrak et al. proposed to automatically apply a random precharging countermeasure at assembly level [17]. The application of this countermeasure is quite natural at this level, as register allocation has already been performed. Empirical measurements made on unsecure code are used to determine the instructions to be secured.

Rauzy et al. also implemented a side-channel countermeasure at assembly level: dual rail with precharge logic [68]. Their approach requires that the code has previously been bitsliced. Their approach also makes it possible to prove that the transformation is correct and that the program obtained after transformation remains semantically correct.

3.3.2 Fault Attack Countermeasures

Moro et al. proposed a countermeasure based on instruction duplication to achieve fault tolerance [55]. This countermeasure is intended to tolerate the jump of one instruction. For this purpose, each instruction is replaced by a sequence of instructions, this sequence being semantically equivalent to the original instruction and being tolerant to one instruction skip. As this countermeasure requires additional registers, it is sometimes necessary to spill some registers. In addition, some instructions (e.g. volatile loads) cannot be replaced by a fault-tolerant sequence. This is the same countermeasure as the one automated by Barry et al. [16] afterwards at compilation level.

De Keulenaer et al. showed how to automatically deploy various countermeasures against fault attacks at binary level using link-time rewriting [34]. Their tool combines both fault tolerance countermeasures and fault detection countermeasures: duplication of conditional jumps, call graph integrity, verification of memory entries, and duplication of loop counters.

3.3.3 Pros and Cons of Assembly Level

This level is mostly used to apply countermeasures that are quite low level, as applying higher-level countermeasures at this level is complicated since it is then necessary to reconstruct a certain amount of information that has been lost. For example, variable typing information is no longer present. In addition, the application of countermeasures often requires the use of additional registers, which requires either register spilling or a complete reallocation of registers.

Thus, during the development of an automatic approach at this level, a major engineering effort is necessary to obtain information that was available at compilation or to redo treatments that had been done by the compiler in a way that was not optimal with respect to the countermeasure to be applied.

However, applying countermeasures at this level avoids having to check manually if the countermeasure is still present in the final code, since the compilation process takes place entirely before the countermeasure application. This allows the use of such a tool by a non-security expert developer. Moreover, this level of application allows to be independent of source code language, which is interesting if several source languages need to be supported. In addition, it allows to secure code after link-time optimisation and to potentially secure binary libraries.

4 Discussion

4.1 Confrontation of Pros and Cons of the Different Levels

This section discusses the advantages and disadvantages of the aforementioned levels of automatic application of countermeasures.

The first aspect to consider is the time taken for developing an automated tool. This aspect depends on the countermeasure that has to be applied. A masking countermeasure is easier to apply at source code level than at assembly level because it requires a modification of the algorithm. The compiler is a place where various countermeasures can be applied, as during compilation the compiler manipulates both quite high-level representations (e.g. with typed variables) and low-level representations (e.g. with assembly instructions).

Developing an automated tool implies parsing and emitting code in the targetted formats. Compilers already have the necessary code for that, and usually the developer only has to add a pragma support to delimit the code zones to be secured. For source code and assembly-level approaches, the developers often have to implement or reuse a parser and/or an emitter for the targetted codes.

The engineering cost taken at using the tools must be considered too. As these tools are automatic, the cost of producing secured code is close to zero, yet the development of the tools requires a lot of work. When the tool applies the countermeasure at source code level, code review is facilitated, but the user has

to check that the countermeasure is still valid at the assembly level. This time-consuming task is one of the main drawbacks of the source code-level approach. The assembly approach does not suffer from this drawback: applying a countermeasure at the assembly level prevents from alteration during compilation. Applying a countermeasure during compilation allows to check that the countermeasure is still valid just before assembly/binary code emission if the developer manages to propagate the necessary information throughout the compiler. If checking the countermeasure before code emission is not possible, a step of assembly code review is still needed.

Considering performance in terms of code size and of execution time, the compiler level allows fine-tuning. When a countermeasure is applied within the compiler, it can benefit from optimisations, whereas if it is applied outside the compilation process, it requires to redevelop some optimisations afterwards. Several approaches that use compilers modify some passes of the compiler to reduce the cost of the countermeasures. The passes that apply the countermeasure can be carefully interleaved with compiler passes to take advantage of these passes without risking the countermeasure to get altered by optimisations [16]. At other levels, tuning transformations for performance may be harder. For example, at assembly level, the need for additional registers either requires to do register spilling or to perform again the register allocation. As a comparison, Moro et al. and Barry et al. implemented the same countermeasure at assembly level and compiler level, respectively. Barry et al. obtained execution time overheads and size overheads lower than Moro et al.

4.2 Future Works

All of these approaches target either side-channel attacks or fault injection attacks, and few of them consider the application of several different countermeasures. Yet, programs have to be secured against both families of attacks and within each family of attack and have to be secured against a large number of variants. Thus, countermeasures have to be combined so that the programs meet the security requirements.

The problem of automatic application of combined countermeasure has not been investigated yet to the best of our knowledge. It raises important questions in order to be able to guarantee that every countermeasure is correctly applied on the produced code.

Similarly to the conflicts that can appear between countermeasures and some optimisation passes of a compiler, conflicts can appear between different countermeasures. The order of application of the countermeasures should be well thought: which countermeasure must be applied first? Must the countermeasures be applied in a combined way? Several compiler approaches are proposed to apply a countermeasure in several steps, interleaved with compiler passes. How should one interleave all the different steps to apply two very different countermeasures? This issue is present whatever the level at which countermeasures are applied and refrains

the simple approach that would consist in simply combining several different tools one after the other as they would not be aware of the countermeasures that are applied by the others.

In addition, when the compiler level is chosen to apply the countermeasures, strategies for the modification of compiler passes have to be made with all countermeasures in mind. For example, register allocation should be compliant with several countermeasures that may have different objectives: one may want to constrain register spilling to prevent distance-based leakage in the presence of a masking countermeasure while needing new registers to implement a fault detection countermeasure.

We encourage future works to consider the problem of compilation for security, to study the interaction between the different countermeasures and performance optimisations, and to rethink the compilation process so that it can optimise at the same time the performance and security goals.

5 Conclusion

The automatic application of countermeasures against physical attacks is a crucial research problem as a lot of platforms are concerned by these threats while securing them manually is costly. We presented the different approaches to automatically deploy software countermeasures against these attacks. Some of them directly modify the source code, others modify the assembly code, and others propose to modify the compiler so that the countermeasure is applied during the compilation process. While developing solutions at the compilation level is not always possible, we encourage this practice as it allows to tune performance while providing confidence that the countermeasure remains correctly applied in the assembly file. We also encourage future research to consider the problem of automatic application of combined countermeasures that has not yet been addressed, their interaction with compiler optimisations, and to try to create compilers that optimise both security and performance. These are interesting and challenging issues to solve to be able to offer security automatically.

Acknowledgements This work was partially funded by the French National Research Agency (ANR) as part of the projects COGITO and PROSECCO, respectively funded by the programs INS-2013 under grant agreement ANR-13-INSE-0006-01 and AAP-2015 under grant agreement ANR-15-CE39.

References

1. G. Agosta, A. Barenghi, G. Pelosi, A code morphing methodology to automate power analysis countermeasures, in *Proceedings of DAC* (2012), pp. 77–82
2. G. Agosta, A. Barenghi, M. Maggi, G. Pelosi, Compiler-based side channel vulnerability analysis and optimized countermeasures application, in *2013 50th ACM/EDAC/IEEE Design Automation Conference (DAC)* (IEEE, Piscataway, 2013), pp. 1–6
3. G. Agosta, A. Barenghi, G. Pelosi, M. Scandale, *Information Leakage Chaff: Feeding Red Herrings to Side Channel Attackers* (ACM Press, New York, 2015), pp. 1–6
4. G. Agosta, A. Barenghi, G. Pelosi, M. Scandale, The MEET approach: securing cryptographic embedded software against side channel attacks. IEEE TCAD **34**(8), 1320–1333 (2015)
5. M. Agoyan, J.-M. Dutertre, D. Naccache, B. Robisson, A. Tria, When clocks fail: on critical paths and clock faults. Lect. Notes Comput. Sci. **6035**, 182–193 (2010)
6. D. Agrawal, B. Archambeault, J. Rao, P. Rohatgi, The em Side-Channel(s). Lect. Notes Comput. Sci. **2523**, 29–45 (2003)
7. M.-L. Akkar, L. Goubin, O. Ly, Automatic integration of counter-measures against fault injection attacks (2003). Pre-print found at http://www.labri.fr/Perso/ly/index.htm
8. A. Amarilli, S. Müller, D. Naccache, D. Page, P. Rauzy, M. Tunstall, Can code polymorphism limit information leakage? in *IFIP International Workshop on Information Security Theory and Practices* (Springer, 2011), pp. 1–21
9. J. Ambrose, R. Ragel, S. Parameswaran, RIJID: Random code injection to mask power analysis based side channel attacks, in *44th ACM/IEEE Design Automation Conference, DAC '07*, June 2007, pp. 489–492
10. F. Amiel, K. Villegas, B. Feix, L. Marcel, Passive and active combined attacks: combining fault attacks and side channel analysis, in *Workshop on Fault Diagnosis and Tolerance in Cryptography, 2007. FDTC 2007* (IEEE, 2007), pp. 92–99
11. A.W. Appel, M. Ginsburg, *Modern Compiler Implementation in C* (Cambridge University Press, New York, 2004)
12. C. Aumüller, P. Bier, W. Fischer, P. Hofreiter, J.-P. Seifert, Fault attacks on RSA with CRT: concrete results and practical countermeasures. Lect. Notes Comput. Sci. **2523**, 260–275 (2003)
13. J. Balasch, B. Gierlichs, V. Grosso, O. Reparaz, F.-X. Standaert, On the cost of lazy engineering for masked software implementations. Lect. Notes Comput. Sci. **8968**, 64–81 (2015)
14. H. Bar-El, H. Choukri, D. Naccache, M. Tunstall, C. Whelan, The sorcerer's apprentice guide to fault attacks. Proc. IEEE **94**(2), 370–382 (2006)
15. M. Barbosa, A. Moss, D. Page, Constructive and destructive use of compilers in elliptic curve cryptography. J. Cryptol. **22**(2), 259–281 (2009)
16. T. Barry, D. Couroussé, B. Robisson, Compilation of a countermeasure against instruction-skip fault attacks, in *Proceedings of the Third Workshop on Cryptography and Security in Computing Systems* (ACM, New York, 2016), pp. 1–6
17. A.G. Bayrak, F. Regazzoni, P. Brisk, F.-X. Standaert, P. Ienne, A first step towards automatic application of power analysis countermeasures, in *Proceedings of the 48th Design Automation Conference* (ACM, 2011), pp. 230–235
18. A.G. Bayrak, N. Velickovic, P. Ienne, W. Burleson, An architecture-independent instruction shuffler to protect against side-channel attacks. ACM Trans. Archit. Code Optim. **8**(4), 20:1–20:19 (2012)
19. A.G. Bayrak, F. Regazzoni, D. Novo, P. Brisk, F.-X. Standaert, P. Ienne, Automatic application of power analysis countermeasures. IEEE Trans. Comput. **64**(2), 329–341 (2015)
20. I. Biehl, B. Meyer, V. Müller, Differential fault attacks on ellitic curve cryptosystems, in *Advances in Cryptology (CRYPTO 2000)*, ed. by M. Bellare. Lecture Notes in Computer Science, vol. 1880 (Springer, Berlin, 2000)

21. D. Boneh, R.A. DeMillo, R.J. Lipton, On the importance of checking cryptographic protocols for faults, in *International Conference on the Theory and Applications of Cryptographic Techniques* (Springer, Berlin, 1997), pp. 37–51
22. D. Boneh, R.A. DeMillo, R.J. Lipton, On the importance of eliminating errors in cryptographic computations. J. Cryptol. **14**, 101–119 (2001)
23. E. Brier, C. Clavier, F. Olivier, Correlation power analysis with a leakage model. Lect. Notes Comput. Sci. **3156**, 16–29 (2004)
24. R.B. Carpi, S. Picek, L. Batina, F. Menarini, D. Jakobovic, M. Golub, Glitch it if you can: parametersearch strategies for successful fault injection, in *Smart Card Research and Advanced Applications*. Lecture Notes in Computer Science (Springer, Cham, 2013)
25. Z. Chen, J. Shen, A. Nicolau, A. Veidenbaum, N. Farhady. CAMFAS: a compiler approach to mitigate fault attacks via enhanced SIMDization, in *2017 Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC)* (IEEE, Piscataway, 2017), pp. 57–64
26. C. Clavier, J.-S. Coron, N. Dabbous, Differential power analysis in the presence of hardware countermeasures, in *Cryptographic Hardware and Embedded Systems - CHES 2000*. Lecture Notes in Computer Science (Springer, Berlin, 2000), pp. 252–263
27. J.-S. Coron, I. Kizhvatov, An efficient method for random delay generation in embedded software, in *International Workshop on Cryptographic Hardware and Embedded Systems*. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), vol. 5747 (2009), pp. 156–170
28. J.-S. Coron, I. Kizhvatov, Analysis and improvement of the random delay countermeasure of CHES 2009, in *International Workshop on Cryptographic Hardware and Embedded Systems*. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), vol. 6225 (2010), pp. 95–109
29. D. Couroussé, T. Barry, B. Robisson, P. Jaillon, O. Potin, J.-L. Lanet, *Runtime Code Polymorphism as a Protection Against Side Channel Attacks*, vol. 9895, Sept. 2016, pp. 136–152
30. S. Crane, A. Homescu, S. Brunthaler, P. Larsen, M. Franz, Thwarting cache side-channel attacks through dynamic software diversity. Internet Society, 2015
31. J.-L. Danger, S. Guilley, T. Porteboeuf, F. Praden, M. Timbert, *HCODE: Hardware-Enhanced Real-Time CFI* (ACM Press, New York, 2014), pp. 1–11
32. F. Dassance, A. Venelli, Combined fault and side-channel attacks on the AES key schedule (2012), pp. 63–71
33. R. de Clercq, I. Verbauwhede, A survey of Hardware-based Control Flow Integrity (CFI) (2017). arXiv:1706.07257
34. R. De Keulenaer, J. Maebe, K. De Bosschere, B. De Sutter, Link-time smart card code hardening. Int. J. Inf. Secur. **15**(2), 111–130 (2016)
35. A. Dehbaoui, J.-M. Dutertre, B. Robisson, P. Orsatelli, P. Maurine, A. Tria, Injection of transient faults using electromagnetic pulses -Practical results on a cryptographic system-. IACR Cryptology EPrint Archive **2012**, 123 (2012)
36. J.-F. Dhem, F. Koeune, P.-A. Leroux, P. Mestré, J.-J. Quisquater, J.-L. Willems, A practical implementation of the timing attack. Lect. Notes Comput. Sci. **1820**, 167–182 (2000)
37. L. Dureuil, M. Potet, P. de Choudens, C. Dumas, J. Clédière, From code review to fault injection attacks: filling the gap using fault model inference, in *Smart Card Research and Advanced Applications - 14th International Conference, CARDIS 2015, Bochum, Germany, November 4–6, 2015. Revised Selected Papers* (2015), pp. 107–124
38. P. Dusart, G. Letourneux, O. Vivolo, Differential fault analysis on AES, in *Applied Cryptography and Network Security (ANCS 2003)*, ed. by M. Yung, Y. Han, J. Zhou. Lecture Notes in Computer Science, vol. 2846 (Springer, Berlin, 2003), pp. 293–306
39. J.-M. Dutertre, S. De Castro, A. Sarafianos, N. Boher, B. Rouzeyre, M. Lisart, J. Damiens, P. Candeier, M.-L. Flottes, G. Di Natale, Laser attacks on integrated circuits: from CMOS to FD-SOI, in *2014 9th IEEE International Conference on Design & Technology of Integrated Systems in Nanoscale Era (DTIS)* (IEEE, 2014), pp. 1–6

40. H. Eldib, C. Wang, Synthesis of masking countermeasures against side channel attacks, in *International Conference on Computer Aided Verification* (Springer, Berlin, 2014), pp. 114–130
41. J. Fan, B. Gierlichs, F. Vercauteren, To infinity and beyond: combined attack on ECC using points of low order. Lect. Notes Comput. Sci. **6917**, 143–159 (2011)
42. K. Gandolfi, C. Mourtel, F. Olivier, Electromagnetic analysis: concrete results. Lect. Notes Comput. Sci. **2162**, 251–261 (2001)
43. D. Genkin, A. Shamir, E. Tromer, Acoustic cryptanalysis. J. Cryptol. **30**(2), 392–443 (2017)
44. L. Goubin, J. Patarin, DES and differential power analysis (The "duplication" method), in *Proceedings of the First International Workshop on Cryptographic Hardware and Embedded Systems, CHES '99* (Springer, London, 1999), pp. 158–172
45. A. Homescu, S. Brunthaler, P. Larsen, M. Franz, *Librando: Transparent Code Randomization for Just-in-Time Compilers* (ACM Press, New York, 2013), pp. 993–1004
46. M. Hutter, J.-M. Schmidt, The temperature side channel and heating fault attacks. Lect. Notes Comput. Sci. **8419 LNCS**, 219–235 (2014)
47. A. Journault, F.-X. Standaert, Very high order masking: efficient implementation and security evaluation, in *Cryptographic Hardware and Embedded Systems - CHES 2017*. Lecture Notes in Computer Science (Springer, Cham, 2017), pp. 623–643
48. P. Kocher, Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems, in *Advances in Cryptology - CRYPTO'96* (Springer, Berlin, 1996), pp. 104–113
49. P. Kocher, J. Jaffe, B. Jun, Differential power analysis. Lect. Notes Comput. Sci. **1666**, 388–397 (1999)
50. J.-F. Lalande, K. Heydemann, P. Berthomé, Software countermeasures for control flow integrity of smart card C codes, in *European Symposium on Research in Computer Security* (Springer, Berlin, 2014), pp. 200–218
51. P. Luo, L. Zhang, Y. Fei, A.A. Ding, Towards secure cryptographic software implementation against side-channel power analysis attacks, in *2015 IEEE 26th International Conference on Application-Specific Systems, Architectures and Processors (ASAP)* (IEEE, Piscataway, 2015), pp. 144–148
52. P. Luo, K. Athanasiou, L. Zhang, Z.H. Jiang, Y. Fei, A.A. Ding, T. Wahl, *Compiler-Assisted Threshold Implementation Against Power Analysis Attacks* (IEEE, Piscataway, 2017), pp. 541–544
53. P. Malagón, J.M. de Goyeneche, M. Zapater, J. Moya, Z. Banković, Compiler optimizations as a countermeasure against side-channel analysis in MSP430-based devices. Sensors (Switzerland) **12**(6), 7994–8012 (2012)
54. S. Mangard, E. Oswald, T. Popp, Power Analysis attacks: revealing the secrets of smart cards (2007). https://doi.org/10.1007/978-0-387-38162-6
55. N. Moro, Security of assembly programs against fault attacks on embedded processors, Theses, Université Pierre et Marie Curie - Paris VI, Nov. 2014
56. N. Moro, A. Dehbaoui, K. Heydemann, B. Robisson, E. Encrenaz, Electromagnetic fault injection: towards a fault model on a 32-bit microcontroller, in *2013 Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC)* (IEEE, Piscataway, 2013), pp. 77–88
57. N. Moro, K. Heydemann, E. Encrenaz, B. Robisson, Formal verification of a software countermeasure against instruction skip attacks. J. Cryptogr. Eng. **4**(3), 145–156 (2014)
58. A. Moss, E. Oswald, D. Page, M. Tunstall, Compiler assisted masking. Lect. Notes Comput. Sci. **7428**, 58–75 (2012)
59. S.S. Muchnick, *Advanced Compiler Design and Implementation* (Morgan Kaufmann Publishers Inc., San Francisco, 1997)
60. S. Ordas, L. Guillaume-Sage, K. Tobich, J.-M. Dutertre, P. Maurine, Evidence of a larger EM-induced fault model, in *International Conference on Smart Card Research and Advanced Applications* (Springer, Berlin, 2014), pp. 245–259
61. S. Ordas, L. Guillaume-Sage, P. Maurine, EM injection: fault model and locality, in *2015 Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC)* (2015), pp. 3–13

62. S. Ordas, L. Guillaume-Sage, K. Tobich, J.-M. Dutertre, P. Maurine, Evidence of a larger EM-induced fault model. Lect. Notes Comput. Sci. **8968**, 245–259 (2015)
63. E. Peeters, *Advanced DPA Theory and Practice: Towards the Security Limits of Secure Embedded Circuits* (2013). https://doi.org/10.1007/978-1-4614-6783-0
64. J. Proy, K. Heydemann, A. Berzati, A. Cohen, Compiler-assisted loop hardening against fault attacks. ACM Trans. Archit. Code Optim. **14**(4), 36:1–36:25 (2017)
65. J.-J. Quisquater, D. Samyde, ElectroMagnetic analysis (EMA): measures and counter-measures for smart cards, in *Smart Card Programming and Security*. Lecture Notes in Computer Science (Springer, Berlin, 2001), pp. 200–210. https://doi.org/10.1007/3-540-45418-7_17
66. A. Rane, C. Lin, M. Tiwari, Raccoon: closing digital side-channels through obfuscated execution, in *Proceedings of the 24th USENIX Conference on Security Symposium, SEC'15* (USENIX Association, Berkeley, 2015), pp. 431–446
67. P. Rauzy, S. Guilley, Countermeasures against high-order fault-injection attacks on CRT-RSA, in *2014 Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC)* (IEEE, 2014), pp. 68–82
68. P. Rauzy, S. Guilley, Z. Najm, Formally proved security of assembly code against power analysis: a case study on balanced logic. J. Cryptogr. Eng. **6**(3), 201–216 (2016)
69. G.A. Reis, J. Chang, N. Vachharajani, R. Rangan, D.I. August, SWIFT: software implemented fault tolerance, in *Proceedings of the international symposium on Code generation and optimization* (IEEE Computer Society, Piscataway, 2005), pp. 243–254
70. M. Rivain, E. Prouff, J. Doget, Higher-order masking and shuffling for software implementations of block ciphers, in *Cryptographic Hardware and Embedded Systems - CHES 2009*. Lecture Notes in Computer Science (Springer, Berlin, 2009), pp. 171–188. https://doi.org/10.1007/978-3-642-04138-9_13
71. T. Roche, V. Lomné, K. Khalfallah, Combined fault and side-channel attack on protected implementations of AES. Lect. Notes Comput. Sci. **7079**, 65–83 (2011)
72. H. Seuschek, F. De Santis, O.M. Guillen, *Side-Channel Leakage Aware Instruction Scheduling* (ACM Press, New York, 2017), pp. 7–12
73. J. Sifakis, A vision for computer science - the system perspective. Cent. Eur. J. Comput. Sci. **1**(1), 108–116 (2011)
74. S. Skorobogatov, Local heating attacks on flash memory devices, in *IEEE International Workshop on Hardware-Oriented Security and Trust (HOST'09)* (IEEE Computer Society, 2009), pp. 1–6
75. S. Skorobogatov, R. Anderson, Optical fault induction attacks. Lect. Notes Comput. Sci. **2523**, 2–12 (2003)
76. Y. Srikant, P. Shankar, *The Compiler Design Handbook: Optimizations and Machine Code Generation*, 2nd edn. (CRC Press, Boca Raton, 2007)
77. N. Timmers, A. Spruyt, M. Witteman, Controlling PC on ARM using fault injection, in *2016 Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC)* (IEEE, 2016), pp. 25–35
78. J. VanLaven, M. Brehob, K. Compton, A computationally feasible SPA attack on AES via optimized search. IFIP Adv. Inf. Commun. Technol. **181**, 577–588 (2005)

Time-Delay Attacks in Network Systems

Gianluca Bianchin and Fabio Pasqualetti

Abstract Modern cyber-physical systems rely on dependable communication channels to accomplish cooperative tasks, such as forming and maintaining a coordinated platooning configuration in groups of interconnected vehicles. We define and study a class of adversary attacks that tamper with the temporal characteristics of the communication channels, thus leading to delays in the signals received by certain network nodes. We show how such attacks may affect the stability of the overall interconnection, even when the number of compromised channels is limited. Our algorithms allow us to identify the links that are inherently less robust to this class of attacks and to study the resilience of different network topologies when the attacker goal is to minimize the number of compromised communication channels. Based on our numerical results, we reveal a relation between the robustness of a certain network topology and the degree distribution of its nodes.

1 Introduction

Networks of cyber and physical agents are broadly employed across diverse engineering applications to model critical infrastructures such as transportation systems and power grids [1, 2]. The increased coupling between physical components and cyber layers oftentimes comes at the expense of vulnerabilities and security weaknesses. Several real-world incidents and recent research papers have highlighted the vulnerabilities of these infrastructures on both their physical and cyber layers [3–6]. The available literature on cyber-physical system security has mainly focused on two categories of attacks: *deception* and *denial of service*. Deception attacks compromise the integrity of the data exchanged across the network and are cast by altering the behavior of sensors, actuators, and communication channels. On the

G. Bianchin (✉) · F. Pasqualetti
University of California, Riverside, Riverside, CA, USA
e-mail: gianluca@engr.ucr.edu; fabiopas@engr.ucr.edu

Ç. K. Koç (ed.), *Cyber-Physical Systems Security*,
https://doi.org/10.1007/978-3-319-98935-8_8

other hand, denial-of-service attacks compromise the availability of resources by, for instance, jamming the communication channels.

Yet, an aspect that critically affects the operation of several classes of cyber-physical systems is the indirect effect of non-ideal communication channels that can introduce timing aberrations in the signals exchanged among their nodes. Timing aberrations can be the indirect result of hardware faults or can be the effect of intentional attacks. For instance, an adversary may temporarily jam communication channels with the goal of delaying the transmitted signal streams while maintaining unaltered the information enclosed in the packets. Although this action does not prevent information from being delivered correctly, it can disrupt the system operation and performance by impeding the correct synchronization among different system components. In this work, we focus on attacks that target the agents communication by delaying the stream of exchanged signals. We consider attacks that are sparse in the set of attacked channels and employ a security metric that captures the stability of the underlying system.

The importance of timing and the effect of time delays in networks of dynamical systems is a well-studied concept (e.g., see [7–10]). Classical methods to study stability of delayed linear systems can be divided into LMI conditions, which arise from a Lyapunov-Krasovskii quadratic function analysis [11, 12], and techniques based on matrix pencils [13, 14]. However, timing-based security is an inherently different issue from standard communication delay approaches, as an attacker can deliberately select the targeted channels and the specific pattern of time delays. The relation between timing and security in cyber-physical systems has been highlighted in some recent work. In particular, while [15] devises a robust output-feedback controller which is resilient to an attack that changes the order at which packets are delivered, the authors in [16] follow a probabilistic approach and model packet drops through Bernoulli processes representing intentional attacker intrusions. The effect of malicious packet drops has also motivated the study and development of resilient controllers in the context of networked control systems [17].

Differently from this line of previous work, securing cyber-physical systems from timing attacks requires the study and design of a specific, well-designed set of delayed channels. This work distinguishes from the above line of research by (i) considering opportunely-defined attacks that do not follow any specific probabilistic model, and by (ii) relating network resilience to topological properties and centrality measures such as the degree distribution. We characterize and study the class of delay attacks from a control perspective and provide a numerical algorithm to identify the set of communication channels that are less robust to timing attacks. Our results suggest that improved robustness can be achieved by network topologies where nodes exhibit significantly large degrees.

2 Problem Setup

This section describes the models we adopt for the analysis of time delays in dynamical systems. The description first introduces the ideal modeling framework in the absence of external attacks and then illustrates the attack scenario.

2.1 Network Model

Consider a network modeled by a directed graph $\mathscr{G} = (\mathscr{V}, \mathscr{E})$, where $\mathscr{V} = \{1, \ldots, n\}$ and $\mathscr{E} \subseteq \mathscr{V} \times \mathscr{V}$ are the vertices and edges sets, respectively. Let $a_{ij} \in \mathbb{R}$ denote the weight associated with the edge $(i, j) \in \mathscr{E}$, and let $a_{ij} = 0$ whenever $(i, j) \notin \mathscr{E}$. We associate a real value x_i (node state) with each node $i \in \{1, \ldots, n\}$ of the graph and model the state dynamics as

$$\dot{x}_i(t) = \sum_{j \in \mathscr{N}_i} a_{ij} x_j(t),$$

where $\mathscr{N}_i \subseteq \mathscr{V}$, $\mathscr{N}_i = \{j : \exists (i, j) \in \mathscr{E}\}$ denotes the set of in-neighbors of node i.

Example 1 (Vehicle Platooning) Consider a group of N vehicles moving along a single lane as in Fig. 1. In a platooning scenario [1], vehicles follow one another and share their state information (e.g., position, velocity, acceleration) with other vehicles by communicating through a V2V communication protocol. The behavior of the i-th vehicle in the platoon, $i \in \{1, \ldots, N\}$, can be described by the two differential equations representing an inertial agent:

$$\dot{r}_i(t) = v_i(t), \qquad \dot{v}_i(t) = \frac{1}{m_i} u_i(t),$$

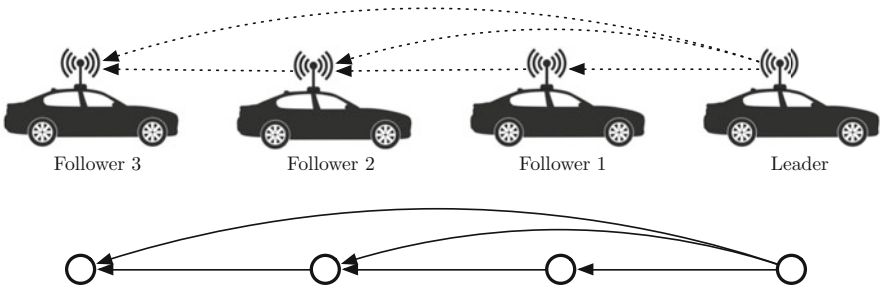


Fig. 1 Vehicle platooning and associated topology. A group of vehicles is traveling along a single lane while maintaining a desired inter-vehicle spacing and a certain steady-state speed. To accomplish this task, each vehicle exchanges information with the platoon leader and the vehicle immediately ahead

where $r_i : \mathbb{R}_{\geq 0} \to \mathbb{R}$, $v_i : \mathbb{R}_{\geq 0} \to \mathbb{R}$, and $m_i \in \mathbb{R}_{>0}$ denote the i-th vehicle position, velocity, and mass, respectively.

The goal of maintaining a desired inter-vehicle spacing can be formulated as the problem of controlling the position and velocities of each vehicle toward the following desired steady-state values:

$$r_i(t) \to \frac{1}{N} \sum_{j=1}^{N} \left(r_j(t) + d_{ij} \right), \qquad v_i(t) \to \bar{v},$$

where $\bar{v}$ denotes the desired steady-state platoon velocity and d_{ij} is the desired spacing distance between agent i and j. The desired steady-state spacing and velocity can be achieved through a double-integrator consensus protocol [18], of the form

$$u_i(t) = \sum_{j=1}^{N} \alpha_{ij}(r_i(t) - r_j(t) - d_{ij}) + \gamma_{ij}(v_i(t) - v_j(t)),$$

where $\sum_{j=1}^{N} \alpha_{ij} = \sum_{j=1}^{N} \gamma_{ij} = 1$ for all $i \in \{1, \ldots, N\}$. Therefore, the goal of attaining a platooning configuration can be achieved by modeling each vehicle as a two-node subsystem with states r_i and v_i, respectively, and dynamics

$$\dot{r}_i(t) = v_i(t),$$

$$\dot{v}_i(t) = \frac{1}{m_i} \sum_{j=1}^{N} \alpha_{ij}(r_i(t) - r_j(t) - d_{ij}) + \gamma_{ij}(v_i(t) - v_j(t)),$$

Thus, the above scenario belongs to the more general class of models considered in this work. □

In order to implement the described cooperation protocol, each node is required to transmit the state over a (potentially lossy) communication channel to all its neighbors. For ideal communication channels, the signal transmitted by agent j and received by agent i coincides; therefore, network dynamics can be modeled by a continuous LTI system as

$$\dot{x}(t) = Ax(t), \tag{1}$$

where $x : \mathbb{R}_{\geq 0} \to \mathbb{R}^n$ contains the node states and $A \in \mathbb{R}^{n \times n}$ is the adjacency matrix of the network. We will make the assumption that the adjacency matrix in (1) is (marginally) Hurwitz, that is, for all $z \in \{z \in \mathbb{C} : \ \det(zI - A) = 0\}$, $\Re(z) \leq 0$.

2.2 Attack Model

Common transmission protocols often reframe signals into streams of data packets before transmission. We assume that this underlying process is intangible, and we will equivalently refer to signal streams or to streams of packets in the remainder. We consider attacks that target communication channels and delay the stream of information in the path between transmitter and receiver (Fig. 2). In order to implement the cooperative protocol (1), we assume that every node $j \in \{1, \ldots, N\}$ shares the current value of the state $x_j(t)$ with the set of available neighbors and denote by $r(i, j, t) : \mathcal{V} \times \mathcal{V} \times \mathbb{R}_{\geq 0} \to \mathbb{R}$ the corresponding continuous-time signal received at node i, $i \in \{1, \ldots, n\}$. In general, the relation

$$r(i, j, t) = x_j(t),$$

may not be satisfied due to the lossy nature of the communication channels. Notably, these have the effect of altering the content of transmitted data packets and/or of introducing time delays in the signal streams. We consider scenarios where attackers can maliciously exploit these features in order to compromise the correct functionality of the system. We make the following assumptions:

1. The attacker does not alter the information contained in transmitted signals.
2. There exists an upper bound $\tau^{\max}$ to the largest packet delay.
3. Data is used as soon as it becomes available at the receiver.

While scenarios where attackers alter the content of the transmitted signals have been extensively studied in previous works (see, e.g., [19]), we argue that malicious attacks targeting the communication timing can lead to similar disruptive behaviors.

Remark 1 (Compensation Mechanisms) In the presence of communication delays, two compensation mechanisms are often adopted. Either data that is classified as obsolete (for instance, by time stamping the transmitted packets) is discarded at the receiver or data is used as soon as it is available at the receiver [20]. We consider scenarios where the latter protocol is used. □

We model received signals in the presence of attacks as

$$r(i, j, t) = x_j(t - \tau_{ij}),$$

where $\tau_{ij} \in \mathbb{R}_{\geq 0}$, $0 \leq \tau_{ij} \leq \tau^{\max}$, for all $i, j \in \{1, \ldots, n\}$, represent (deterministic) time delays introduced by the attacker. Then, the dynamics of agent i in the presence

Fig. 2 Time-delay attacks can occur in the communication channel between every pair of nodes

of attacks can be written as

$$\dot{x}_i(t) = \sum_{j \in \mathcal{N}_i} a_{ij} r(i, j, t) = \sum_{j \in \mathcal{N}_i} a_{ij}\, x_j(t - \tau_{ij}).$$

We denote by $\mathcal{A} \subseteq \mathcal{E}$ the set edges under attack, that is,

$$\mathcal{A} = \{(i, j) : \tau_{ij} > 0\}.$$

Then, the time evolution of the network state can be written as

$$\dot{x}(t) = \bar{A} x(t) + \sum_{(i,j) \in \mathcal{A}} \tilde{A}_{ij}\, x(t - \tau_{ij}), \tag{2}$$

where $\tilde{A}_{ij} \in \mathbb{R}^{n \times n}$,

$$\tilde{A}_{ij}(p, q) = \begin{cases} a_{ij} & \text{if } p = i, q = j, \text{ and } \tau_{ij} > 0, \\ 0 & \text{otherwise,} \end{cases}$$

for all $p, q \in \{1, \ldots, n\}$, and $\bar{A} = A - \sum_{(i,j) \in \mathcal{A}} \tilde{A}_{ij}$.

Example 2 (Transmitter Delay and Receiver Delay Attacks) Scenarios where an attacker compromises the behavior of a certain network node and deliberately transmits (receives) delayed messages can be modeled as in (2). For instance, consider the circumstance where a compromised node intentionally (i) transmits obsolete information to all its neighbors or (ii) updates its state with obsolete neighboring data. These two classes of vulnerabilities are referred to as *transmitter delay attacks* and *receiver delay attacks*, respectively, and are discussed next.

Transmitter delay attacks, illustrated in Fig. 3(a), model scenarios where a certain time shift is intentionally introduced in all the packets transmitted from an agent to its neighbors. Let $i \in \{1, \ldots, n\}$ denote the (single) agent under attack; then

$$r(j, i, t) = x_i(t - \tau)$$

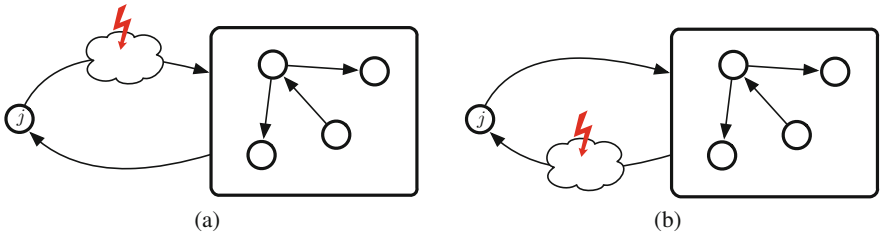


Fig. 3 Illustration of (**a**) transmitter delay attack and (**b**) receiver delay attack. Red patterns represent attacker intrusions

for all j such that $i \in \mathcal{N}_j$. Moreover, the network model under transmission delay attack can be written as

$$\dot{x}(t) = \bar{A}x(t) + \tilde{A}x(t-\tau),$$

where $\tilde{A}$ has only n nonzero entries corresponding to its i-th column, that is,

$$\tilde{A}(p,q) = \begin{cases} a_{pq} & \text{if } q = i \\ 0 & \text{otherwise,} \end{cases}$$

for all $p, q \in \{1, \ldots, n\}$, and $\bar{A} = A - \tilde{A}$.

Receiver delay attacks, illustrated in Fig. 3(b), model scenarios where the attacker prevents a timely state update of a certain node. This, for instance, can be the result of overloading the local processing units of the node. Let $i \in \{1, \ldots, n\}$ denote the (single) node under attack; then

$$r(i, j, t) = x_j(t-\tau),$$

for all $j \in \mathcal{N}_i$. Moreover, the network model under resources overload attack can be written as

$$\dot{x}(t) = \bar{A}x(t) + \tilde{A}x(t-\tau),$$

where $\tilde{A} \in \mathbb{R}^{n\times n}$ has only n nonzero entries corresponding to its i-th row,

$$\tilde{A}(p,q) = \begin{cases} a_{pq} & \text{if } p = i, \\ 0 & \text{otherwise,} \end{cases}$$

and $\bar{A} = A - \tilde{A}$.

2.3 Problem Formulation

In this work, we focus on attacks that aim at compromising the stability properties of (2). Next, we recall a standard definition of convergence.

Definition 1 (Convergence Criteria) The time evolution of system state in (2) is convergent to a limit vector $\bar{x} \in \mathbb{R}^n$ if, for every $\epsilon > 0$, there exist $\bar{t} \in \mathbb{R}_{\geq 0}$ such that

$$\|x(t) - \bar{x}\| \leq \epsilon, \quad \text{for all } t \geq \bar{t}.$$

We recall that the convergence of (2), in general, depends on the nominal adjacency matrix A, the attack set $\mathscr{A}$, and the time delays τ_{ij}. We then restrict our analysis to the uniform delay case, that is, on the model:

$$\dot{x}(t) = \bar{A}x(t) + \tilde{A}x(t-\tau), \tag{3}$$

and focus on the following problem.

Problem 1 Find the minimal cardinality attack set $\mathscr{A}^*$ that makes dynamics (3) non-convergent.

3 Minimum Cardinality Attack Sets

In this section we propose a numerical technique to solve Problem 1. Recall that the trajectories of (3) are convergent if and only if all the characteristic roots, which are the zeros of[1]

$$\det(sI_n - \bar{A} - \tilde{A}e^{-s\tau}) = 0, \qquad s \in \mathbb{C},$$

where $I_n \in \mathbb{R}^{n\times n}$ denotes the identity matrix, are in the open left half plane (see, e.g., [7]). We then report a well-known result for time-delay dynamical systems that will be needed in the subsequent analysis.

Theorem 1 ([21, Theorem 4.1]) *Consider the delayed dynamical system* (3)*, and define the dual characteristic equation*

$$\det(sI_n - \bar{A} - \tilde{A}e^{-j\theta}) = 0, \tag{4}$$

where $s \in \mathbb{C}$ *and* $\theta \in [0, 2\pi]$*. The time evolution of* (3) *is convergent if and only if any solution* s *to* (4) *satisfies*

$$s \in \{s = \sigma + j\omega : \sigma \in \mathbb{R}, \omega \in \mathbb{R}, \sigma < 0\} \cup \{0\},$$

for all $\theta \in [0, 2\pi]$.

The following comments are in order. First, Theorem 1 allows us to simplify the nonlinear dependency of the primal characteristic equation from variable s by introducing the independent variable θ. Second, since θ only affects the coefficients of the polynomial (4), the corresponding roots are continuous functions of θ. Therefore, the roots of (4) form closed curves in the complex plane as θ is varied over the interval $[0, 2\pi]$. It follows that the convergence of linear dynamical systems

[1] This will be referred to as primal characteristic equation.

in the presence of delayed communication edges can be assessed through the study of the real part of the eigenvalues of the pencil $\bar{A} + \tilde{A}e^{-j\theta}$ as the scalar parameter θ is varied over $[0, 2\pi]$. Third, by comparing the primal and dual characteristic equations, it immediately follows that if $s = j\omega$ is a root of (4) for a fixed value of θ, then the choice $\tau = \theta/\omega$ will satisfy the primal equation. In the remainder, we will use the compact notation (s, θ) to denote a root s of (4) associated with a fixed θ.

The following result provides a characterization of the roots of (4).

Lemma 1 (Hermitian Property) *Let the pair (s, θ) denote a solution to (4), where $s = \sigma + j\omega$, $\sigma \in \mathbb{R}$, $\omega \in \mathbb{R}$. Then, $(\bar{s}, -\theta)$ is also a solution to (4), with $\bar{s} = \sigma - j\omega$.*

Proof Recall that $\det\left(sI_n - \bar{A} - \tilde{A}e^{-j\theta}\right) = 0$ if and only if

$$(\bar{A} + \tilde{A}e^{-j\theta})v = (\sigma + j\omega)v,$$

for some $v \in \mathbb{C}^n$. By taking the complex conjugate of the above equation, we obtain

$$(\bar{A} + \tilde{A}e^{j\theta})\bar{v} = (\sigma - j\omega)\bar{v},$$

where $\bar{v}$ denotes the complex conjugate of v that proves the claimed statement. □

The conjugate property illustrated in the above lemma, combined with the periodic relation $e^{-j\theta} = e^{j(2\pi-\theta)}$, implies that the curves describing the roots of (4) for $\theta \in [0, \pi]$ are the complex conjugate of the curves describing the corresponding roots for $\theta \in (\pi, 2\pi)$. An illustration of the behavior of the roots of the dual characteristic equation (4) as a function of θ is presented in Fig. 4.

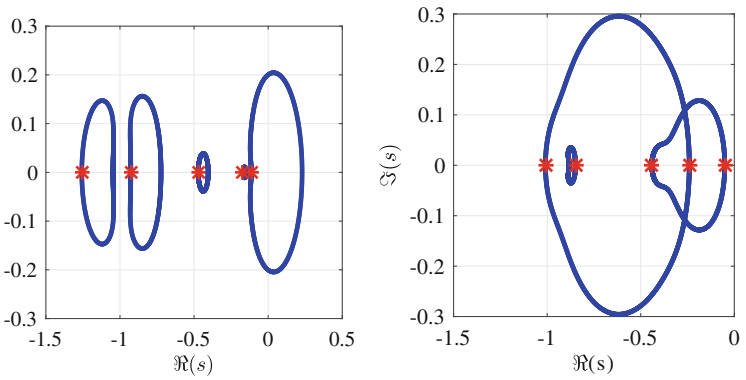


Fig. 4 Numerical study of the roots of (4) for two realizations of a full adjacency matrix A with uniform entries in the interval [0, 1]. As highlighted in the comparison, the roots (**a**) may cross the imaginary axis, or (**b**) may not cross the imaginary axis. The symbols $*$ represents the roots for $\theta = 0$

We now employ the above characterization of time-delay linear systems for the solution of Problem 1. Let $\Psi = [\psi_{ij}] \in \mathbb{R}^n$, with $\psi_{ij} \in \{0, 1\}$, and decompose the network adjacency matrix as

$$A = \underbrace{(1_{n\times n} - \Psi) \circ A}_{\bar{A}_\Psi} + \underbrace{\Psi \circ A}_{\tilde{A}_\Psi},$$

where $1_{n\times n} \in \mathbb{R}^{n\times n}$ denotes a n by n matrix of ones and $\circ$ denotes the Hadamard operator. The notation $\bar{A}_\Psi$ and $\tilde{A}_\Psi$ is employed to emphasize the dependency on Ψ. We then formalize Problem 1 as the following minimization problem: given the network adjacency matrix A and an upper bound to the largest communication delay $\tau^{\max}$, determine the delayed adjacency matrix $\Psi \circ A$ satisfying

$$\begin{aligned}
\Psi^* = \quad & \underset{\Psi,\theta,v,\omega}{\arg\min} \quad \|\Psi\|_{\ell_1} \\
& \text{subject to} \quad A = \underbrace{(1_{n\times n} - \Psi) \circ A}_{\bar{A}_\Psi} + \underbrace{\Psi \circ A}_{\tilde{A}_\Psi}, && \text{(5a)} \\
& (\bar{A}_\Psi + \tilde{A}_\Psi e^{-j\theta})v = j\omega v, && \text{(5b)} \\
& \psi_{ij} \in \{0, 1\}, && \text{(5c)} \\
& \theta \le \omega\tau^{\max}, && \text{(5d)}
\end{aligned}$$

where $i, j \in \{1, \dots, n\}$. It should be observed that (5) is of the form of a mixed-integer optimization problem, where the Boolean variables ψ_{ij}, the real variables θ, ω, and the complex variable v are the optimization parameters. Two major complexities arise in solving (5). First, the optimization variables ψ_{ij} are integers. Second, the variables $\tilde{A}_\Psi$, θ, v, and ω are related by the nonlinear constraint (5b). It is also worth noting that the feasibility of the constraint set depends on the largest allowed time delay $\tau^{\max}$, and it is independent on the nominal adjacency matrix A. To see this, we observe that for any A with eigenvalues $\lambda_1, \dots, \lambda_n$, by letting $\theta = \pi/2$ and $\Psi = 1_{n\times n}$, then $\bar{A} + \tilde{A}e^{-j\theta} = Ae^{-j\theta}$ has eigenvalues $j\lambda_1, \dots, j\lambda_n$. It follows that, the feasible set of (5) is always nonempty.

3.1 Optimal Delay Attacks

We now reformulate minimization problem (5) to facilitate its solution. We perform two simplifying steps to rewrite the Hadamard product (5a) and the eigenvalue constraint (5b).

Let $\mathrm{vec}(M) = [m_{11}, \ldots, m_{m1}, m_{12}, \ldots, m_{mn}]$ denote the vectorization of matrix $M = [m_{ij}] \in \mathbb{R}^{m\times n}$, and let $\mathrm{diag}(v) \in \mathbb{R}^{n\times n}$ denote a diagonal matrix with diagonal entries given by the elements of vector $v \in \mathbb{R}^n$. Then, the Hadamard products in (5a) are linear functions of the entries of Ψ, as formalized in the following result.

Lemma 2 (Linearity of Hadamard Product) *Let* $\bar{A}_\Psi = (1_{n\times n} - \Psi) \circ A$ *and* $\tilde{A}_\Psi = \Psi \circ A$. *Then*

$$\mathrm{vec}\left(\bar{A}_\Psi\right) = \mathrm{diag}\left(\mathrm{vec}(A)\right)\left(1_{n^2} - \mathrm{vec}(\Psi)\right),$$
$$\mathrm{vec}\left(\tilde{A}_\Psi\right) = \mathrm{diag}\left(\mathrm{vec}(A)\right)\mathrm{vec}(\Psi),$$

where $1_{n^2} \in \mathbb{R}^{n^2}$ *denotes the vector of all ones.*

Proof The claimed statement can be verified by inspection. □

Next, we drop the dependency of constraint (5b) on the complex variable v.

Lemma 3 (Rank Constraint) *Let* $\bar{A}_\Psi + \tilde{A}_\Psi = A$. *There exists a solution* $v = v_\Re + j v_\Im$, $\omega \in \mathbb{R}_{\geq 0}$ *and* $\theta \in [0, 2\pi]$ *to* (5b) *if and only if*

$$\mathrm{Rank}\,(\Lambda_\Psi) < 2n,$$

where

$$\Lambda_\Psi = \begin{bmatrix} -\tilde{A}_\Psi \sin\theta - \omega I_n & A + \tilde{A}_\Psi(\cos\theta - 1) \\ -A - \tilde{A}_\Psi(\cos\theta - 1) & -\tilde{A}_\Psi \sin\theta - \omega I_n \end{bmatrix}. \tag{6}$$

Proof By substituting $\bar{A}_\Psi = A - \tilde{A}_\Psi$, $e^{-j\theta} = \cos\theta - j\sin\theta$, and $v = v_\Re + j v_\Im$ into (5b) and by expanding the products, we obtain

$$\left(A + \tilde{A}_\Psi(\cos\theta - 1) - j\tilde{A}_\Psi \sin\theta\right)(v_\Re + j v_\Im) = j\omega(v_\Re + j v_\Im),$$

or equivalently, by separating real and imaginary parts,

$$(A + \tilde{A}_\Psi(\cos\theta - 1))v_\Re + \tilde{A}_\Psi \sin\theta\, v_\Im = -\omega v_\Im,$$
$$\left(A + \tilde{A}_\Psi(\cos\theta - 1)\right) v_\Im - \tilde{A}_\Psi \sin\theta\, v_\Re = \omega v_\Re.$$

The two equations above can be collected together and rewritten in matrix form as

$$\underbrace{\begin{bmatrix} -\tilde{A}_\Psi \sin\theta & A + \tilde{A}_\Psi(\cos\theta - 1) \\ -A + \tilde{A}_\Psi(\cos\theta - 1) & -\tilde{A}_\Psi \sin\theta \end{bmatrix}}_{\Lambda_\Psi} \begin{bmatrix} v_\Re \\ v_\Im \end{bmatrix} = \omega \begin{bmatrix} v_\Re \\ v_\Im \end{bmatrix},$$

from which the claimed statement follows. □

These simplifications lead to the following result.

Lemma 4 (Equivalent Minimization Problem) *Let Λ_Ψ be defined as in (6) and let $\bar{A}_\Psi + \tilde{A}_\Psi = A$, where $\tilde{A}_\Psi$ satisfies*

$$\operatorname{vec}\left(\tilde{A}_\Psi\right) = \operatorname{diag}\left(\operatorname{vec}(A)\right)\left(1_{n^2} - \operatorname{vec}(\Psi)\right).$$

The following minimization problem is equivalent to (5)

$$\begin{aligned} \Psi^* = \underset{\Psi,\theta,\omega}{\textit{arg min}} \quad & \|\Psi\|_{\ell_1} \\ \textit{subject to} \quad & \operatorname{Rank}(\Lambda_\Psi) < 2n, \\ & \psi_{ij} \in \{0,1\}, \\ & \theta \le \omega \tau^{\max}. \end{aligned} \tag{7}$$

It should be noticed that the two simplifying steps performed allow us to (i) write the entries of Λ_Ψ as linear functions of the optimizing variables ψ_{ij}, and (ii) discard the dependency of the optimization problem from the complex variable v. In the next section, we further simplify the optimization problem (7) and propose a numerical method to find an approximate solution.

3.2 Numerical Methods for Finding Optimal Attacks

We observe that the optimization problem (7) is not convex because of (i) the presence of integer optimization variables Ψ, (ii) the nonlinear relation between Λ_Ψ and θ in (6), and (iii) the rank constraint that is nonlinear in the entries of Λ_Ψ. We now develop a numerical method to find a delayed set of edges that can be used to gain information about the solution to (7). First, we relax the original integer variables by letting ψ_{ij} vary on the interval $[0, 1]$. Second, we emphasize that rank constraints produce challenging nonconvex feasible sets, for which all known finite-time algorithms have exponential running times [22]. We will therefore focus on proposing a heuristic that solves a relaxed version of the problem (7). A good heuristic is a tractable method that in practice will solve the considered optimization problem, although there is no guarantee on its optimality.

Recent works (see, e.g., [22]) propose to relax rank constraints to constraints on the nuclear norm of the considered matrix. Formally, for a (nonnecessarily square) matrix $M \in \mathbb{R}^{m\times n}$, the nuclear norm is defined as

$$\|M\|_* = \sum_{i=1}^{\min\{m,n\}} \sigma_i(M),$$

where σ_i denotes the i-th singular value of M. The nuclear norm is a convex function that can be optimized efficiently and is a good convex approximation of the rank

function [23]. Loosely speaking, $\|M\|_*$ represents the ℓ_1-norm of the vector of the singular values of M; therefore, constraining the nuclear norm will promote sparsity in such vector [22]. Thus, we consider the relaxed version of the rank constraint (7), that is,

$$\|\Lambda_\Psi\|_* < 2n. \tag{8}$$

Nuclear norm regularization constraints can be reformulated in the form of SDP constraints [23], as formalized in the following result.

Lemma 5 (SDP Constraint) *There exists Λ_Ψ that satisfies (8) if and only if there exist symmetric matrices $M \in \mathbb{S}^{n\times n}$ and $N \in \mathbb{S}^{n\times n}$ that satisfy*

$$\begin{bmatrix} M & \Lambda_\Psi \\ \Lambda_\Psi^\mathsf{T} & N \end{bmatrix} \succeq 0, \quad \text{and} \quad \text{Trace}\,(M+N) = n - \frac{1}{2}.$$

Proof The proof follows immediately from [23, Lemma 1]. □

These simplifications lead to the following relaxed version of (7):

$$\begin{aligned} \hat{\Psi}^* = \underset{\Psi,\theta,\omega,M,N}{\arg\min} \quad & \|\Psi\|_{\ell_1} \\ \text{subject to} \quad & \begin{bmatrix} M & \Lambda_\Psi \\ \Lambda_\Psi^\mathsf{T} & N \end{bmatrix} \succeq 0, \\ & \text{Trace}\,(M+N) = n - \frac{1}{2}, \\ & \psi_{ij} \in [0,1], \\ & \theta \le \omega \tau^{\max}, \end{aligned} \tag{9}$$

where $M \in \mathbb{S}^{n\times n}$, $N \in \mathbb{S}^{n\times n}$, and Λ_Ψ are defined in (6). We observe that the feasible set in (9) is a convex set in the optimization variables Ψ, ω, M, and N, as all its constraints are linear functions of these variables. In the next section, we numerically solve (9) for fixed θ and present how the resulting solutions provide an insight on the relation between the smallest cardinality attack sets and network topology.

4 Optimal Attack Sets and Relation with Topology

This section discusses numerical simulations in support of the approximate solution method proposed in Sect. 3, and includes numerical investigations that provide useful insights regarding the resilience of different network topologies under attack.

We first focus on numerically evaluating the optimality gap between the optimization problem (5) and its relaxation (9). Recall that Ψ^* denotes the true combinatorial optimal solution to (5) and $\hat{\Psi}^*$ denotes the solution of the convex relaxation (9). We observe that, in general, the inequality $\|\hat{\Psi}^*\|_{\ell_1} \leq \|\Psi^*\|_{\ell_1}$ holds as the feasible set of the combinatorial problem is a subset of the feasible set of (9). We employ a rounding algorithm that uses the solution of the convex relaxation with objective value $\hat{\Psi}^*$ to produce a feasible integer solution with (possibly suboptimal) value $\hat{\Psi}^*_{\text{FEAS}}$. The relation between Ψ^*, $\hat{\Psi}^*$, and $\hat{\Psi}^*_{\text{FEAS}}$ is depicted in Fig. 5.

To evaluate the optimality gap, that is, the gap between $\|\Psi^*\|_{\ell_1}$ and $\|\hat{\Psi}^*_{\text{FEAS}}\|_{\ell_1}$, we consider graphs constructed by interconnecting nodes randomly [24], where each edge is included in the graph with probability $p = 1/2$, independent from every other edge. Edge weights are chosen randomly in the interval $[0, 1]$, and θ is chosen equal to $\pi/2$. A Monte Carlo simulation obtained by sampling from the above set of graphs is illustrated in Fig. 6, where a feasible optimal solution is compared with the combinatorial solution Ψ^* for increasing network sizes. The comparison shows that feasible solutions originated from the relaxed problem (9) represent, in this scenario, accurate approximations of the combinatorial optimal solution.

Next, we employ the proposed optimization technique to compare the robustness of different network topologies against timing attacks. We consider (i) the class of random graphs with edge probability $p = 1/2$, (ii) the line topology (Fig. 7a), and (iii) the platooning formation (Fig. 7b). Figure 8 shows a comparison between the norms $\|\hat{\Psi}^*\|_{\ell_1}$ and $\|\hat{\Psi}^*_{\text{FEAS}}\|_{\ell_1}$ for increasing network sizes. It is worth noting that,

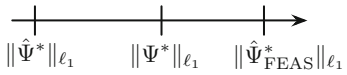


Fig. 5 Relation $\|\Psi^*\|_{\ell_1} \leq \|\hat{\Psi}^*\|_{\ell_1} \leq \|\hat{\Psi}^*_{\text{FEAS}}\|_{\ell_1}$ and optimality gap for the considered problem

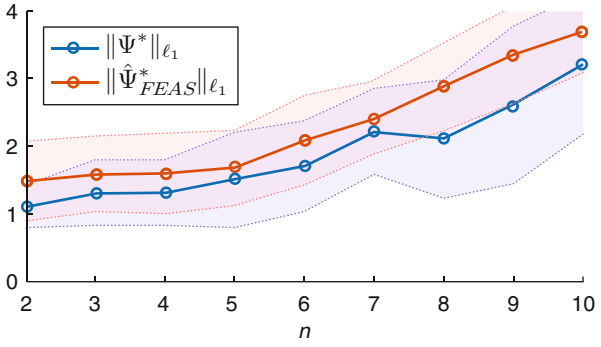


Fig. 6 Monte Carlo simulation illustrating the optimality gap for random graphs where edges between each pairs of nodes have probability $p = 1/2$. Solid lines represent the mean value over the sample and colored bands illustrate standard deviation. Sample size is chosen equal to 10

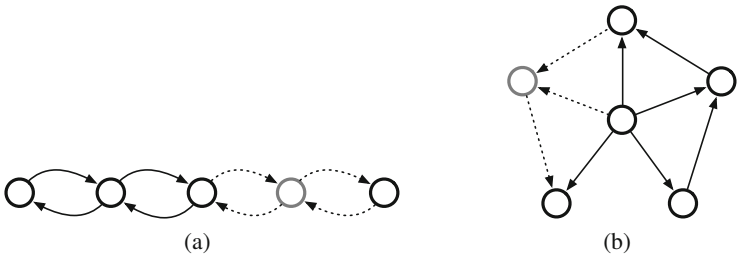


Fig. 7 Considered topologies for variable n, (**a**) line, (**b**) platoon

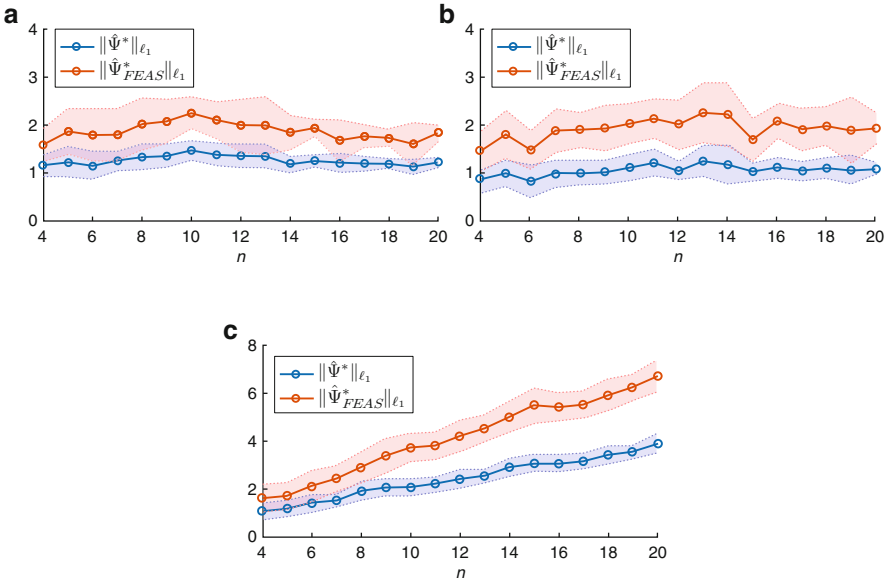


Fig. 8 Norm of optimal attacks for different network topologies. (**a**) Line topology, (**b**) platooning formation, (**c**) random graphs with $p = 1/2$. Solid lines represent the mean value over the sample and colored bands illustrate standard deviation. Sample size is chosen equal to 10

while solving the combinatorial problem (5) is prohibitive for significantly large n, $\|\hat{\Psi}^*\|_{\ell_1}$ and $\|\hat{\Psi}^*_{\text{FEAS}}\|_{\ell_1}$ provide a lower bound and an upper bound to this quantity, respectively (Fig. 5).

The comparison shows that the resilience of line and platoon topology degrades for increasing network sizes n, as opposed to the class of random graphs. We interpret this result by observing that the average degree[2] of the nodes in the random graphs scales with the network size, as opposed to the constant degree of

[2] $|\mathcal{N}_i|$ represents the degree of node i, $i \in \{1, \ldots, n\}$.

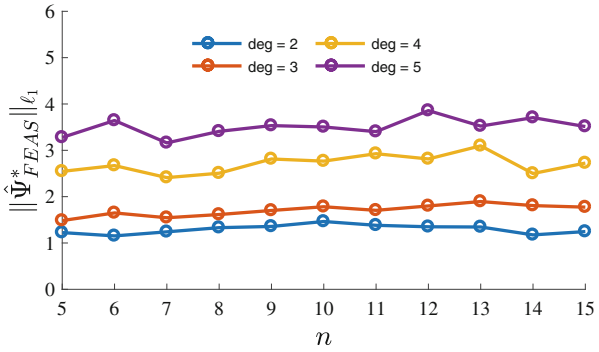


Fig. 9 Mean value of Monte Carlo simulations for optimal attacks to graphs with fixed degree for all nodes. Edge weights are uniform in the interval [0, 1], and sample size is chosen equal to 10

the nodes in the two topologies in Fig. 7. This consideration suggests a relation between attack resilience and the degree distribution of the nodes in the network. To validate this interpretation, we consider random graphs where all nodes have fixed identical degree and compare optimal attacks as a function of this parameter. The comparison shown in Fig. 9 numerically validates this claim and suggests that improved robustness can be achieved by designing networks with large node degrees.

5 Conclusions

This work defines and studies a class of attacks that tamper with the temporal characteristic of the communication channels, leading to time delays in the signals exchanged between adjacent nodes. Differently from considering conventional channel communication delays, the problem of securing network systems from intentional and specific timing aberrations sets out new security challenges and design goals. In addition to providing a framework to characterize and study timing attacks from a control perspective, this work proposes numerical ways and algorithms to identify links that are inherently less robust to tampering. Our methods suggest that improved robustness can be achieved by designing network topologies in which all nodes have large degree distributions. The numerical nature of the proposed study motivates more rigorous formalization in future works.

References

1. M. di Bernardo, A. Salvi, S. Santini, Distributed consensus strategy for platooning of vehicles in the presence of time-varying heterogeneous communication delays. IEEE Trans. Intell. Transp. Syst. **16**(1), 102–112 (2015)
2. R. Poovendran, K. Sampigethaya, S.K.S. Gupta, I. Lee, K.V. Prasad, D. Corman, J.L. Paunicka, Special issue on cyber-physical systems. Proc. IEEE **100**(1), 6–12 (2012)
3. J. Slay, M. Miller, Lessons learned from the Maroochy water breach, in *International Conference on Critical Infrastructure Protection* (Springer, Berlin, 2007), pp. 73–82
4. J.P. Farwell, R. Rohozinski, Stuxnet and the future of cyber war. Survival **53**(1), 23–40 (2011)
5. A.A. Cárdenas, S. Amin, Z.-S. Lin, Y.-L. Huang, C.-Y. Huang, S. Sastry, Attacks against process control systems: risk assessment, detection, and response, in *Proceedings of the 6th ACM Symposium on Information, Computer and Communications Security* (ACM, New York, 2011), pp. 355–366
6. F. Pasqualetti, F. Dörfler, F. Bullo, Attack detection and identification in cyber-physical systems. IEEE Trans. Autom. Control **58**(11), 2715–2729 (2013)
7. J.K. Hale, E.F. Infante, F.-S.P. Tsen, Stability in linear delay equations. DTIC Document, Tech. Rep., 1982
8. E.A. Lee, The past, present and future of cyber-physical systems: a focus on models. Sensors **15**(3), 4837–4869 (2015)
9. R. Olfati-Saber, R.M. Murray, Consensus problems in networks of agents with switching topology and time-delays. IEEE Trans. Autom. Control **49**(9), 1520–1533 (2004)
10. A. Seuret, D.V. Dimarogonas, K.H. Johansson, Consensus under communication delays, in *47th IEEE Conference on Decision and Control, CDC 2008* (IEEE, Piscataway, 2008), pp. 4922–4927
11. P.-A. Bliman, LMI characterization of the strong delay-independent stability of linear delay systems via quadratic Lyapunov–Krasovskii functionals. Syst. Control Lett. **43**(4), 263–274 (2001)
12. X. Li, C.E. De Souza, Delay-dependent robust stability and stabilization of uncertain linear delay systems: a linear matrix inequality approach. IEEE Trans. Autom. Control **42**(8), 1144–1148 (1997)
13. S.-I. Niculescu, Stability and hyperbolicity of linear systems with delayed state: a matrix-pencil approach. IMA J. Math. Control Inf. **15**(4), 331–347 (1998)
14. J. Chen, G. Gu, C.N. Nett, A new method for computing delay margins for stability of linear delay systems, in *Proceedings of the 33rd IEEE Conference on Decision and Control*, vol. 1 (IEEE, Piscataway, 1994), pp. 433–437
15. Y. Shoukry, J. Araujo, P. Tabuada, M. Srivastava, K.H. Johansson, Minimax control for cyber-physical systems under network packet scheduling attacks, in *Proceedings of the 2nd ACM International Conference on High Confidence Networked Systems* (ACM, New York, 2013), pp. 93–100
16. J. Moon, T. Başar, Minimax control over unreliable communication channels. Automatica **59**, 182–193 (2015)
17. G. Fiore, Y.H. Chang, Q. Hu, M.D. Di Benedetto, C.J. Tomlin, Secure state estimation for cyber physical systems with sparse malicious packet drops, in *American Control Conference (ACC)* (IEEE, Piscataway, 2017), pp. 1898–1903
18. W. Ren, R.W. Beard, *Distributed Consensus in Multi-Vehicle Cooperative Control* (Springer, Berlin, 2008)
19. F. Pasqualetti, F. Dörfler, F. Bullo, Attack detection and identification in cyber-physical systems. IEEE Trans. Autom. Control **58**(11), 2715–2729 (2013)
20. J.P. Hespanha, P. Naghshtabrizi, Y. Xu, A survey of recent results in networked control systems. Proc. IEEE **95**(1), 138–162 (2007)
21. W. Michiels, S.-I. Niculescu, Characterization of delay-independent stability and delay interference phenomena. SIAM J. Control Optim. **45**(6), 2138–2155 (2007)

22. M. Fazel, Matrix rank minimization with applications, Ph.D. dissertation, Stanford University, 2002
23. M. Jaggi, M. Sulovsk, et al., A simple algorithm for nuclear norm regularized problems, in *Proceedings of the 27th International Conference on Machine Learning (ICML-10)* (2010), pp. 471–478
24. P. Erdös, A. Rényi, On random graphs, I. Publ. Math. Debr. **6**, 290–297 (1959)

Attack Tree Construction and Its Application to the Connected Vehicle

Khaled Karray, Jean-Luc Danger, Sylvain Guilley, and M. Abdelaziz Elaabid

Abstract Remote connectivity of today's and future cars increases their capabilities of autonomy and safety, but also their attack surface, as reported by several research papers. In the automotive domain, the security has a direct impact on the user's safety. Thus, the management of risk is becoming the main concern of automotive manufacturers, especially for the future fully connected and autonomous cars. A possible way to quantify the overall risk of a system is the systematic construction of attack graphs and attack trees. These formalisms are presented as one of the possible solutions in the new Cybersecurity Guidebook for Cyber-Physical Vehicle Systems (SAE-J3061). In this chapter we propose to use graph transformation to formally model the car architecture and its state evolution in order to study cyber-physical attacks against it. The resulting attacks are converted into attack trees which are used to estimate the overall risk of the system. Consequently, it becomes possible to study improvements while building a more secure architecture. The proposed method is designed to support the conceptual phase of the vehicle's cyber-physical system. We illustrate the method on a small pedagogical example to show how it is possible to prove its efficiency.

K. Karray (✉)
Télécom ParisTech, Paris, France
e-mail: khaled.karray@telecom-paristech.fr

J.-L. Danger
Télécom ParisTech, Paris, France

Secure-IC S.A.S., Cesson-Sévigné, France
e-mail: jean-luc.danger@telecom-paristech.fr

S. Guilley
Télécom ParisTech, Paris, France

Secure-IC, Paris, France

École normale supérieure, Paris, France
e-mail: sylvain.guilley@telecom-paristech.fr; sylvain.guilley@secure-ic.com

M. Abdelaziz Elaabid
PSA-GROUPE, Paris, France
e-mail: abdelaziz.elaabid@mpsa.com

Ç. K. Koç (ed.), *Cyber-Physical Systems Security*,
https://doi.org/10.1007/978-3-319-98935-8_9

1 Introduction

During the last 15 years, the automotive domain has been subject to many developments that allow car manufacturers to enhance most of features by digital processing assistance. These changes helped to build more robust, safer, more comfortable, and user-friendly cars. Later developments introduced the car to remote connectivity technologies that allowed even more advanced functionality to be integrated in the car. Wi-Fi and Bluetooth connectivity to smartphone and cellular connectivity offer a plethora of possibilities for the user. Unfortunately these technologies exposed the car to the outside world, and like every connected device, the car became one of the targets for hackers [6, 15]. Motivation of such hackers could go from a simple privacy violation where the goal is to steal private information from the car owner or the car manufacturer to a more dangerous scenario that could threaten user safety.

Since the car used to be a closed system, hypothesis about some trusted domains that used to be true in a not connected car is no longer valid, especially when it comes to the internal communication buses. An attacker may leverage these trust relations along with some code vulnerabilities to retrieve, compromise, and steal private information or, even worse, take control over the entire vehicle or some of its functionalities [4, 6, 15].

In fact, in the last years, some researchers began to report multiple important issues related to the design and implementation of the car architecture. In [8] an early work of Hoppe et al. pointed out the threats on in-vehicle bus networks based on characterizations carried out on a simulated bench. These conclusions have been further confirmed later in [12] and [4] that performed CAN frame injections on a real vehicle but with direct physical access to the communication bus of the car. In a more advanced attack, Miller et al. reported in [15] that the physical access to the vehicle is not necessary if an attacker can find some code vulnerability that allows him to reach these communication buses using another wireless attack vector. In general, the recent works performed in this area have shown that combinations of exploits and mis-configurations are the typical means by which an attacker breaks into a car communication bus. Facing this increasingly growing threat, car manufacturers have to guarantee a certain security level of the equipment embedded in the automobile. The obvious approach to this problem is to conduct a security assessment study. The goal of the security assessment is to identify the assets and the associated attack scenarios regarding availability, integrity, and confidentiality. There are available methods like EBIOS and TVRA that could be adapted to conduct such a study in the automotive domain. Ultimately the study will help the manufacturer decide where to best spend the security budget. To do so, the risk is evaluated relatively to each attack scenario based on its impact and its likelihood.

$$Risk = \sum_i Impact(sc_i) \times P_{occ}(sc_i) \qquad \text{where}$$

- $\{sc_i\}$ is the set of identified attack scenarios.
- $Impact$ is a function that evaluates the impact of a given scenario.
- P_{occ} is the likelihood or the probability of occurrence of the given scenario.

While the impact of the attack scenario has to be defined by the security experts, determining what attack scenarios are likely to occur is a little more complex and strongly depends on the given architecture. Eventually security experts have to imagine every possible way the attacker can exploit the system in order to reach her objective. A fairly good way to model these attacks and to document them is to use attack trees or the attack graphs.

Attack trees have been introduced in [21] as a useful way to document and understand attacks on a given system and most importantly is a way of making decisions about how to improve the security of the target system. The root node in an attack tree represents the attack goal (or attack scenario), and leaf nodes represent basic attacks. Each node in the tree is either an [AND] node or an [OR] node. An [AND] node has child nodes that represent different steps of achieving the goal, and an [OR] node has child nodes that represent different ways of achieving the goal. Attack graphs are also a good way to document attacks. They are composed of vertices (that represent the system states) and edges (that represent attacks performed on the system).

Attack trees are well designed to support risk assessment studies [13]. Nevertheless, the elaboration of attack trees can be a tedious task and error prone for large systems. This is why automated techniques to generate such representations of attacks have been proposed.

1.1 Attack Trees in the Automotive Domain

In the automotive domain, little work has been conducted in such direction. To the best of our knowledge, the work of Salfer et al. [19, 20] is the only one that proposes such approach. In [20], Salfer et al. present automated attack tree generation as a reachability problem of assets inside the cyber-physical architecture of the vehicle. Nevertheless, the proposed model focuses on scalability issue using heuristic techniques and does not address the exhaustivity of the attack paths.

Lugou et al. [14] and Apvrille et al. [3] use SysML-Sec modeling language to model safety and security aspects of the car architecture and formally prove safety (with Uppaal) and security (with ProVerif) properties. In [2] Apvrille et al. explain how to use an input attack graph modeled with SysML-Sec for the verification of a system. The issue of how to create such an attack graph is not addressed; in other words the attack scenarios are not automatically generated and need to be manually fed to the tool.

1.2 Attack Tree Generation

In contrast automated generation of attack trees has been addressed in other domains, especially in network security and enterprise security [1, 9, 11, 17].

In [17] Phillips et al. build an attack graph based on topology and vulnerability information; they also use the attack graph to identify attack paths with high probability or low cost. In [18] Ritchey et al. used a model checker to provide single attack scenarios to depict vulnerabilities due to the configuration of various hosts in a network. The pieces of information about the network are fed to a model checker and then assert that an attacker cannot acquire a given privilege on a given host. The model checker provides a counterexample (the attack steps) in case the assertion is false. As an extension of this work, in [22] Sheyner et al. present an automated method to analyze a network of hosts with known vulnerabilities and produce an attack graph that depicts *all* possible ways for the attacker to reach his goal. Later works focused on reducing the complexity of the approaches. In [1] Ammann et al. propose a scalable attack graph generation based on the monotonicity assumption (an exploit never invalidates another exploit). In [16] Ou et al. introduced a logic-based approach for network security analysis. The method relies on inference rules implemented on a modified version of the XSB inference engine to depict *all* attack paths combining vulnerabilities in a network. In [9] Ingols et al. use network configuration data to automatically compute network reachability, classify vulnerabilities, and build an attack graph used to recommend actions to improve network security. In [11] Jajodia et al. use topological information to analyze vulnerability dependencies and assess the impact of individual and combined vulnerabilities on overall security, then identify key vulnerabilities, and provide strategies for protection.

The problem has also been investigated for enterprise security domain [5, 10]. The goal is to implement enterprise security policy against possible “insider attack” or attacks that leverage certain “trust” relations and social interactions between actors (employees). Thus efforts focused on modeling trust relations and asset mobility. In [10] Ivanova et al. present a general framework of a model for enterprise security and how to transform this model to an attack tree that exploits possible trust relations between actors. In [5] only the modeling aspect of the problem is discussed and focuses also on trust relations and asset mobility.

Those generation techniques rely on models that are not suited for the automotive domain. However the general approach could be adapted. This approach is more or less the same for all of the presented works : first the real system is abstracted in a model that captures only the important aspects. Second the modeled system is expressed using the language of an inference engine (model checker, Horn clauses, etc.). It is then processed by the inference engine whose output is a set of possible attacks to be analyzed.

1.3 Contributions

In this chapter we **first** propose a method to model elements of the cyber-physical architecture of the vehicle using graphs. The model captures the security policy implemented as well as vulnerability information and access rights. Besides we consider an attacker model as a set of attacks originating from *all* the attack vectors (short range, long range, and indirect physical access). The system and attacker are modeled with behavioral rules using graph transformation system.

Second, we use the model to generate possible attack paths (combinations of actions) that can be used by the attacker to drag the system into a vulnerable state. Thus the generated attacks are more detailed and we can capture more information about the possible attacker actions. The simulation of this behavioral model will allow us to find *all* vulnerable states and to retrace attacker actions that allowed him to reach it. Using this information we generate an attack tree that summarizes all possible steps that allow the attacker to reach his goal.

Based on such model, we can try to answer questions like:

- Is a vulnerable configuration/state reachable from an initial state? In other words is an attack scenario achievable on the proposed architecture?
- Which sequence of basic attacks the attacker has to perform in order to reach such vulnerable state?

In what follows, Sect. 2 gives some preliminary notions and definitions in the automotive domain and introduces the formal modeling language used. In Sect. 3 we explain how to generate attack scenarios. Section 4 shows how to deduce the attack tree necessary risk assessment. The small example introduced in Sect. 2.2 helps the reader understand the methodology throughout the chapter. And finally Sect. 6 concludes the paper.

2 Background and Definitions

In this section we introduce to the reader some notions relative to the automotive domain that will be used in the reminder of the article, we introduce the modeling language that we used to build the model, and finally we present a small example that we will use in the following sections to illustrate the different steps.

2.1 Automotive Architecture

The cyber-physical architecture of the car is composed of multiple components that could be categorized in four main categories:

- *Sensors*: these are components whose role is to report information about the state of the vehicle (speed, closed/open doors, break/acceleration,etc.) and its

surroundings (vision radars, ...). The data that produced by the sensors are sent to Electronic Control Units (ECUs) to be processed.
- *Actuators*: these are the components that transform commands coming from different ECUs into actions (engine, wheel orientation, ...)
- *ECUs*: short for Electronic Control Units that are the most important part of the architecture. In general they are composed of hardware electronic components (memories, microcontrollers, etc.) that have a processing capacity and that embed algorithms (software) needed to ensure the control of every single functionality inside the vehicle from breaking to air-conditioning and more advanced functionality that ensure the user's comfort (e.g., Internet connectivity, smart applications, etc.. . .)
- *Communication buses*: they are an important part of the architecture as they represent the main medium of communication between the ECUs. Multiple technologies of buses could be found in today's cars, e.g., CAN, FlexRay, Ethernet, and others. Each technology has some characteristics that justify its presence between certain ECUs: robustness, throughput, etc. For historical reasons and up until lately, these technologies (CAN protocol being on top of the list) did not implement security mechanisms as these communications were assumed to be "trusted."

Some of the ECUs inside this architecture implement advanced services that open the whole architecture to the outside world. Examples of these services are internet connection, Bluetooth, Wi-Fi for smartphones, etc. Given the fact that automotive architecture used to be a closed system and that internal communication buses still used were not designed with security in mind, the new attack vectors expose the architecture to sophisticated attacks that leverage these "trusted" relations.

2.2 Architectural Graph

In this section we identify key elements of the automotive architecture introduced in Sect. 2.1 and the relations between them. Using these elements, we model the cyber-physical system of the car using graphs: the main idea is to model communication buses, hardware components of the ECUs (including the sensors and actuators), and the software components as graph nodes. Arcs in the graph model relations between the nodes. Figure 1 represents an example of an architectural graph.

- Service node:

  Automotive services are built around the notion of function blocks, which emphasizes the connection of inputs and outputs to core software modules. Those software modules are then mapped to different ECUs. Automotive software frameworks such as AUTOSAR are particularly designed to support such architecture. We base the model around this concept. A service is modeled as a node in the graph. To communicate data, services use read and write access to shared memory or to/from network hardware, sensors and actuators. These

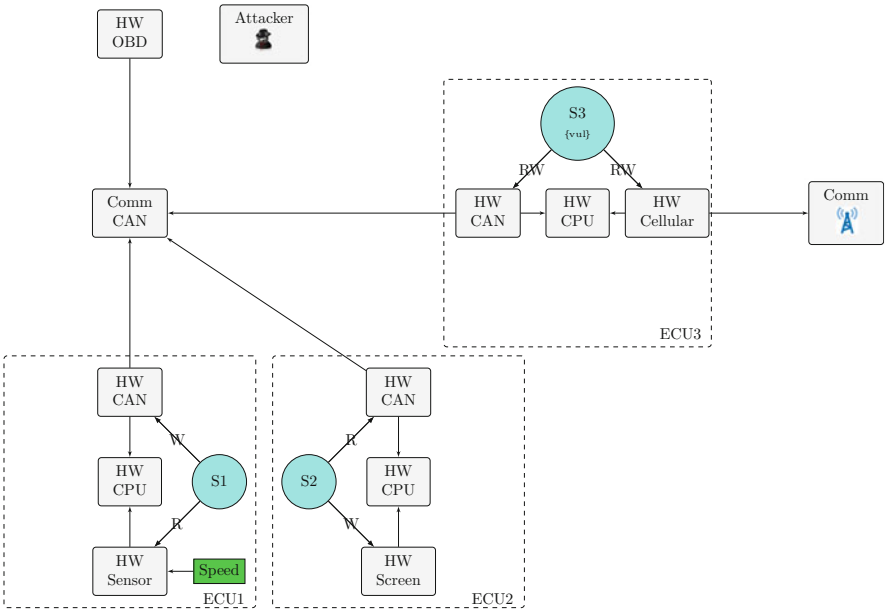


Fig. 1 Small example

access rights are modeled with arcs directed to the corresponding hardware elements. Besides access rights, we also capture the vulnerability information of the modeled service.

- Hardware node:
  A hardware node is a special node that abstracts a hardware component: it is used to represent sensors, actuators, memory components, communication controllers, etc. Software nodes have access rights over the hardware nodes.
- Communication node:
  A communication node models communication mechanisms between hardware nodes that are used to exchange data. It is designed to model a bus communication between multiple hardware nodes as well as a point-to-point communication between only two hardware nodes.
- Data node:
  A data node models a data asset that can be located on a service node, a hardware node, or a communication node.

Running Example

To clarify the model, let us consider the example of Fig. 1. In this example we propose an architecture composed of three ECUs connected to the same CAN bus. The CAN bus is connected to the OBD port: ECU-1 contains a CAN transceiver that allows it to communicate over the CAN bus, a processing unit and a speed sensor.

On the CPU runs a service whose job is to make the speed acquisition and send it over the CAN bus. This service has a read access to the sensor and a write access to the CAN transceiver. On ECU-2 we have also a CAN transceiver, a processing unit, and a screen. On this processing unit runs a service whose job is to read the speed information from the CAN bus and to pass it to the screen to be displayed. Thus it has a read access over the CAN transceiver and a write access over the screen hardware. ECU-3 is connected to both the CAN bus and the cellular network. We suppose that on the CPU of this ECU runs a vulnerable service that has a read/write access to the cellular hardware and a read/write access to the CAN transceiver. ECU-1 is supposed to sense the speed and send it over the CAN bus to be displayed by ECU-2 over the screen.

2.3 Graph Transformation

In this section we briefly introduce graph transformation system (GTS) as a rule-based modeling approach that allows to capture the structural as well as behavioral aspects of a system. We use it as the underlying formal modeling language supporting the methodology.

A graph transformation system is a formal approach for structural modifications of graphs via the application of transformation rules. A graph transformation system is thus a tuple (G, R) where G is a graph and R is a set of transformation rules. A typed GTS is a GTS where each element of the graph is assigned a type. Transformation rules are then type preserving. We consider typed graphs. A graph transformation rule consists of a left-hand side graph L, a right-hand side graph R, a Negative Application Condition (NAC), and a mechanism specifying how to transform L into R when the NAC is satisfied.

In general we model three types of transformation rules:

- Transformation rules to describe the behavior of services (one or multiple rules for each service). This rule is conditioned by the availability of the input data. We assume that as soon as the input data are all available, the transformation rule can be triggered. The effect of this transformation rules will be to delete the input data (consumed by the service) and to create the output data with the correct output type and made available for other services. Some transformation rules may add an attribute to the input data node when there is no need to delete and create another data "type."
- Transformation rules to describe the normal behavior of the hardware components. For each type of hardware node, we define a behavioral model. For instance, a *memory* node used to store *data* accepts a *data* node only from a service that has a write (w) access right on it and also can transfer data only to a service that have a read (r) access right on it.
- Transformation rules to describe the attacker actions: the behavior of the attacker is modeled with transformation rules that represent basic attacks or actions that the attacker can perform on the system to interact with it.

Example

As an example let us introduce transformation rules that we model for the architectural graph (Fig. 1) of the example introduced above.

1. To describe the behavior of the *speed-acquisition* service (S1), we implement the transformation rule of Fig. 2. This rule means that if the speed data is available, the *speed-acquisition* service (S1) will read a *speed* data from the *speed sensor*.

   The transformation rule of Fig. 3 means that is the speed data is available on the *speed-acquisition* service (S1) and that service has the *write* access right to the CAN hardware; then the service can send the data the CAN hardware.
2. To describe the behavior of the CAN hardware, we implement the transformation rule of Fig. 4. Note that at this stage, it does not matter if the data is a speed data or not. This is mainly because this behavioral rule is designed to send any data

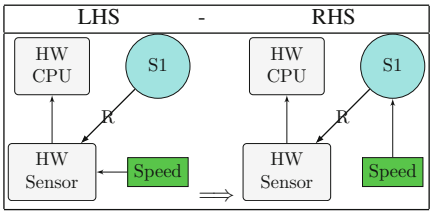


Fig. 2 Speed-acquisition rule (R1)

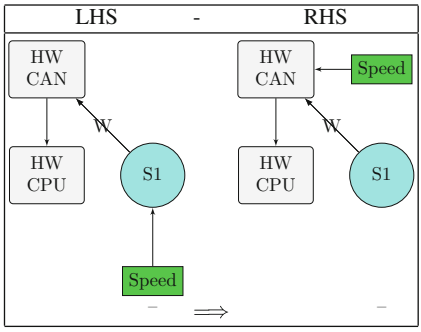


Fig. 3 Speed send to CAN rule (R2)

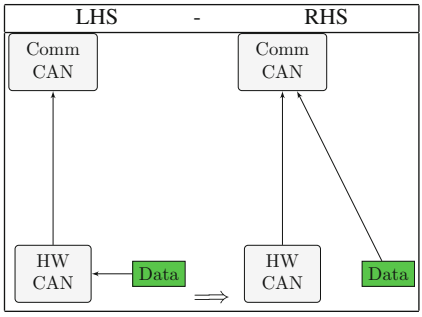


Fig. 4 CAN send rule (R3)

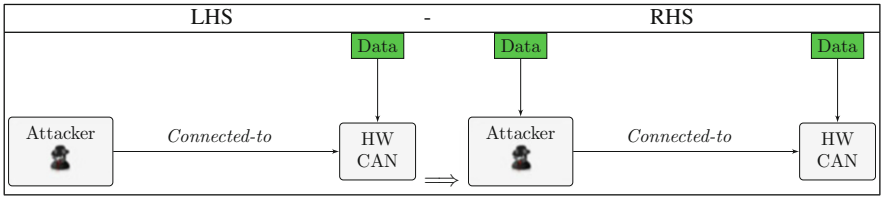


Fig. 5 Example of attacker rule

on the CAN bus. The rule is also common to all CAN hardware as opposed to some rules that are sometimes specific to one in particular.

3. To describe the behavior of the attacker, we model transformation rules that gives her the ability to connect to any communication link and to read and write data on that link. She is also able to exploit vulnerabilities and modify the collected data and replay it. The transformation rule (Fig. 5) could be read as follows: if the attacker (Att) is connected to the CAN network, and if there is a data packet transiting on the CAN network, then the attacker can copy the data.

3 Attack Graph Generation

Given a start graph, and a set of transformation rules, the recursive application of the transformation rules on the start graph will generate a state space which represents all possible states that could be generated from the set of transformation rules. In the state space, each state represents a graph, and a transition between two states (source and destination) represents a rule application that allowed the transformation of the graph from the source state to match that of the destination state. For instance, the application of the transformation rules of Figs. 2 and 3 for the modeled example of Fig. 1 will produce the state space of Fig. 6.

The modeled rules are a combination of attacker actions (or basic attacks) and rules that describe the behavior of the modeled elements. The produced state space contains transitions that model both attacker steps and element behavior. By definition of the attack graph, the state space contains the attack graph.

Given a particular attack scenario (attacker objective), we have to make a query to find states in the state space where the scenario is realized (the attacker has reached its objective). Thus queries allow to detect a vulnerable state. They are expressed also in the form of graph. Figure 7 gives an example of a query that allows us to detect if there is a state of the system where the screen displays a *modified* speed. In practice we model the architectural graph and transformation rules using a tool named GROOVE [7]. This tool allows the transformation of the input model and produces the associated state space.

In the next section, we will process this state space to capture only attacker actions in the form of an attack tree.

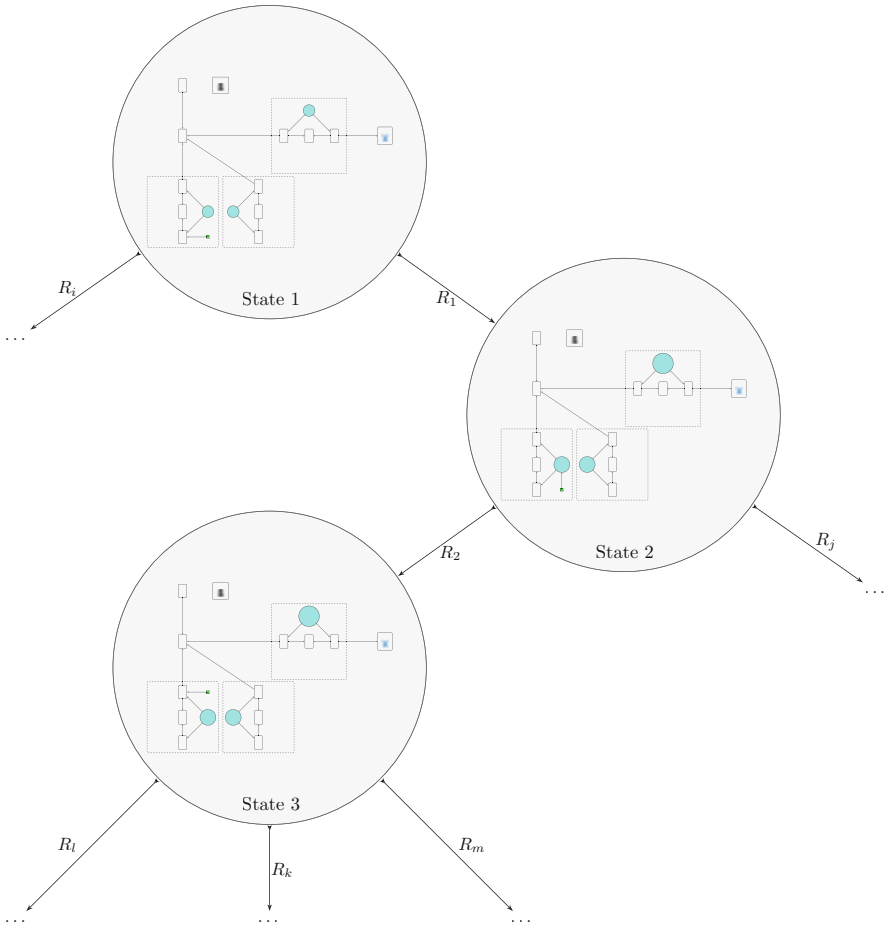


Fig. 6 Application of transformation rules (state space)

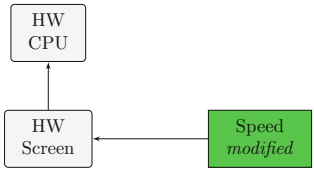


Fig. 7 Query: false speed

4 Attack Tree Generation

The generated state space is quite complex and large (for the introduced small example, we have 768 states and 2860 transitions). Besides it includes transitions that describe the behavior of the system as well as transitions that describe basic

attacks performed by the attacker. A convenient way to reduce the information is to transform this state space into an attack tree that only combines attacker actions. The attack tree procedure will allow us to discard all the transitions that are not basic attacks and to output, in the form of a tree, a compact representation of complex attacks. Based on the attack graph (state space) generated in the previous section, we follow a simple way to transform this attack graph into an attack tree. Let (G, R) be an attack graph with a start state S_0. Let S_v be a vulnerable state of the system (i.e., a state where a security breach has been detected). The security breach detected by that state is placed on the root of the attack tree. The goal is to collect all sequences of transformations (attacker actions only) that led to that state. We explore the attack graph from the target state (S_v) backward to the start state: each time we encounter a state with more than one incoming edges, we place an [Or] node in the attack tree (meaning that there is more than one way to reach that state), and each time we encounter a state with only one incoming edge, we place an [And] node in the attack tree. And finally each time we encounter a state with more than one outgoing edges, we place a sub-tree and check if we already computed that sub-tree. Sub-trees are attack trees that are present more than once inside a single attack tree (Fig. 8).

Using the example introduced in Fig. 1, we make a query (Fig. 7) to detect vulnerable states where the attacker can force the system into a state where the

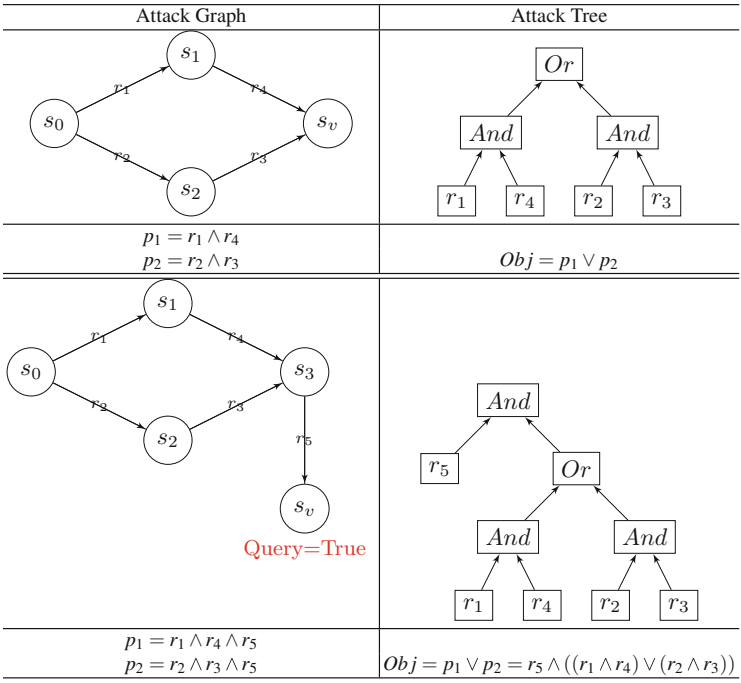


Fig. 8 Example of attack graph to attack tree transformation

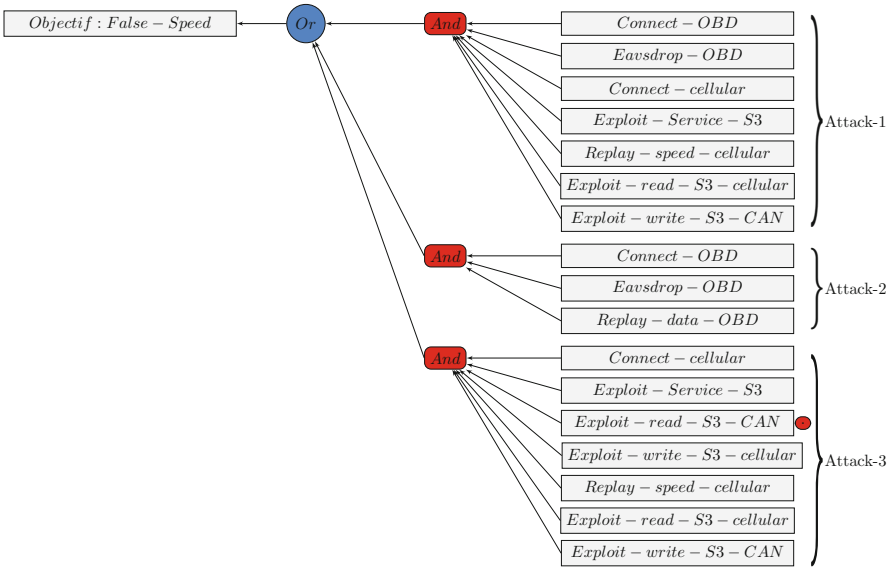


Fig. 9 Attack tree automatically produced from input model Fig. 1

screen displays a *modified* speed. The results produced by the methodology are shown in Fig. 9.

The attack tree shows that the analysis of the modeled architecture identified three attacks.

- Attack-1: where the attacker connects to the OBD port, eavesdrops on the CAN bus to dump the speed frame, then connects to the vehicle from the cellular network, exploits an exposed vulnerable service (S3) that has a write access on the CAN controller, and then replays the data
- Attack-2: where the attacker connects to the OBD port, eavesdrops on the CAN bus to dump the speed frame, and then replays it from the OBD port on the same CAN bus.
- Attack-3: where the attacker only operates from the cellular, connects to the cellular, exploits the exposed service, and then uses that service to eavesdrop on the CAN hardware to dump and then replay the speed frame

5 Countermeasure

We are interested in attack-3 where the attacker only operated from the cellular network attack vector. This attack seems important as it does not require a physical access to the vehicle. Analyzing the steps of the attack, if we can prevent at least

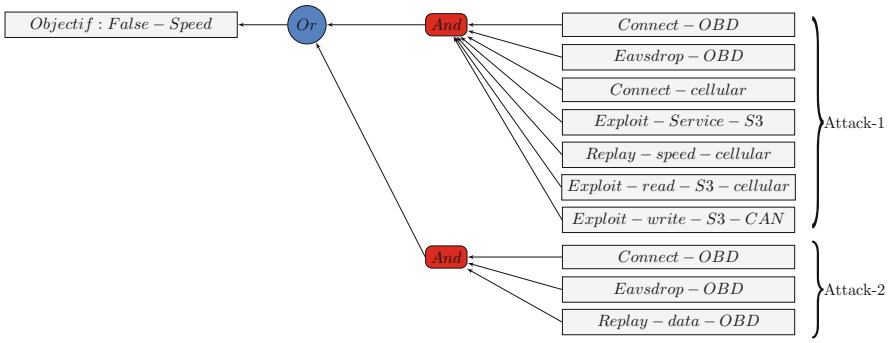


Fig. 10 Attack tree automatically produced from Architecture-1

one of the basic attacks from happening, we can prevent the whole attack from happening as it is an [And] node. It seems possible to either deploy a patch for the vulnerable service or to revoke the read access right of the service to the CAN controller. As a short-term solution, we opt for the second choice. Rerunning the simulation with the modified architecture produces the attack graph presented in Fig. 10.

We can see that now there are only two attack options for the attacker, since the service does not have a read access right to the CAN controller; the third attack scenario is no longer valid and hence not reported in the attack tree. The overall impact of the objective is affected because through the changes we made, we were able to block the attack path.

6 Conclusion

The chapter presents a modeling methodology using graph transformation to construct attack trees in order to analyze attacks to a connected vehicle. The attack tree synthesizes all possible attack paths with respect to the model and, thus, serves as the basis for further analysis. Impact quantification and sensitivity analysis can be conducted given such attack tree whose goal is to improve the overall security of an automotive architecture during design phase. The described methodology has nevertheless certain limitations due to required input data, which are:

- A structural and behavioral model of the service nodes
- A structural and behavioral model for hardware components
- An attacker model

From the car manufacturer point of view, the fact that the modeling methodology requires architectural information of service nodes can be considered as a limitation as some software architecture and implementation tasks are outsourced to other companies.

References

1. P. Ammann, D. Wijesekera, S. Kaushik, Scalable, graph-based network vulnerability analysis, in *Proceedings of the 9th ACM Conference on Computer and Communications Security* (ACM, New York, 2002), pp. 217–224
2. L. Apvrille, Y. Roudier, Sysml-sec attack graphs: compact representations for complex attacks, in *International Workshop on Graphical Models for Security* (Springer, Berlin, 2015), pp. 35–49
3. L. Apvrille, L. Li, Y. Roudier, Model-driven engineering for designing safe and secure embedded systems, in *Architecture-Centric Virtual Integration (ACVI), 2016* (IEEE, Piscataway, 2016), pp. 4–7
4. S. Checkoway, D. McCoy, B. Kantor, D. Anderson, H. Shacham, S. Savage, K. Koscher, A. Czeskis, F. Roesner, T. Kohno, et al., Comprehensive experimental analyses of automotive attack surfaces, in *USENIX Security Symposium*, San Francisco (2011)
5. T. Dimkov, W. Pieters, P. Hartel, Portunes: representing attack scenarios spanning through the physical, digital and social domain, in *Joint Workshop on Automated Reasoning for Security Protocol Analysis and Issues in the Theory of Security* (Springer, Berlin, 2010), pp. 112–129
6. I.D. Foster, A. Prudhomme, K. Koscher, S. Savage, Fast and vulnerable: a story of telematic failures, in *WOOT'15 Proceedings of the 9th USENIX Conference on Offensive Technologies* (2015)
7. Groove: graphs for object-oriented verification. http://groove.cs.utwente.nl/
8. T. Hoppe, S. Kiltz, J. Dittmann, Security threats to automotive can networks–practical examples and selected short-term countermeasures, in *International Conference on Computer Safety, Reliability, and Security* (Springer, Berlin, 2008), pp. 235–248
9. K. Ingols, R. Lippmann, K. Piwowarski, Practical attack graph generation for network defense, in *22nd Annual Computer Security Applications Conference, 2006. ACSAC'06* (IEEE, Piscataway, 2006), pp. 121–130
10. M.G. Ivanova, C.W. Probst, R.R. Hansen, F. Kammüller, Transforming graphical system models to graphical attack models, in *International Workshop on Graphical Models for Security* (Springer, Berlin, 2015), pp. 82–96
11. S. Jajodia, S. Noel, Topological vulnerability analysis, in *Cyber Situational Awareness* (Springer, Berlin, 2010), pp. 139–154
12. K. Koscher, A. Czeskis, F. Roesner, S. Patel, T. Kohno, S. Checkoway, D. McCoy, B. Kantor, D. Anderson, H. Shacham, et al., Experimental security analysis of a modern automobile, in *2010 IEEE Symposium on Security and Privacy (SP)* (IEEE, Piscataway, 2010), pp. 447–462
13. R. Kumar, E. Ruijters, M. Stoelinga, Quantitative attack tree analysis via priced timed automata, in *International Conference on Formal Modeling and Analysis of Timed Systems* (Springer, Berlin, 2015), pp. 156–171
14. F. Lugou, L.W. Li, L. Apvrille, R. Ameur-Boulifa, Sysml models and model transformation for security, in *Conferénce on Model-Driven Engineering and Software Development (Modelsward'2016)* (2016)
15. C. Miller, C. Valasek, Remote exploitation of an unaltered passenger vehicle. Black Hat USA (2015)
16. X. Ou, S. Govindavajhala, A.W. Appel, Mulval: a logic-based network security analyzer, in *USENIX Security* (2005)
17. C. Phillips, L.P. Swiler, A graph-based system for network-vulnerability analysis, in *Proceedings of the 1998 Workshop on New Security Paradigms* (ACM, New York, 1998), pp. 71–79
18. R.W. Ritchey, P. Ammann, Using model checking to analyze network vulnerabilities, in *SP'00 Proceedings of the 2000 IEEE Symposium on Security and Privacy* (IEEE, Piscataway, 2000), pp. 156–165
19. M. Salfer, C. Eckert, Attack surface and vulnerability assessment of automotive electronic control units, in *2015 12th International Joint Conference on e-Business and Telecommunications (ICETE)*, vol. 4 (IEEE, Piscataway, 2015), pp. 317–326

20. M. Salfer, H. Schweppe, C. Eckert, Efficient attack forest construction for automotive on-board networks, in *International Conference on Information Security* (Springer, Berlin, 2014), pp. 442–453
21. B. Schneier, Attack trees. Dr. Dobbâs J. **24**(12), 21–29 (1999)
22. O. Sheyner, J. Haines, S. Jha, R. Lippmann, J.M. Wing, Automated generation and analysis of attack graphs, in *2002 Proceedings IEEE Symposium on Security and Privacy* (IEEE, Piscataway, 2002), pp. 273–284

Reinforcement Learning and Trustworthy Autonomy

Jieliang Luo, Sam Green, Peter Feghali, George Legrady, and Çetin Kaya Koç

Abstract Cyber-Physical Systems (CPS) possess physical and software interdependence and are typically designed by teams of mechanical, electrical, and software engineers. The interdisciplinary nature of CPS makes them difficult to design with safety guarantees. When autonomy is incorporated, design complexity and, especially, the difficulty of providing safety assurances are increased. Vision-based reinforcement learning is an increasingly popular family of machine learning algorithms that may be used to provide autonomy for CPS. Understanding how visual stimuli trigger various actions is critical for trustworthy autonomy. In this chapter we introduce reinforcement learning in the context of Microsoft's AirSim drone simulator. Specifically, we guide the reader through the necessary steps for creating a drone simulation environment suitable for experimenting with vision-based reinforcement learning. We also explore how existing vision-oriented deep learning analysis methods may be applied toward safety verification in vision-based reinforcement learning applications.

1 Introduction

Cyber-Physical Systems (CPS) are becoming increasingly powerful and complex. For example, standard passenger vehicles have millions of lines of code and tens of processors [1], and they are the product of years of planning and engineering

J. Luo (✉) · S. Green · P. Feghali · G. Legrady
University of California Santa Barbara, Santa Barbara, CA, USA
e-mail: jieliang@ucsb.edu; sam.green@cs.ucsb.edu; peterfeghali@ucsb.edu; glegrady@ucsb.edu

Ç. K. Koç
İstinye University, İstanbul, Turkey

Nanjing University of Aeronautics and Astronautics, Nanjing, China

University of California Santa Barbara, Santa Barbara, CA, USA
e-mail: cetinkoc@ucsb.edu

Ç. K. Koç (ed.), *Cyber-Physical Systems Security*,
https://doi.org/10.1007/978-3-319-98935-8_10

between diverse teams. In similar ways, both smaller CPS, like smart locks, and larger CPS, like electric power transmission systems, require teams with diverse skills. These modern systems are being augmented with full or partial autonomy. Because of the physical aspect of certain CPS, it is critical that they operate reliably and safely. However, the interdisciplinary nature of CPS makes them difficult to design with safety and security guarantees. When autonomy is incorporated, design complexity and, more importantly, the difficulty of providing safety and security assurances are increased. Yet because of the benefits of autonomy, we will continue to see its use expand.

Reinforcement learning (RL) is a family of machine learning algorithms aimed at providing autonomy, and these algorithms are being used to provide autonomous capabilities to CPS. RL has historic roots in dynamic programming and the psychological concepts of operant conditioning, or learning by trial and error. RL methods are flexible. For example, an RL policy may be developed using simulation and then transferred to a physical agent, or a policy may incorporate prior knowledge about the environment, or a policy may be trained tabula rasa. Because of the flexibility of these methods, RL with CPS is ideally suited for interdisciplinary development, with each field bringing distinct capability-enhancing contributions. On the other hand, an individual can develop basic autonomous methods with RL which may later be improved.

In mammals, vision processing accounts for a high percentage of neural activity. Likewise, in CPS, visuomotor control will be an important area of research, e.g., for self-driving cars or package delivery drones. Understanding the visual stimuli that influences behavior is critical for safe autonomy. Advanced RL methods typically process visual inputs with convolutional neural networks (CNNs). In 2012, CNNs gained popularity when the AlexNet architecture became the first CNN method to win the ImageNet competition [2]. At that time, CNNs were treated like black box functions that worked well, but it was difficult to determine why. Since then, a number of visualization algorithms have been devised to provide introspection into the behavior of a CNN. Such methods may also be applied in the context of RL in order to provide insight into the reliability of a particular CNN.

In this chapter we introduce reinforcement learning in the context of Microsoft's AirSim drone simulator. AirSim is a physics-based simulator which enables experimentation with self-driving and flight applications. We have extended the simulator to support teaching a drone how to autonomously navigate a sequence of waypoints (Fig. 1). The source code for these experiments is available at https://github.

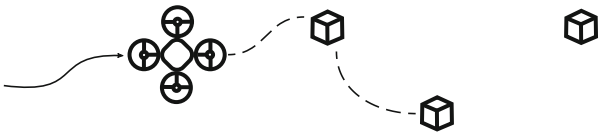


Fig. 1 In this chapter we train a drone to use its camera to perform path planning. Training is performed via reinforcement learning, and the goal is to learn vision-to-action mappings which allow the drone to collect cubes

com/RodgerLuo/CPS-Book-Chapter. After an introduction to RL and AirSim, we explain how to use CNN visualization techniques to increase trust in the safety of RL-based drone control.

2 Reinforcement Learning Preliminaries

Reinforcement learning is a family of methods aimed at training an **agent** to collect rewards from an environment. At each **time step** t, the agent is given information about the **state** of its environment in the form of an observation s_t and then makes an **action** a_t. The agent's **policy** $\pi(s_t)$ is the logic which takes state observations and returns action selections. After each action the environment will return a new state observation s_{t+1} and **reward** r_{t+1}. This cycle is illustrated in Fig. 2.

The agent's policy is parameterized by the tensor θ, giving it a more explicit notation of $\pi_\theta(s_t)$. For example, in a linear model, the policy would have the form:

$$\pi_\theta(s) = \theta_1 s_1 + \theta_2 s_2 + \cdots + \theta_m s_m = \theta^\top s, \tag{1}$$

where the time step t has been dropped for notation clarity and m is the number of features in the state space. In a similar manner, a neural network would be parameterized by a parameter tensor. The goal of the agent is to maximize collection of rewards from the environment. In a finite time horizon, the goal is accomplished by finding parameters $\theta^\star$ which provide this maximization:

$$\theta^\star = \arg\max_\theta \sum_{t=0}^{T-1} r(s_t, a_t), \tag{2}$$

where $T-1$ is the number of time steps experienced and $r(s_t, a_t)$ is the environment's reward function. In many real-world cases, the reward function is not given as a closed-form expression, but must be sampled by the agent's interactions with the environment. One of the strengths of RL is its ability to learn using this experiential method.

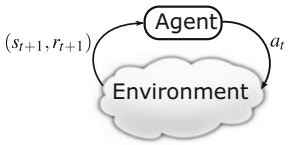


Fig. 2 In reinforcement learning, an agent interacts with an environment. At each time step, the agent receives a state and reward signal from the environment. Based on this information, the agent selects its next action

In the remainder of this section, we describe attributes of the environment in which RL agents are assumed to exist, and we introduce a common method to solve the objective given in Eq. (2). This background will prepare the reader for drone control tasks described in Sect. 3 and for approaching more efficient methods found in literature.

2.1 Markov Decision Processes

Markov Decision Processes (MDPs) are the environments which reinforcement learning was developed to solve. A major aspect of MDPs is that they have states in which an agent exists, and the outcomes of actions depend only on the current state, not on past states and actions; in this sense MDPs are **memoryless**. The memoryless property is captured in the environment's state-transition and reward function notation:

$$\begin{aligned} p(s_{t+1}|s_t, a_t, s_{t-1}, a_{t-1}, \ldots, s_0, a_0) &= p(s_{t+1}|s_t, a_t), \\ r(s_{t+1}|s_t, a_t, s_{t-1}, a_{t-1}, \ldots, s_0, a_0) &= r(s_{t+1}|s_t, a_t). \end{aligned} \tag{3}$$

The state-transition and reward functions in Eq. (3) state that function outcomes depend only on the current state and action and are independent of past states and actions.

A second major aspect of MDPs is that the state-transition and/or reward functions may be **stochastic**, which means their return values are drawn from some underlying probability distributions. In standard RL settings, these distributions must be **stationary** which means the probabilities do not shift over time. Methods exist for using RL in nonstationary environments. Investigating such advanced methods is critical for using RL in safety-critical CPS applications. For example, state-transition and reward distributions may shift from what was observed during training in the event of an anomalous situation, e.g., an emergency. In which case it could be disastrous were the agent to follow its policy decisions blindly. For that reason, consider the methods introduced in this chapter as an introduction to what is possible, but safety mechanisms would be put in place for a real-world RL+CPS application.

Within an MDP, agents may observe their current state and make actions which attempt to affect the future state. The agent's objective is to maximize collection of rewards. An example three-state MDP[1] is given in Fig. 3. In this example, the initial state is s_0, and the agent has two action options: a_0 and a_1. If the agent chooses action a_0, it is guaranteed to stay in state s_0, denoted by $p(s_0|s_0, a_0) = 1$. If the agent chooses action a_1, there is a 25% probability that it will transition to s_1, denoted

[1] In the notation for this example, the subscripts denote "options," versus the usual meaning, which is time in this chapter.

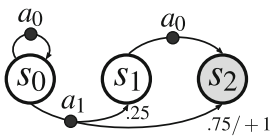


Fig. 3 Example Markov Decision Process. There are three states and two actions. Unless otherwise indicated, the state-transition probability is 1 and reward is 0. Transition from s_0 to s_2 is the most interesting with $r(s_2|s_0, a_1) = 1$ and $p(s_2|s_0, a_1) = 0.75$

by $p(s_1|s_0, a_1) = 0.25$, and a 75% probability it will transition to s_2, denoted by $p(s_2|s_0, a_1) = 0.75$. The environment returns reward of 0 for all state transitions except for $s_0 \rightarrow s_2$, and in this case it returns $r(s_2|s_0, a_1) = 1$.

The agent's only goal is to maximize collection of rewards. In the context of Fig. 3, the agent should always select action a_1, as it is the only action that leads to a non-zero reward. While *we* can see that is the solution, an agent must learn it. There are two general approaches for learning in RL: **value iteration** methods and **policy gradient** methods. Value iteration is the classical approach to RL and includes the Q-learning algorithm and its descendents. Policy gradient methods directly optimize a policy through gradient ascent. Policy gradient methods perform well in many situations, they are relatively straightforward to implement, and we focus on them in this chapter. More advanced methods combine value iteration and policy gradient methods.

2.2 *Reinforce Method*

In the context of reinforcement learning, our first-order objective was defined in Eq. (2) as the sum of rewards, but here we will refine it. As stated in the previous subsection, MDPs often have stochastic state-transition and reward functions; for that reason the **objective** $J(\theta)$ of the agent is actually to maximize the *expected* sum of rewards under the **trajectory probability distribution** (defined in Eq. (9)). This is achieved by discovering optimal policy parameters $\theta^\star$ for the objective function $J(\theta)$:

$$\theta^\star = \arg\max_\theta \mathbb{E}_{\tau \sim p_\theta} \sum_{t=0}^{T-1} r(s_t, a_t) = \arg\max_\theta J(\theta), \tag{4}$$

where τ is the **trajectory** of state-action pairs $(s_0, a_0, s_1, a_1, \ldots, s_T, a_T)$ and p_θ is the trajectory probability distribution.

The REINFORCE method uses gradient ascent to adjust the policy parameters in a direction which increases $J(\theta)$ [3]. For notation convenience let

$r(\tau) = \sum_{t=0}^{T-1} r(s_t, a_t)$, and by the definition of expectation, the objective can be written as:

$$J(\theta) = \mathbb{E}_{\tau \sim p_\theta}\, r(\tau) = \int p_\theta(\tau) r(\tau) d\tau, \tag{5}$$

where $p_\theta(\tau)$ is the probability of a specific trajectory. Taking the gradient of $J(\theta)$ with respect to θ then gives:

$$\nabla_\theta J(\theta) = \nabla_\theta \int p_\theta(\tau) r(\tau) d\tau = \int \nabla_\theta p_\theta(\tau) r(\tau) d\tau. \tag{6}$$

For reasons that will become clear, we recall the following identity:

$$\nabla_\theta p_\theta(\tau) = p_\theta(\tau) \frac{\nabla_\theta p_\theta(\tau)}{p_\theta(\tau)} = p_\theta(\tau) \nabla_\theta \log(p_\theta(\tau)), \tag{7}$$

allowing us to rewrite Eq. (6) as:

$$\begin{aligned} \nabla_\theta J(\theta) &= \int p_\theta(\tau) \nabla_\theta \log(p_\theta(\tau)) r(\tau) d\tau, \\ &= \mathbb{E}_{\tau \sim p_\theta}\, \nabla_\theta \log(p_\theta(\tau)) r(\tau). \end{aligned} \tag{8}$$

We now explain why the identity in Eq. (7) was used. The probability of a sampled (i.e., experienced) trajectory τ has a probability that can be explicitly calculated only if the underlying state-transition function is known:

$$p_\theta(\tau) = p(s_0) \prod_{t=0}^{T-1} \pi_\theta(a_t|s_t) p(s_{t+1}|s_t, a_t), \tag{9}$$

where $p(s_0)$ is the probability of starting the trajectory in state s_0 and is independent of θ and $\pi_\theta(a_t|s_t)$ is the probability of the selected action given the state observation s_t. To better understand the notation $\pi_\theta(a_t|s_t)$, note that the policy is **stochastic**. In other words, when the policy is given a state observation s_t, the output of $\pi_\theta(s_t)$ is a vector of probabilities derived from the $softmax$ function.[2] In the discrete action-space environments considered here, there is one output probability per possible action. A random action is then drawn from the given probability distribution, and the probability of the selected action is denoted $\pi_\theta(a_t|s_t)$.

[2] $softmax(x_i|\boldsymbol{x}) := \frac{e^{x_i}}{\sum_{j=1}^{|\boldsymbol{x}|} e^{x_j}}$, where $\boldsymbol{x}$ is a vector of reals.

In real-world problems, $p(s_{t+1}|s_t, a_t)$ is not known, so $p_\theta(\tau)$ would be impossible to calculate. However:

$$\begin{aligned}\log p_\theta(\tau) &= \log\left(p(s_0)\prod_{t=0}^{T-1}\pi_\theta(a_t|s_t)p(s_{t+1}|s_t,a_t)\right)\\ &= \log p(s_0) + \sum_{t=0}^{T-1}\log\pi_\theta(a_t|s_t) + \log p(s_{t+1}|s_t,a_t),\end{aligned} \tag{10}$$

and replacing $\log p_\theta(\tau)$ in Eq. (8) with its expanded form gives:

$$\begin{aligned}\nabla_\theta J(\theta) &= \mathbb{E}_{\tau\sim p_\theta}\nabla_\theta\left[\log p(s_0) + \sum_{t=0}^{T-1}\log\pi_\theta(a_t|s_t) + \log p(s_{t+1}|s_t,a_t)\right]r(\tau),\\ &= \mathbb{E}_{\tau\sim p_\theta}\sum_{t=0}^{T-1}\nabla_\theta\log\pi_\theta(a_t|s_t)r(\tau).\end{aligned} \tag{11}$$

In this form, we are able to approximate the gradient. Recall that π_θ is a neural network (or some other differentiable function), so the gradient of its log may be calculated explicitly given each a_t and s_t over the trajectory. Also, we know the sum of rewards $r(\tau)$ for each trajectory. Finally, the outer expectation is approximated by performing N **episodes**, i.e., experiencing multiple trajectories, and then averaging the sums giving:

$$\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{n=1}^{N}\sum_{t=0}^{T-1}\nabla_\theta\log\pi_\theta(a_{n,t}|s_{n,t})r(\tau_n). \tag{12}$$

After having obtained an approximation of the objective's gradient, we may use it to update the neural network parameters with standard stochastic gradient ascent:

$$\theta = \theta + \alpha\nabla_\theta J(\theta), \tag{13}$$

where α is the learning rate and whose appropriate value must be experimentally found.

The REINFORCE method works surprisingly well for a broad range of problems, and there are many improvements that have been made to it to increase its performance. Understanding the method presented here is a good foundation for approaching current literature. The source code we provide for this chapter at https://github.com/RodgerLuo/CPS-Book-Chapter uses REINFORCE.

3 Microsoft's AirSim

3.1 Overview

Developed by Microsoft Research AI in 2017, AirSim is an Unreal Engine plug-in to provide physically realistic simulations for autonomous vehicles in Unreal Engine environments [4]. The goal of AirSim is to offer an open-source platform for artificial intelligence research, especially in developing and comparing reinforcement learning algorithms.

The installation of AirSim is straightforward, and its official web page provides explicit instructions: https://github.com/Microsoft/AirSim/blob/master/docs/build_windows.md. AirSim delivers binaries and builds for Windows and for Linux. We recommend installing AirSim using builds because it gives more freedom for customizing environments in Unreal. For this chapter, we built AirSim on Windows 10 Pro version 1709. For the rest of the section, we will introduce the core features of AirSim concerning reinforcement learning and its Python APIs. We will discuss how to customize Unreal environments in the following section.

By default, AirSim has two built-in vehicle models: `multirotor` and `car`. For the rest of the chapter, we will use "drone" to indicate `multirotor` mode. Users can choose to either manually fly the drone or drive the car with a remote control or programmatically control the vehicles in C++, C#, Python, Java, etc. This section focuses on programmatic control, particularly using APIs in Python. Readers can find the official document for remote control configuration on this web page: https://github.com/Microsoft/AirSim/blob/master/docs/remote_control.md. We chose AirSim for our studies mainly because of two features: (1) the combination of AirSim and Unreal provides a more realistic training environment than other existing RL environments such as OpenAI Gym or Unity ML library, and (2) the support of software-in-loop and hardware-in-loop with popular flight controllers provides a potential smooth transition from the simulator to the real world. This chapter focuses on drone simulation in which AirSim provides four different flight modes, discussed in Table 1.

To choose a simulation mode and configure other settings, users can edit `setting.json` at `Documents\AirSim` on Windows or ~/Documents/AirSim on Linux. The minimal configuration is as follows:

```
{
 'SettingsVersion':1.0
}
```

Listing 1 The minimal version of AirSim configuration

`SettingsVersion` instructs AirSim to load default settings when the Unreal Engine starts. It is the only item required in `setting.json` and is usually listed as the first item in the file. Any items after the `SettingsVersion` override the related parameters in the default setting.

Table 1 Drone simulation modes in AirSim

Simulation mode	Description
Computer vision	No vehicle physics and dynamics involved in this mode. Users can use keyboards or APIs to navigate and position the virtual drone. This mode is usually used for proof of concept and rapid prototyping
Simple flight	A built-in flight controller provides realistic drone-flying experience in Unreal. It doesn't require additional configurations
Hardware-in-loop[a]	The flight controller runs on physical hardware, which communicates with AirSim using the USB port. The mode is the closest scenario in comparison to flying a real drone, but requires additional setups and is usually hard to debug
Software-in-loop[a]	This mode is similar to hardware-in-loop, except the firmware runs on the computer as opposed to a separate board. Regarding the relationship to flying the drone in the real world, this mode is in-between the simple flight and hardware-in-loop

[a]Hardware-in-loop and software-in-loop are usually for advanced users. By far, AirSim supports PX4 flight controller and plans to support ROSFlight and Hackflight in the future

Listing 2 shows how to load the virtual drone and use the computer vision mode in the environment. `SimMode` determines whether to load the drone or car by setting the parameter to `multirotor` or `car`, respectively. Setting `UsageScenario` to `ComputerVision` disables physical simulation, so the drone would hang in the air. Readers can find a comprehensive description about the settings of AirSim from this web page: https://github.com/Microsoft/AirSim/blob/master/docs/settings.md.

```
{
 'SettingsVersion': 1.0,
 'SimMode': 'Multirotor',
 'UsageScenario': 'ComputerVision'
}
```

Listing 2 A configuration of AirSim to load the drone model and activate the computer vision mode

3.2 Python APIs

In this section, we introduce two major features of the AirSim Python APIs: drone navigation and image processing.

In the APIs for drone navigation, except for computer vision mode, all simulators need to obey the rules of flying a real drone. Namely, the drone needs to be `armed` before taking off and `disarmed` after landing. All other navigation commands should wait until the drone takes off and hovers at a stable height. Table 2 shows five auto-flight modes in the Python APIs that facilitate the drone navigation process. In

Table 2 Five auto-flight modes in AirSim Python APIs

Auto-flight command	Description
`armDisarm`	Mandatory before taking off or after landing the drone
`takeoff`	Take off and ascend to default height
`land`	Land drone at its current position
`goHome`	Move drone to its take-off location, followed by `land` command
`hover`	Hover the drone at its current position

the computer vision mode, however, the drone is spawned directly in the air with no physics and dynamics and doesn't need to obey any aforementioned rules. We can consider the drone as a non-gravity block with visual inputs in the computer vision mode.

Before we discuss more sophisticated APIs to navigate the drone, it is necessary to introduce four aircraft terms: pitch, roll, yaw, and throttle. Unlike driving a car, controlling a drone is performed by making it rotate in three axes: **normal axis**, **lateral axis**, and **longitudinal axis**. Figure 4 illustrates these axes.

The normal axis, also known as the vertical axis, is perpendicular to the body of the drone and is directed toward the bottom. **Yaw** is the motion for this axis. Positive yaw moves the head of the drone to the right.

The lateral axis is directed to the right of the drone and parallel to an invisible line drawn from the left edge to the right edge. **Pitch** is the motion for this axis. Positive pitch raises the head of the drone and lowers the end.

The longitudinal axis is directed forward, parallel to the body of the drone. **Roll** is the motion for this axis. Positive roll lifts the left side of the drone and lowers the right ride.

Besides the three motions, throttle controls the vertical movements of the drone. Positive throttle raises the drone vertically.

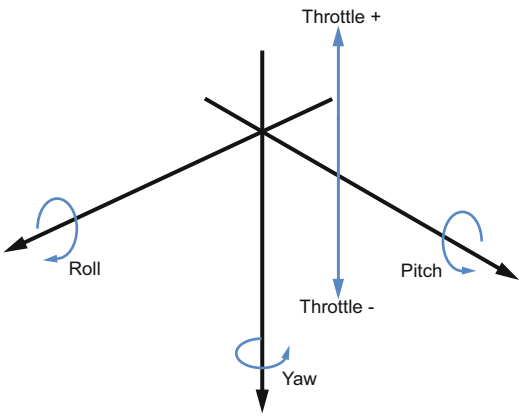


Fig. 4 An illustration of the four drone axes: roll, yaw, pitch, and throttle

Table 3 Five common navigation methods in AirSim

Methods	Parameters
`moveByAngle`	`pitch, roll, z, yaw, duration`[a]
`moveByVelocity`	`vx, vy, vz, duration, drivetrain`[b], `yaw_mode`[c]
`moveByPath`	`path, velocity, max_wait_seconds`[d], `drivetrain, yaw_mode, lookahead`[e], `adaptive_lookahead`[e]
`moveToPosition`	`x, y, z, velocity, max_wait_seconds, drivetrain, yaw_mode, lookahead, adaptive_lookahead`
`moveByManual`	`vx_max, vy_max, z_min, duration, drivetrain, yaw_mode`

[a]Methods taking the `duration` parameter are usually followed by a `time.sleep` function, because the methods release control immediately. Without the `time.sleep` function, the methods would not have enough time to finish

[b]`Drivetrain` has two modes: `ForwardOnly` and `MaxDegreeOfFreedom`. `ForwardOnly` keeps the drone's front always pointing in the direction of travel, while `MaxDegreeOfFreedom` doesn't have such restriction

[c]`Yaw_mode` has two fields: `is_rate` and `yaw_or_rate`. Usually, it is set to `yaw_mode` (false, 0) to keep the yaw constant

[d]In comparison to `duration`, `max_wait_seconds` blocks the amount of time, in order to make sure the action completes

[e]Most of the time, we set `lookahead` $= -1$ and `adaptive_lookahead` $= 0$ to let the drone auto-decide the path

AirSim Python APIs provide five commands to navigate the drone by taking different physical parameters as inputs. Some commands also have simpler versions that take less parameters. Table 3 lists the five commands and the parameters each command takes.

A necessary component for vision-based deep reinforcement learning is acquiring visual inputs from the drone. AirSim provides seven image types: `scene`, `depth planner`, `depth perspective`, `depth vis`, `disparity normalized`, `segmentation`, and `surface normals`. We used the `scene type` to get images for our studies. The following line of code demonstrates how to acquire raw image data from the AirSim Unreal environment:

```
rawData = self.client.simGetImages([ImageRequest(1,
    AirSimImageType.Scene, False, False)])
```

Listing 3 A command using AirSim API to acquire image data. The command takes four variables: `camera_id`, activating a particular camera on the drone; `image_type`, choosing one of the seven image types mentioned above; `pixels_as_float`, determining whether the data type of pixels is float or integer; and `compress`, determining whether the acquired image is compressed or not. The above command chooses the front-center camera, uses `scene` image type, encodes the pixel values as integers, and keeps the image uncompressed

AirSim embedded five cameras on the drone: three of them are in the front, one is in the back, and one is on the bottom of the drone. The three front cameras are located on the right, center, and left, respectively. Camera No.1 refers to the front-center camera, which is the camera we used for our reinforcement learning tasks.

The orientations and positions of the cameras are customizable in Unreal. Besides APIs for drone navigation and image processing, other useful APIs consist of `Collision`, `Reset`, `SimGetObjectPos`, and `ApiControlEnabled`. More details about the AirSim Python APIs can be accessed from: (1) https://github.com/Microsoft/AirSim/blob/master/docs/apis.md and (2) https://github.com/Microsoft/AirSim/blob/master/PythonClient/AirSimClient.py

4 Reinforcement Learning in AirSim

In this section we give a detailed introduction of how to implement RL using the AirSim extensions detailed in Sect. 3. Our task is to train the drone to automatically "collect" cubes in the environment, which consists of (1) an Unreal level for dynamically positioning the cubes and monitoring the interactions between the drone and the cubes and (2) a Python library used as an intermediate to receive and send data between the Unreal level and RL algorithms. In terms of the navigation, we simplify the drone's movements by constraining it to three actions: left action, right action, and forward action. Figure 5 demonstrates the drone's movements used in the cube collection task.

4.1 Unreal Dynamic Environment

Our Unreal environment was designed specifically for the cube collection task. The environment fulfills the following requirements: (1) for each episode in the RL training, the cubes should be randomly positioned in a restricted area; (2) the cubes should vanish when the drone hits them or bypasses them. We don't want the drone to accidentally collide with the cubes by moving to the left or right when no cube is visible in the drone's camera view, because it will result in

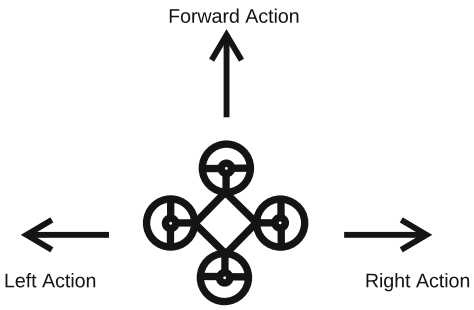


Fig. 5 Drone's movements were simplified to fulfill the requirement for our cube collection task

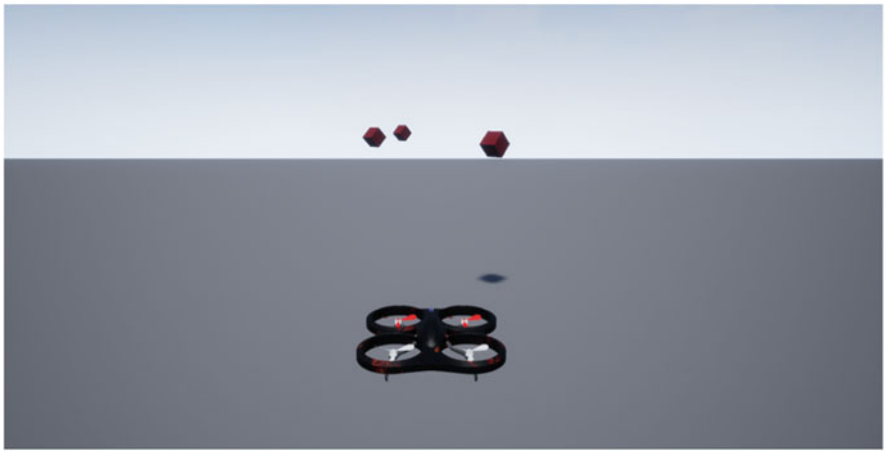


Fig. 6 A screenshot from Unreal, immediately after the drone spawned in the environment

false rewards[3]; (3) the environment should have invisible boundaries on the front, left, and right to constrain the drone's movements; (4) the number of the cubes should be dynamic, and the invisible boundaries should be automatically adjusted according to the number of the cubes. We adapted the Block Unreal environment created by AirSim (https://github.com/Microsoft/AirSim/blob/master/docs/unreal_blocks.md) and removed the unnecessary environmental details to create a minimal Unreal environment for our task. Users can also use AirSim in any other Unreal environments. A documentation of how to install AirSim in Unreal environments is here: https://github.com/Microsoft/AirSim/blob/master/docs/unreal_custenv.md.

Generally speaking, Unreal has two modes: (1) `edit mode`, for editing the environment, and (2) `play mode`, for testing the environment. The drone only appears in `play mode` and spawns at the origin in the environment. To change the spawning position of the drone, users need to modify the position of `PlayerStarter`, an object corresponding to the drone's position. In `play mode`, users can see different camera views by pressing `1`, `2`, or `3`. Pressing `F1` can activate a detailed help menu. Figure 6 shows an initial state after spawning the drone in the environment.

The requirements mentioned above were implemented in Unreal's own visual programming language, called **blueprints**. We created two blueprints to manage the drone-cube interactions in the environment. The first blueprint is applied to all the cubes so that each cube would be removed immediately after the drone hits or passes it. The second blueprint is to randomly spawn all cubes in a constrained space, once the drone is reset to its origin. Given the visual complexity of the two blueprints, we present the high-level abstract structure of the two blueprints in Algorithm 1 and

[3]This is because our RL policy is memoryless. If an RNN or LSTM were used, instead of a vanilla CNN, the policy gains memory, and it would be possible for the drone to learn to bump into cubes which it can no longer see.

Algorithm 1: Apply to all the cubes in the environment that each cube would be removed immediately after the drone hits or passes it

```
while env is running do
    if collision then
        remove the cube
    else if drone.xPosition > cube.xPosition then
        remove the cube
end
```

Algorithm 2: Randomly reposition all the cubes in a constrained space, once the drone is reset to its origin

```
while env is running do
    if drone.position == origin.position then
        remove_all_cubes()
        for cube_index in range(num_cubes) do
            spawn.xPos = cube_index * cube_distance + starting_pos
            spawn.yPos, spawn.zPos = random(-bound, bound)
            spawn_cube(spawn.position)
        end
    end
end
```

Algorithm 2, respectively. Readers can download the blueprint files from our source code repository.

Usually, blueprints and models are located in `Asset` folder in Unreal. However, since the drone is imported from AirSim rather than created in Unreal, it doesn't exist in the `Asset` folder. To modify the drone's blueprint, we need to switch to `play` mode in which an object called `BP_FlyingPawn` highlighted in gold will appear in the `World Outliner` window. The blueprint exists at the `Event Graph` menu after clicking `Edit` → `BP_FlyingPawn`.

We also expanded the field of view of the original camera on the drone to better keep cubes in the drone's view. To modify the cameras' attributes, we open the editor of `BP_PIPCCamera`, by using the same method of accessing the drone's blueprint. The field of view can be found on the `Details` window, under the `Projection` menu. We changed the field of view from 90.0 to 135.0.

4.2 Python Environment Library

Our Python environment library serves as an intermediate to send and receive data between the Unreal environment and our reinforcement learning algorithms. It is designed explicitly for the AimSim Python APIs and is compatible with any automatic differentiation framework, such as TensorFlow or PyTorch.

Table 4 A high-level description of the Python environment library

Function	Description
`__init__`	Setup connections to the AirSim Python APIs. Parse the parameters of the drone and the environment
`reset`	Position the drone to the origin. Reset parameters
`step`	Send the updated drone's position to AirSim. Return state, reward, done, and info[a]
`get_image`	Convert and process raw image data from AirSim to NumPy arrays which are compatible with our convolutional neural network

[a]The details of the four parameters will be introduced in the following paragraph

The library has four major functions: (1) initialize the environment; (2) reset the environment; (3) for each step, navigate the drone based on the received action and return the essential training data, such as state, reward, and done; and (4) process raw image data from Unreal to make them compatible with the deep neural network. Each function can be called independently during the training process. Table 4 shows a high-level description of the library.

The core part in the library is the `step` function of which the logic is presented in Algorithm 3. The function is responsible for returning four essential values for our RL training:

Algorithm 3: *step* function, a core function in our Python Environment Library that takes action as the input and returns state, reward, done, and number of collected cubes

Input: action
Output: state, reward, done, number of collected cubes

```
the drone's current position = GetPosition();
distance = Move(action);
SetPosition(the drone's current position + distance);
state = GetImage();
collision info = GetCollisionInfo();
if collided then
    reward = 1;
    num of collected cubes += 1;
end
if the drone out of the boundary then
    done = 1
else if num of collected cubes == num of cubes placed in the env then
    done = 1
else if num of steps == threshold then
    done = 1
else
    done = 0
end
```

- `state`: represents the drone's observation of the environment. In our case, the states are the images captured by the camera on the drone.
- `reward`: the amount of rewards acquired by the previous action. We set the reward to 1 for collecting a cube and 0 for any other actions.
- `done`: whether the episode ends. In our case, one episode ends when the drone collects all the cubes in the environment or moves outside of the defined boundaries.
- `info`: diagnostic information useful for debugging. We are particularly interested in the number of cubes collected by the drone.

4.3 REINFORCE Method in AirSim

We now adapt the concepts of the REINFORCE method from Sect. 2.2 to the cube collection task. A simple convolutional neural network is used to represent the policy. We will refine the objective (from Eq. (2)) of finding network parameters which maximize the expected sum of rewards across all time steps in the episode:

$$\theta^{\star} = \arg\max_{\theta} \mathbb{E}_{\tau \sim p_{\theta}} \sum_{t=0}^{T-1} r(s_t, a_t).$$

In the cube collection task, a reward of 1 is provided by the environment each time a cube is collected by the drone, and 0 reward when no cube is collected, so the objective is to collect as many cubes as possible in each episode (i.e., during time step $t = 0 \ldots T-1$, where $T-1$ is the step when the last cube is collected or the drone has gone out of bounds). In the context of REINFORCE, this objective was discovered by taking its gradient (from Eq. (12)):

$$\nabla_{\theta} J(\theta) \approx \frac{1}{N} \sum_{n=1}^{N} \sum_{t=0}^{T-1} \nabla_{\theta} \log \pi_{\theta}(a_{n,t}|s_{n,t}) r(\tau_n),$$

and then updating the neural network parameters based on the gradient. Recall that π_{θ} is the neural network, and, in the drone collection tasks, $\pi_{\theta}(a_{n,t}|s_{n,t})$ is the output probability[4] of going left, right, or forward, given input pixels from the drone's camera.

The weakness of Eq. (12) for our context is that the rewards are sparse, because there are only three cubes total to collect in each trajectory. If $r(\tau_n)$ is used directly as the reinforcing signal, then *entire* trajectory probabilities are increased or are unchanged. This results in high variance in performance between each episode. An approach to get faster results in the cube collection task is to "smooth" the attribution

[4] Softmax of the network's logits.

Algorithm 4: REINFORCE algorithm in the context of the AirSim cube collection task

Input: Old policy parameters θ, learning rate α, tuple of (observations s, actions a, rewards r) from last cube collection episode
Output: Updated policy θ

Apply Eq. (14) to obtain discounted rewards g from a
Normalize g
Set $sumofgrads = 0$
for $t = 0 \ldots T-1$ **do**
  $sumofgrads = sumofgrads + g_t \nabla_\theta \log \pi_\theta(a_t|s_t)$
end
$\theta = \theta + \alpha sumofgrads$
return θ

of rewards from later stages to earlier stages by applying a **discounted return** to the gradient at each time step. Discounted return is defined as:

$$g_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \gamma^3 r_{t+3} + \gamma^{t+T} r_{t+T} = \sum_{k=0}^{T-1} \gamma^k r_{t+k+1}, \tag{14}$$

where $\gamma \in [0, 1]$ is the discount rate. The resulting $\boldsymbol{g}$ vector of Eq. (14) is also normalized[5] in the cube collection task. Using $\boldsymbol{g}$ we update Eq. (12) to give:

$$\nabla_\theta J(\theta) \approx \sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t|s_t) g_t, \tag{15}$$

where we are only collecting a single trajectory between applications of gradient ascent. There are better approaches than using the discounted return, and our source code example is parameterized to allow experimentation between other reward function alternatives.

We summarize the use of the REINFORCE method in the context of our AirSim cube collection task formally in Algorithm 4. This algorithm is implemented in the provided source code.

5 Increased Trustworthiness Through Visualization

Machine vision algorithms have changed radically in the era of artificial intelligence. For example, prior to the age of AI, if we asked a machine to look for a face in an image, we knew what the machine would look at. With the development of deep

[5]Normalization is defined as $\boldsymbol{g} \leftarrow (\boldsymbol{g} - \mu(\boldsymbol{g}))/\sigma(\boldsymbol{g})$, where scalar operations are applied element-wise to the vector.

neural networks, we significantly increased machine vision performance, surpassing that of traditional methods, but we initially had limited understanding regarding which part of the input triggered the machine to come to its conclusion. Such "black-box" performance is tolerated for some applications, but, in order to develop trust in AI's decision-making process, it is important for engineers to understand the mechanism behind the machine's decision-making.

This section will introduce three common visualization techniques: T-SNE, action visualization, and attribution visualization. We adapt these visualization methods for applications of understanding deep reinforcement learning. As a case study, we use imagery and trained policies from the cube collecting example which was detailed in previous sections. For each visualization technique, we analyze three policies that have been trained to collect cubes using the REINFORCE algorithm. The policies have the following descriptions:

- **Good policy**: collects most cubes in the environment.
- **Poor policy**: is unable to collect any cubes except randomly
- **Right-and-forward policy**: is only able to collect cubes directly in front or to the right. Ignores all cubes on the left

5.1 t-SNE

t-Distributed Stochastic Neighbor Embedding (t-SNE) is a dimensionality reduction algorithm developed by Laurens van der Maaten and Geoffrey Hinton [5]. It is well suited for visualizing high-dimensional datasets in 2D or 3D spaces. Specifically, the visualization maps each high-dimensional data point to a two- or three-dimensional space in a way that similar data points are nearby and dissimilar ones are distant. The technique was adapted later to reveal structure of images at many different scales. The most recent application of visualizing images via t-SNE is to use a convolutional neural network to extract features from images, input the extracted features of each image to t-SNE to get the "position" of each image, and then arrange the images on a 2D or 3D space based on the given positions.

Formally, t-SNE measures the pair-wise distance between all points in a high-dimensional dataset. It then projects the high-dimensional dataset to a 2D or 3D space and adjusts the points in the projection to have pair-wise distances which are similar to the high-dimensional dataset. The pair-wise distance between points x_i and x_j in dataset $\boldsymbol{x}$ is defined as the probability that x_j would be chosen from a Gaussian centered at x_i:

$$p_{j|i} = \frac{\mathrm{e}^{(-\|x_i - x_j\|^2/2\sigma_i^2)}}{\sum_{k \neq i} \mathrm{e}^{(-\|x_i - x_k\|^2/2\sigma_i^2)}}, \tag{16}$$

where σ_i is the variance of a Gaussian centered on data point x_i. Using a similar metric, the distances between low-dimensional points y_i and y_j are measured as:

$$q_{j|i} = \frac{e^{(-\|y_i - y_j\|^2)}}{\sum_{k \neq i} e^{(-\|y_i - y_k\|^2)}}. \tag{17}$$

For each low-dimension point y_i, the similarity of the high-dimensional and low-dimensional dataset is measured by taking the sum of the Kullback-Leibler divergences between $p_{*|i}$ and $q_{*|i}$. A cost function is defined by this sum, and then the projection of point y_i is adjusted via gradient descent. By iteratively adjusting each low-dimension point in this way, the low-dimension dataset takes on distance characteristics of the high-dimensional dataset.

In this subsection, we use t-SNE to examine the representations learned from the cube collection task to have a visual overview of the three trained policies. To distinguish the three different actions (forward, right, and left) triggered by the visual inputs, we tinted each visual input based on its predicted action: red indicates forward, green indicates left, and blue indicates right.

Figure 7 shows the two-dimensional t-SNE embedding of the visual inputs when using the poor policy (left) and the right-and-forward policy (right). The t-SNE visualization provides insight into the policy's quality. In the poor policy, e.g., the drone only moved to the left from the beginning to the end of each episode, regardless of the visual inputs. In the right-and-forward policy, the majority of the

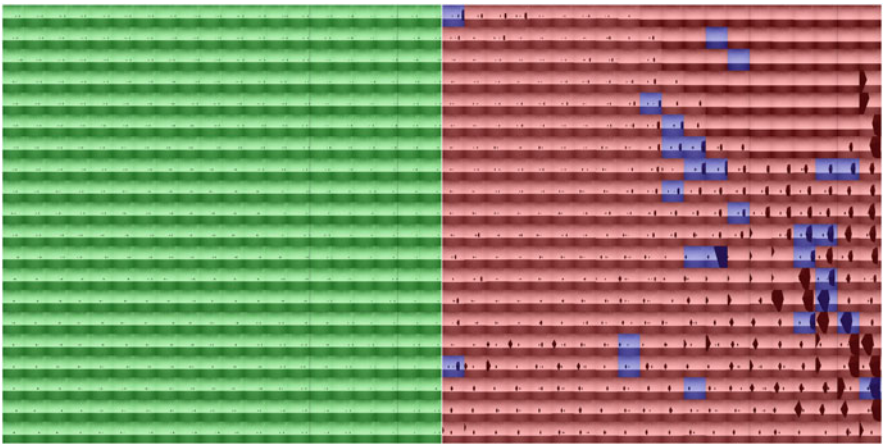


Fig. 7 Two-dimensional t-SNE embedding of visual inputs obtained by applying the poor policy (left) and right-and-forward policy (right) to the cube collection task. The left shows 400 inputs from 15 episodes from the poor policy; the right presents 400 inputs from 8 episodes from the right-and-forward policy. Tinted colors indicate the predicted actions: red, forward; green, left; and blue, right. It can be seen that the poor policy always chooses the left action (green), no matter the input. The right-and-forward policy usually chooses forward (red) and will occasionally go right (blue)

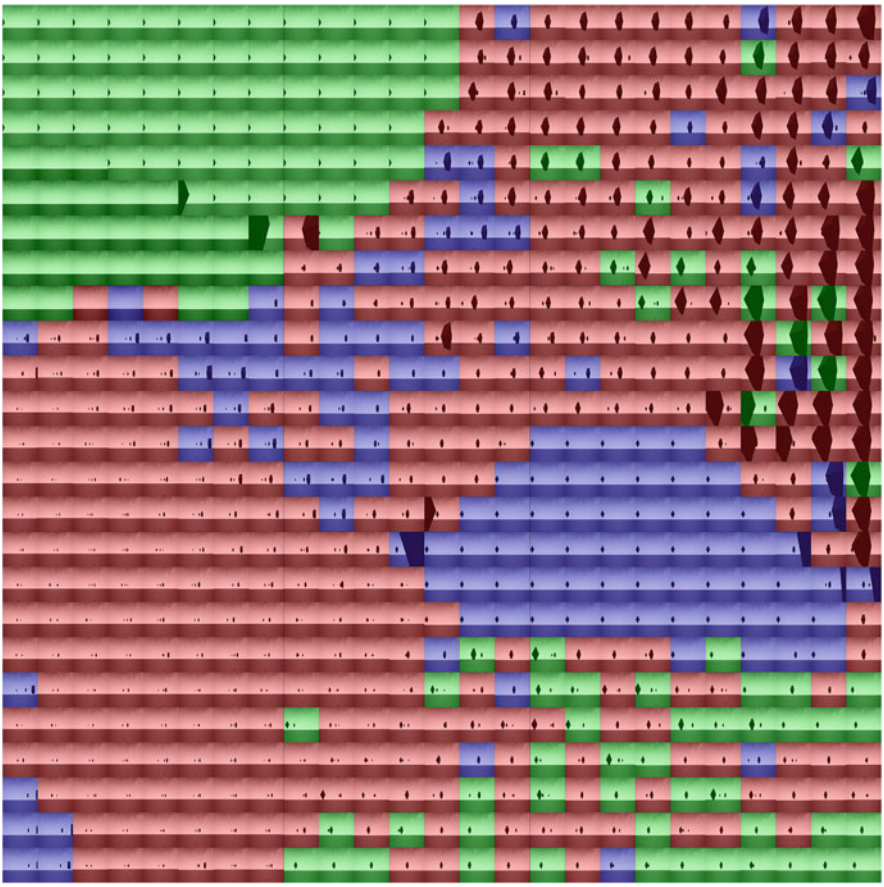


Fig. 8 Two-dimensional t-SNE embedding of 625 visual inputs from 7 episodes of applying the good policy to the cube collection task. Tinted colors indicate the predicted actions: red, forward; green, left; and blue, right. The large red cluster makes sense, because the policy should choose forward in those images. Furthermore, most of the scattered blue and green make sense. But the larger blue cluster highlights a weakness in the so-called good policy, as the policy should go left in such settings, but instead chooses right

moves are forward, and occasionally the policy chose to move to the right. As t-SNE arranges images based on their visual similarity, the scattered blue images indicate that something is wrong with the policy. Images that trigger the same behavior should logically be placed together by t-SNE.

By observing the t-SNE visualization of the so-called good policy in Fig. 8, we see that cubes in the center of the image are likely to trigger the forward action and cubes in the left or right of the images are likely to trigger the left or the right action, respectively. By examining Fig. 8, we also see that t-SNE may be used as an efficient tool to detect flaws in RL policies, even if the policies have good performance, like

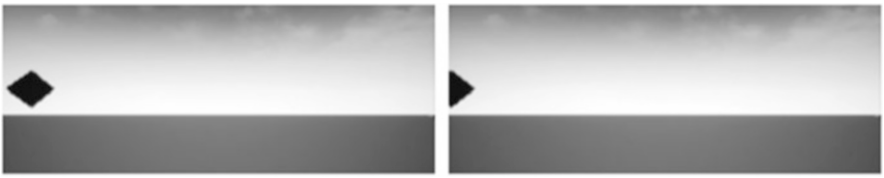


Fig. 9 These two visually similar images triggered opposite actions (left and right) in the so-called good policy. This indicates a flaw in the trained policy. t-SNE enables the RL policy developer to quickly spot such anomalies

this one. For example, in Fig. 8 the green area on the upper left and the blue area on the lower right both contain cubes located on the left edge of the image; however it can be seen that those images triggered two opposite actions, which is against the intuition of a good policy for the cube collection task. After examining the raw data, we found the drone stuck in such situations by moving back and forth between left and right. Figure 9 highlights the two individual visual inputs causing the opposite actions.

5.2 Action Visualization

Action visualization methods generate visual inputs which would activate a particular action in a *trained* policy network. This approach allows for a high level of human comprehension about the behavior of a network, rather than treating the network like a black box model. For our specific action visualization approach, we use the **Class Model Visualization** (CMV) technique introduced in [6]. CMV generates inputs which will trigger any specific output class in a trained convolutional neural network.

In this subsection we are interested in understanding *what* visual input causes specific actions to be selected by the RL policy of interest. In our study, we generate inputs to optimize for the three action cases: left, right, and forward. After choosing an action to optimize for, a uniformly random image is generated.[6] This initially random image is then evaluated by the trained policy network. The output probability for the desired action is then increased through back-propagation, where the gradient is calculated with respect to the image pixels. We repeat this process of forward and back-propagation, until the action probability is maximized.

Formally, we let a represent the action for which we want to generate an input image to trigger; s is the input image which will be optimized such that the action probability is maximized. We let $\pi_\theta(a|s)$ represent the probability of taking action

[6]For the cube collection task used in this chapter, we used a simple CNN with a grayscale image as input, so we generate grayscale images for action visualization.

Fig. 10 Action visualizations for the good policy. The drone learns that when the camera is occluded (dark spots), it should move in the direction of the occlusion. (**a**) Left action. (**b**) Forward action. (**c**) Right action

$a \in \{\text{left, right, forward}\}$, given the image s. The goal then is to solve the following optimization problem:

$$s^* = \arg\max_s \pi(a|s). \tag{18}$$

In practice, the optimal image s^* is found using gradient ascent by an automatic differentiation tool like TensorFlow or PyTorch.

Note that CMV differs from the normal application of back-propagation, which typically considers the input (s in this case) to be fixed and instead finds neural network parameters θ which optimize the objective function. In the case of CMV, parameters are locked after policy training, and it is instead s which is optimized.

Generating action visualizations is currently an art, and, unfortunately, if the basic objective derived from Eq. (18) is used, an unsatisfactory optimal image s^* may be generated. However, there are a number of refinements that generate more meaningful images using CMV. If images are unsatisfactory, one refinement is to preprocess the current input image s by subtracting the mean and standard deviation of the training set images from s between each iteration.[7] Another refinement is to blur s between iterations. Another refinement is to use large learning rates during optimization, e.g., 20. See [7] and [6] for more optimizations.

Using CMV, we generated action visualizations shown in Figs. 10, 11, and 12. These visualizations illustrate the inputs which would maximize probabilities for selecting the three possible drone actions by the good policy, poor policy, and the right-and-forward policy. If we choose a single random image s and iterate on it for an extended period of time using the methods described above, we will obtain an input image which will trigger the desired action in the policy of interest.

The good policy's action visualization in Fig. 10 clearly explains what the network is looking for. The bias toward the left, forward, or right is apparent in each image, depending on the position of the cube. Specifically, if a cube blocks the view of the camera on the left, then the policy will most likely choose the left action. Similarly for the forward and right actions, note that horizon is somewhat visible (near the center) in the left and forward action visualizations. It is also interesting to

[7] In this case, the training set consists of the set of images captured by the drone during its episodes.

Fig. 11 Action visualizations for the poor policy. The forward action visualization shows that only the forward action is most probable. All actions have roughly equivalent action visualizations, indicated by the "murky" figures. (**a**) Left action. (**b**) Forward action. (**c**) Right action

(a) Left action (b) Forward action (c) Right action

Fig. 12 Action visualizations for the right-and-forward policy. In (**a**) we observe that only an empty field of view (which should never happen in our simulation) would cause the drone to move left. In (**b**) the forward action visualization is also incorrect (see Fig. 10 for a correct version) and objects on the left will trigger a forward action. In (**c**) the right action has the expected action visualization

note that the images are not quite symmetric (the left action visualization does not look like the right action visualization).

Action visualizations of the poor policy help explain why the drone performs so poorly. Figure 11a–c illustrates that the actions of the poor policy are equally triggered by noise. The policy used to generate the images in Fig. 11 was unable to collect cubes in the simulation.

Action visualizations for the right-and-forward policy are given in Fig. 12. At a high level, we can see that only the right action visualization makes sense. That is, when the camera is occluded on the right, the policy will choose to move to the right. The left action visualization shows that there is a small response to camera occlusion on the left, but the forward action visualization dominates the left action. These visualizations explain why the policy collects 100% of the cubes that occur on the right side of the drone, but no cubes on the left side.

The degenerate right-and-forward policy is especially insightful and highlights one of the challenges in reinforcement learning. If the learning rate (α in Eq. (13)) is too high, the policy might finalize its decision-making process based on early experience. In this case, the drone experienced an early success by moving right and forward almost exclusively during early experiences. Combined with a learning rate that was too high, this early success led to a catastrophic elimination of left action probabilities. The remedy for this was to lower the learning rate of the policy updates and retrain the policy.

The ability to visualize and understand the desired inputs for any individual policy action is a useful tool for verification and debugging of policies. The downside to the Class Model Visualization technique presented here is the number of hyperparameters which must be manually tuned.

We now move to a technique which is able to highlight areas of an experienced input image responsible for triggering specific actions.

5.3 Attribution Visualization

Attribution visualization techniques highlight regions in an input which were most responsible for a particular action in a CNN-based policy. In this subsection we use an attribution visualization technique called **Gradient-Weighted Class Activation Mapping** (Grad-CAM) [8].

CNNs are particularly well suited for attribution visualization, because they maintain spatial structure of the input as it flows through the network; this is why we can extract meaning from the last layer. A **feature map** is the output of a convolutional layer after it has passed through a nonlinearity function (e.g., ReLU[8]). Feature maps typically have many channels, and the goal of Grad-CAM is to find which channels contribute the most to an action taken. Grad-CAM achieves that goal by calculating the average derivative of the policy network, given a specific action a and input image s, with respect to the feature map of interest[9]:

$$\alpha_k = \frac{1}{Z} \sum_i \sum_j \frac{\partial \pi_\theta(a|s)}{\partial A_{ij}^k}, \tag{19}$$

where A is the feature map of our target convolutional layer, A^k is the channel k of A, A_{ij}^k is the neuron at position i, j, and $Z = i \times j$. α_k is known as the **importance weight** for feature map channel k.

To help clarify Eq. (19), consider that each partial derivative $\partial \pi_\theta(a|s)/\partial A_{ij}^k$ gives the change in the probability of the desired action, with respect to activation i, j. Summing over all i, j in a channel gives the total change in the probability with respect to all of the channel activations. Finally, α_k is the average derivative of the feature map channel. Feature maps typically have many channels, denoted by K, and a unique α_k is calculated for each one.

Once importance weights $\alpha_1, \alpha_2, \ldots, \alpha_K$ are known, they may be used to linearly combine feature map channels 1 through K, giving a "class activation map":

$$\text{Grad} - \text{CAM} = \text{ReLU}\Big(\sum_k \alpha_k A^k\Big), \tag{20}$$

where ReLU is being used to filter for derivatives with a positive effect.

[8] ReLU stands for rectified linear unit and is defined as $\text{ReLU}(x) = \max(0, x)$.

[9] When using Grad-CAM, a and s are sampled from the policy and environment, while the policy controlled the drone, whereas with CMV (presented in the previous subsection) s was generated by the method and a was specified.

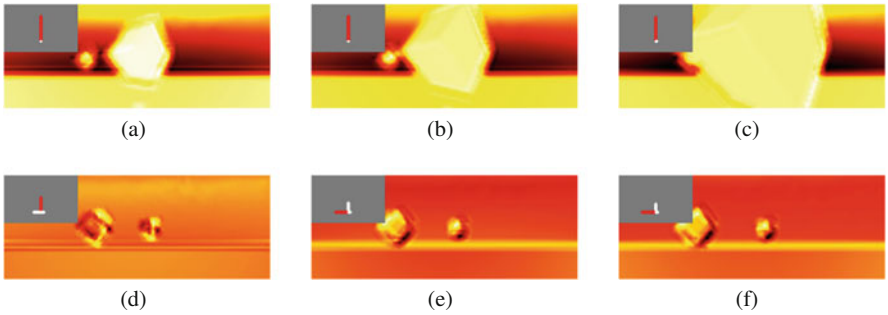


Fig. 13 Attribution visualizations for a sequence of observation-action pairs from the good policy. The ⊥ shape in the gray region indicates action probabilities (by the lengths) and the action that was chosen (colored red). Brighter areas indicate regions most responsible for actions

In the remainder of this section, we use Grad-CAM to create attribution visualizations for the left, forward, and right actions. In addition, the action visualization created the good policy, poor policy, and right-and-forward policy. In each example, we have six images which were captured while the drone performed in the simulator. Each visualization shown below also has an indicator which shows three bars: one for left, forward, and right, where the length of each bar gives the action probability output from the policy. The action indicated by the red bar is the action for which attribution visualization is generated in each image.

In Fig. 13, we first consider attribution visualizations for the good policy. First consider Fig. 13a. In this image, we see that the left action has the highest probability. But because the forward action was chosen, it is colored red. Grad-CAM was then used to visualize exactly what in the input image cased the forward action to have the probability that it had. The bright spots in the image give that information. We can see that the cube is very bright, which indicates that it had a high influence on choosing the forward movement. Figure 13f is also worth considering. In this case the forward action was the only action with a high probability, and the entire cube is bright, indicating its responsibility for the action. Also note in Fig. 13f how the ground is bright, which indicates that our policy has learned not just to look for cubes, but also pays attention to other aspects of the environment.

In Fig. 14 we see visualizations for the poor policy. In that figure, observe that all action probabilities are roughly the same, as indicated by the ⊥ shape, regardless of the position of the drone relative to the cubes. For example, in Fig. 14f, the action probability for left is slightly larger than that for right, even though the cube is on the right. For a rational probability for that observation, consider Fig. 13c, where the left action probability dominates other possibilities.

Figure 15 provides attribution visualizations for the right-and-forward policy. In the top row, we see a sequence (a–c) where the policy should be predicting left actions, but it does not. The policy is blind to objects on the left. In the bottom row, sequence (d–f), the policy does respond in an expected manner to objects to the front

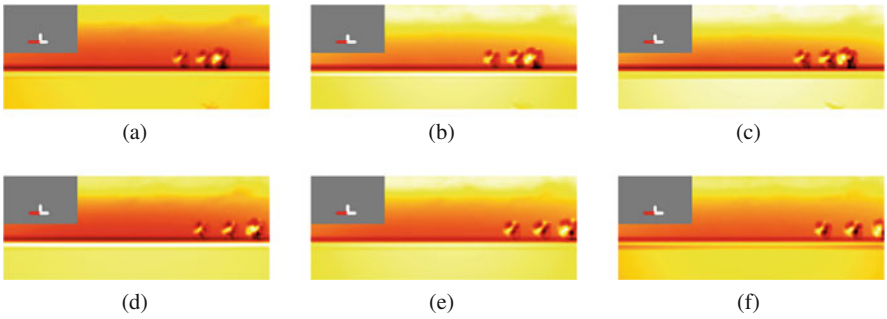


Fig. 14 Attribution visualizations for a sequence of observation-action pairs from the poor policy. Highlighting varies randomly from frame to frame, indicating that the policy has not learned to pay attention to the correct features

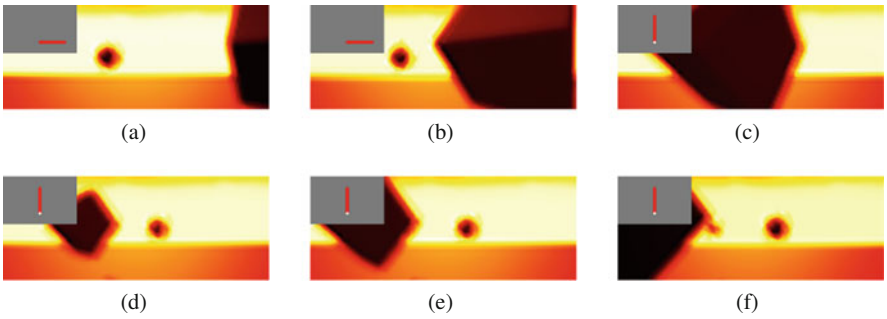


Fig. 15 Attribution visualizations for a sequence of observation-action pairs from the degenerate right-and-forward policy. The policy is not paying any attention to the cube, as it is black. All decisions are made based on the view of the horizon

and right, but it is not the object that triggers the action, but only the shape of the horizon.

6 Conclusion

In this chapter we have presented how to use Microsoft's drone simulation environment and reinforcement learning to train a drone to navigate to and "collect" cubes which are scattered in front of it. In addition, we showed how to use existing deep neural network visualization techniques to understand the reinforcement learning-derived control policies. Because they can be improved and extended, the methods introduced here are a good starting point for multidisciplinary teams aiming to apply reinforcement learning to Cyber-Physical Systems.

Specifically, this chapter may be used by academic or industrial engineering teams as an entry point into the field of vision-based deep reinforcement learning.

In this chapter, a basic reinforcement learning algorithm was introduced, and it may be extended in many ways. For example, if a team has the ability to build optimal control policies, then more advanced reinforcement learning methods may be used. Similarly, if a team has the ability to build physical drones, then the policies learned in simulation may be transferred to the physical drone. The avenues for enhancing what was presented here are limited only by the diversity of the team.

The source code for the cube collection environment and solution is available at https://github.com/RodgerLuo/CPS-Book-Chapter.

References

1. R.N Charette, This car runs on code. IEEE Spectr. **46**(3), 3 (2009)
2. A. Krizhevsky, I. Sutskever, G.E. Hinton, Imagenet classification with deep convolutional neural networks, in *Advances in Neural Information Processing Systems* (2012), pp. 1097–1105
3. R.J. Williams, Simple statistical gradient-following algorithms for connectionist reinforcement learning, in *Reinforcement Learning* (Springer, Berlin, 1992), pp. 5–32
4. S. Shah, D. Dey, C. Lovett, A. Kapoor, Airsim: high-fidelity visual and physical simulation for autonomous vehicles, in *Field and Service Robotics* (2017)
5. L. van der Maaten, G. Hinton, Visualizing data using t-SNE. J. Mach. Learn. Res. **9**, 2579–2605 (2008)
6. K. Simonyan, A. Vedaldi, A. Zisserman, Deep inside convolutional networks: visualising image classification models and saliency maps. Preprint, arXiv:1312.6034 (2013)
7. C. Olah, A. Mordvintsev, L. Schubert, Feature visualization, in *Distill* (2017)
8. R.R. Selvaraju, M. Cogswell, A. Das, R. Vedantam, D. Parikh, D. Batra, Grad-CAM: visual explanations from deep networks via gradient-based localization, in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2017), pp. 618–626

Identifier Randomization: An Efficient Protection Against CAN-Bus Attacks

Khaled Karray, Jean-Luc Danger, Sylvain Guilley, and M. Abdelaziz Elaabid

Abstract The Cyber-Physical Architecture of vehicles is composed of sensors, actuators, and electronic control units all communicating over shared communication buses. For historical reasons the internal communication buses, as the Controller Area Network (CAN), do not implement security mechanisms; the communications are assumed to be "trusted." Recently these trusted relations have been challenged and leveraged to launch cyber-physical attacks against modern vehicles. As a result, it becomes urgent to enhance the security features of vehicles and notably the robustness of the CAN bus which represents an important channel of attacks.

In this work we develop identifier randomization procedures whose aim is to protect the CAN protocol from reverse-engineering, replay, and injection attacks. The idea behind this proposition is to constantly change the message identifiers in a random fashion in a way that both sender and receiver can recover the original message identifier but not the adversary. We present the main challenges of the CAN-ID randomization solution, we highlight the weaknesses of state-of-the-art solutions presented in other scientific papers, and we propose and study candidate solutions

K. Karray (✉)
Télécom ParisTech, Paris, France
e-mail: khaled.karray@telecom-paristech.fr

J.-L. Danger
Télécom ParisTech, Paris, France

Secure-IC S.A.S., Cesson-Sévigné, France
e-mail: jean-luc.danger@telecom-paristech.fr

S. Guilley
Télécom ParisTech, Paris, France

Secure-IC, Paris, France

École normale supérieure, Paris, France
e-mail: sylvain.guilley@telecom-paristech.fr; sylvain.guilley@secure-ic.com

M. Abdelaziz Elaabid
PSA-GROUPE, Paris, France
e-mail: abdelaziz.elaabid@mpsa.com

Ç. K. Koç (ed.), *Cyber-Physical Systems Security*,
https://doi.org/10.1007/978-3-319-98935-8_11

to overcome these weaknesses. To compare our solutions to state-of-the-art solution, we propose to use the entropy and the conditional entropy as a metrics of security. Results show that the randomization functions that we propose outperform the state-of-the-art solution in terms of both entropy and conditional entropy.

1 Introduction

Two important requirements of today's cars are a high level of safety and connectivity with the outside world. This involves the use of advanced technologies based on a computing infrastructure composed of numerous electronic components—named electronic control units (ECUs)—embedded inside the vehicle. These ECUs are in charge of processing sensed data through embedded sensors and transforming it into commands for the actuators. Communication buses in the automotive domain were introduced as soon as the ECUs embedded in the vehicle have reached a certain level of complexity. This made a point-to-point communication approach no longer viable and impossible to implement and maintain. At that point, the car is a system on its own, as isolated from its external world. The choice of communication buses was not motivated by information security, but rather by safety and robustness issues. The Controller Area Network (CAN) imposed itself as the de facto communication bus for the automotive applications. It implements an approach known as multiplexing, which consists in connecting to the same wires (a bus) a large number of computers. The communication is orchestrated by the protocol. Almost all automotive manufacturers are implementing the CAN bus in their cars. The CAN protocol is used for periodic and event-based messages that allows the ECUs to monitor the vehicle state. It is also used to control and supervise the state of sensors and actuators.

Recently, the CAN protocol has become the center of multiple cyber-security issues [1, 4]. Miller and Valasek [16] showed how the CAN protocol can be an important attack vector that enables remote control of a vehicle. In this context, Hoppe et al. [8] were the first to point out the weaknesses of the CAN bus. These findings were further investigated and confirmed by Koscher et al. [13] and Checkoway et al. [1] that performed frame replay and frame injection attacks on a real vehicle. In these attacks, the attacker physically connects to the CAN network and replays or injects messages on the CAN bus. Given the fact that arbitrary read and write accesses are possible on the CAN network, the attacker who sends the right message with the right identifier and payload cannot be detected. Thus, the attacker message will be processed by all receiving ECUs giving her the ability to anonymously influence and control the behavior of these ECUs. This can greatly impact the safety of passengers.

Even though the CAN standard is used by almost all car manufacturers, each one implements its own messages depending on its own needs and the underlying Cyber-Physical Architecture. Each car manufacturer customizes the set of used message identifiers, payload information, and their periodicity. Hence the attacks, as the injection and replay attacks, are generally specific to a car manufacturer and not always reproducible to another car designed by another company.

The CAN frame does not contain a source and destination, but rather a frame identifier that indicates the "content of the data" conveyed by the frame. The same information is always sent over the same message identifier. For instance, the vehicle speed information coming from the speed sensor is always sent over the same message identifier in order for the receiving ECUs to recognize the message. This makes that some equipment have multi-architectures and are backwards compatible within the same car manufacturer. To overcome this issue, reverse engineering the protocol of a specific manufacturer is an important attack, which gives rise to other attacks against the target architecture. Precisely, the reverse-engineering attack is to identify the message identifiers, their periodicity, and the payload information before injecting messages. Miller et al. [15] show the difficulty to engage in such task. As this step seems particularly tedious given the number of messages used, there are some automatic and statistical tools, as penetration testing tools, that have emerged [19, 21]. These tools can lower the complexity of the reverse-engineering step and make it rather straightforward. Due to these weaknesses, the CAN bus has to be hardened in order to protect the connected car from potentially harmful attacks. Researchers have already proposed possible security countermeasures to protect the CAN bus from some of these attacks. Nevertheless these solutions are not fully satisfactory, as they have flaws which are presented in the next section.

In this chapter we make the following contributions:

- First, we identify state-of-the-art protection mechanisms for the CAN bus and highlight their respective flaws.
- Second, we develop identifier randomization procedures to protect the CAN network from reverse-engineering, replay, and injection attacks, both at software and hardware levels.
- Third, we propose information theory-based metrics to evaluate the proposed methods and to compare them with state-of-the-art solutions.

In what follows, Sect. 2 introduces the main state-of-the-art solutions dedicated to the CAN bus and their flaws. In Sect. 3, we focus on a particular solution, namely, the identifier randomization, and we propose novel randomization strategies both at software and hardware levels, to enhance the security of the CAN protocol. These protections are evaluated with information theoretic metrics. In Sect. 4 we compare the aforementioned randomization strategies with the state-of-the-art solutions. Finally Sect. 5 concludes the chapter.

2 State-of-the-Art CAN Protections

In this section, we identify the current solutions that are proposed to secure the CAN network from different security breaches. We start by explaining the CAN protocol stack; then we expose the principles of protection mechanisms that are payload protection, intrusion detection and prevention, and identifier protection. For each mechanism we identify the flaws and limitations.

2.1 Controller Area Network Overview

The CAN bus is an asynchronous, serial field bus system. It was introduced in 1983 by Bosch company for the networking of control devices in automobiles. The aim of this communication bus was the reduction of cable length and weight. Since 1991, CAN is internationally standardized as ISO 11898 and defines the Layer 2 (data link layer) [10] and Layer 1 (physical layer) [11, 12] of the OSI reference model presented in Fig. 1. The physical layer can be realized in two versions: high-speed CAN (ISO 11898-2) and low-speed CAN (ISO 11898-3). Usually these layers are implemented, respectively, in a CAN transceiver and a CAN controller. A CAN frame (Fig. 2) is composed of multiple fields. It starts with a Start-Of-Frame (SOF) bit, then arbitration field which is the frame identifier (ID), control field that indicates the length of data, data field, followed by a Cyclic Redundancy Check (CRC) field for integrity check and an acknowledgment field, and finally the End-Of-Frame (EOF) field. Each ECU that communicates on the CAN bus is equipped with a CAN controller and a CAN transceiver (Fig. 3). On the application level, whenever the ECU software wants to send a message on the CAN bus, it specifies the ID and the payload to the CAN controller (Layer 2). Then the CAN controller constructs the appropriate CAN frame by adding the remaining fields and sends it to the CAN transceiver (Layer 1) whose role is to physically send the frame on the communication bus.

The CAN bus is event-triggered protocol. Whenever a node needs to send a frame on the bus, it needs to check whether the bus is free; then it starts sending. Sometimes collision between two nodes trying to send information at the same time happens. The CAN protocol gives the priority to one of them according to the arbitration rule. The arbitration in the CAN protocol is decided during the sending of the frame identifier (or arbitration field) and is governed by the following rule.

Data Link (Layer 2)	ISO 11898-1	
Physical (Layer 1)	ISO 11898-2 [CAN High speed]	ISO 11898-3 [CAN Low speed]

Fig. 1 CAN layer model

SOF	Identifier	RTR	IDE	r	DLC	Data	CRC	ACK	EOF
1	11 bits	1	1	1	4 bits	0-8 bytes	16 bits	2	7 bits
	Arbitration Field	Control-Field				Data-Field	Check-Field		

Fig. 2 CAN frame

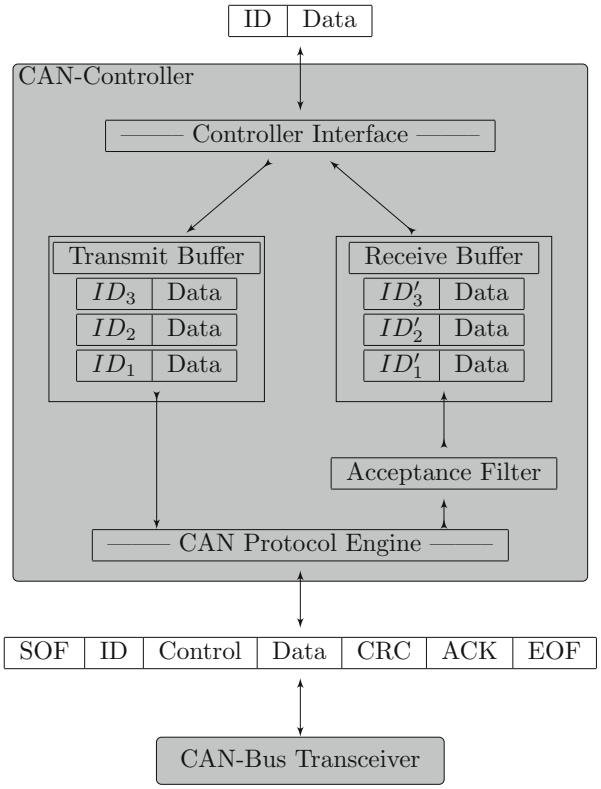


Fig. 3 CAN controller

If one node transmits a dominant bit ("0") and another node transmits a recessive bit ("1"), then there is a collision and the dominant bit "wins." Notice that the ID is sent bit by bit starting from the Most Significant Bit (MSB). Whenever there is a conflict between two ECUs trying to send different messages with different IDs, the smaller ID wins the arbitration and will be sent; the other will have to wait for the next frame. To send a frame to other nodes over the CAN bus, the ECU software needs to specify the ID and payload of the frame to the CAN controller (Layer 2). This information is temporarily stored in a buffer waiting to be sent on the bus. The CAN controller constructs the appropriate frame by adding the remaining fields and sends it to the CAN transceiver (Layer 1). To physically send the frame on the bus, the CAN transceiver needs to check if the bus is free (no information is being sent). Sometimes collision between two nodes trying to send frames at the same times happens. The CAN protocol gives the priority to the frame with lower ID.

Usually, for safety reasons, safety-critical signals are assigned to priority message identifiers. The more the signal is safety relevant, the higher the priority is assigned to its identifier.

2.2 Payload Protection

Historically, one of the first and obvious family of solutions that were proposed to secure the CAN bus is the payload protection solutions. In fact, if we consider the problem as an authenticity violation, the first step toward overcoming this weakness is to protect the payload with cryptographic mechanisms. Nilsson et al. [18] proposed to send message authentication codes over consecutive CAN frames to authenticate the messages. Hartkopp et al. [7] proposed to use Cipher-Based Message Authentication Code (CMAC) as a symmetric authentication measure between the sender and the receiver.

Flaws

The main limitation of this type of protections is that the produced data is larger than the original data which causes a bandwidth overhead on the communication link. For instance, if we want to use an encryption function to protect the confidentiality of the data, state-of-the-art encryption solutions suggest to use no less than 128-bit encryption key with a block cipher of minimum 64 bits (128 bytes for AES-128). This involves a significant increase of data size, as the data size is of 8 bytes or 16 bytes, even if the required payload is a few bytes. Similarly, if we want to protect the authenticity of the sent data, we have to send a data authentication code along with the original data. While the impact of this transformation could be negligible for only one message, the generalization of the use of such solutions to all the messages will cause a significant network overhead. This will have practical side effects such as increasing the delay of messages and increasing errors on the CAN network.

Furthermore, the payload protection mechanism does protect the confidentiality and integrity of the data, but does not protect against reverse-engineering and frame injection attacks. Since the identifiers are kept unchanged, the attacker can still reverse the messages and their periods. Also the attacker can perform exhaustion attack on the ECUs, by sending messages with correct identifier but wrong payload, thus forcing dummy and useless processing.

2.3 Intrusion Detection and Prevention Systems

Another family of protection solutions is known as CAN network Intrusion Detection and Prevention Systems (CAN-IDPS). The role of these systems is to monitor the CAN network for suspicious behavior like frame injection and replay attacks and either physically kill suspicious frame by causing a frame error or by filtering out the suspicious frames. In general, state-of-the-art detection mechanisms can be categorized into two main classes: *rule-based* detection mechanisms and *statistical* detection mechanisms.

Rule-based intrusion detection mechanisms tend to define specific rules of how the network traffic should be. Any message that does not comply with these predefined rules are reported as intrusions. Miller et al. [15] propose to define detection rules of diagnostics messages based on the state of the vehicle and detection rules of periodic messages based on their respective periods. In fact, since the goal of the CAN-IDPS is to protect against injecting extra packets onto the network, and given the fact that most normal frames have predefined frequencies, then if the particular message does not respect its frequency, it should be reported as an intrusion. Taylor et al. [20] propose a detection algorithm based on the comparison of previous and current frame timings to implement this principle. Another example of rule-based detection mechanism is proposed by Marchetti et al. [14] who use the CAN identifier sequence to detect possible injected frames.

Regarding *statistical* detection mechanisms, they define statistical measurements computed over a window of time and used to classify normal and suspicious behavior. In this context, an early work of Müter et al. [17] proposes to use the entropy of the CAN bus as a measurement to classify normal and abnormal behaviors observed on the CAN bus. Dario et al. [2] propose an intrusion detection algorithm that identifies anomalies by computing the Hamming distance between consecutive payloads. This Hamming distance is compared with minimum and maximum thresholds defined during set-up phase.

Flaws

On the one hand, statistical detection measurements do not allow to know the precise CAN frames which have been attacked. They report misbehavior detected on a relatively large time window, which means that the attacker can always inject and replay frames. On the other hand, rule-based detection mechanisms are more effective as they allow only for compliant packets to be accepted. In practice these rules have to be more flexible due to communication imperfections, and the attacker can use this flexibility for her own benefit. If we take the example of frequency-based detection, with a message identifier ID and with a periodicity p, we define a rule that accepts only one packet of identifier ID within a time window p. All the other messages with the same identifier will have to be filtered out or killed. It follows that once the algorithm is synchronized with a legitimate first frame at t_0, the next frames that arrive at $t_0 + n \times p$ will be accepted; all the others will be blocked. In practice the periodicity is not fixed and is subject to a certain variability of approximately 10%. If the attacker injects a frame close to the legitimate frame, the detection algorithm cannot know which one is the legitimate frame. Besides these flaws, neither statistical nor rule-based IDPS, do not protect against reverse engineering of the protocol.

2.4 Identifier Protection

The identifier protection family is efficient to protect the CAN bus from reverse-engineering, injection, and replay attacks. The idea behind this security principle is to make the identifiers not fixed, but instead constantly changing. In fact, if the identifier is not fixed across vehicles, a large-scale attack that could affect all the cars is no longer possible as the identifiers of messages will change from one vehicle to another. Moreover, if the identifiers are not fixed within the same vehicle, a frame replay attack will not succeed because the identifier is constantly changing, and thus no ECU will catch the replayed frame. A frame injection attack neither will work, because the attacker will have to "predict" what will be the next identifier to be injected in order for the attack to be successful.

Humayed et al. [9] proposed a solution called ID-Hopping to counter DOS (Denial-Of-Service) attacks directed against a specific message. Their method works closely with an intrusion detection mechanism which is needed to identify the existence of an attack against a specific message. Once the attack is detected, the ID-Hopping mechanism is activated. Its role is to assign a new but previously defined identifier as a substitute identifier for the attacked message. Another interesting solution to protect the CAN protocol is to constantly randomize the identifier. The constraint is that both the sender and receiver share the same identifier. Han et al. [5, 6] proposed a candidate randomization function. To the best of our knowledge, this solution is the only one that has been proposed for this purpose in the state of the art.

Flaws

While it effectively protects against replay and injection attacks, this randomization principle is not efficient enough to protect against reverse engineering. In the next section, we expose and analyze in details this solution and propose an enhanced protection.

3 Solutions Based on Randomization and Their Evaluation

In this section we focus on the identifier protection family, precisely those based on randomization functions which present the best security features. We first identify the main characteristics and constraints of the randomization function that have to be used for protection. Then we analyze the state-of-the-art solutions and propose new functions that offer better protection from the security point of view. The evaluation of these different solutions is done by defining formal security metrics coming from the information theory.

3.1 Principle and Formalism

The way the CAN protocol is used today by car manufacturers is the following: each information that needs to be communicated between two ECUs is sent in a CAN frame. Figure 4 illustrated the CAN protocol and the histogram of the identifiers. Each frame has a fixed identifier which is known by the sender and the receiver. It can also be known to the attacker as it is communicated in clear on the CAN bus. The identifiers are fixed during the design phase of the vehicle and respond to priority criteria for safety reasons. The priority level defines the criticality of the information and allows the CAN protocol arbitrates between concurrent messages. It is directly linked to the ID value: the lower the ID, the greater the priority of the associated message.

As explained in Sect. 1, the fact that the same information is always sent over the same frame identifier enables the attacker to reverse the protocol and forge frames that can be accepted by the vehicle ECUs. The attacker first starts with a reverse-engineering step during which she identifies the message identifiers and their frequencies. Then she builds an attack by injecting, or replaying, one or multiple CAN frames.

In order to protect the CAN network from such attacks, we want to change the message identifiers every time the ECU needs to send information. This should be done in a way that the receiving ECUs can recover the original identifier and do not allow the attacker to reverse the protocol or inject messages that can be accepted by other ECUs. To do so, an identifier randomization function F is added in such a way that it takes the original identifier ID and substitutes it with another identifier ID_r that changes at every occurrence m of a new frame on the CAN bus. The index m is the value of a message counter which has to be embedded in every ECU for consistency reasons, as m is not communicated on the CAN bus:

$$\mathrm{ID}_r = F(\mathrm{ID}, m) \tag{1}$$

At the receiver side, the ID is recovered by using the inverse function of F and F^{-1}s and the value m of the internal counter of the ECU:

$$\mathrm{ID} = F^{-1}(\mathrm{ID}_r, m) \tag{2}$$

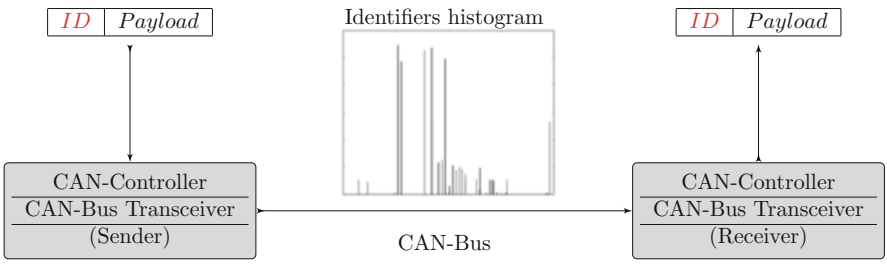


Fig. 4 Controller Area Network with original identifier distribution

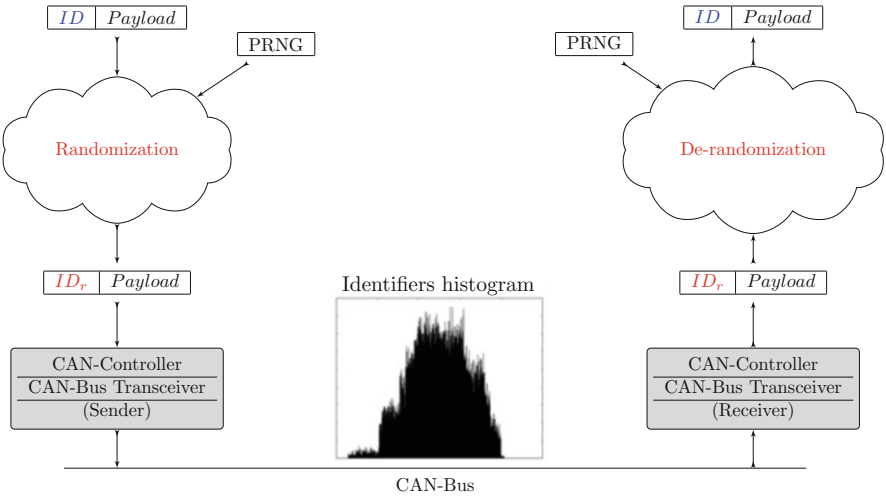


Fig. 5 CAN-ID randomization principle

The randomization function F has to satisfy certain conditions in order to be effective:

- First, F has to be injective and efficiently computable in order for the receiver to recover rapidly the original identifier. Figure 5 shows how this function could be integrated. We can see in this figure that the histogram of randomized identifier ID_r is more spread compared to the one in Fig. 4.
- Second, and for safety reasons, the function F has to be priority-preserving. This means that the priority of message identifiers ID_1 and ID_2 has to be the same as $F(\mathrm{ID}_1)$ and $F(\mathrm{ID}_2)$, respectively. This boils down to the following condition:

$$\mathrm{ID}_1 < \mathrm{ID}_2 \Rightarrow F(\mathrm{ID}_1, m) < F(\mathrm{ID}_2, m) \tag{3}$$

- Third, the priority condition has to be preserved over time. Indeed, the message can go through a transmission buffer before being physically sent to the bus. Consequently, the state of every ECU counter m can be different from the real number of transaction counter on the physical layer. In order to be consistent, the randomization function has to guarantee that the identifiers keep their priority even if the transaction counter m is different. This is expressed by:

$$\begin{aligned} \mathrm{ID}_1 < \mathrm{ID}_2,\ m_1 < m_2 \ \Rightarrow\ & F(\mathrm{ID}_1, m_1) < F(\mathrm{ID}_2, m_2), F(\mathrm{ID}_1, m_2) \\ & < F(\mathrm{ID}_2, m_1) \end{aligned} \tag{4}$$

- Fourth, the output of the randomization function F has to be unpredictable. An attacker that has some information about previous outputs or identifiers should not be able to predict with high probability the randomized identifier. We achieve

this goal by choosing a randomization function based on a cryptographically secure pseudorandom number generator (PRNG).

3.2 Evaluation Metrics

Many functions with randomization can meet the previous constraints. In order to compare these functions between them, we need to define security metrics that measure the ability of these functions to protect against reverse-engineering and replay/injection attacks. These metrics are based on information theory, which links them to *optimal attacks*, that is, attacks which maximize the likelihood of success [3].

3.2.1 Reverse-Engineering Attack

In the presence of a randomization scheme, the attacker knows that each original identifier has multiple substitute identifiers. The reverse-engineering challenge is to be able to determine for each original identifier the set of substitute identifiers that it could be randomized into. A randomization scheme is perfect if the resulting randomized identifiers are identically distributed over the set of identifiers. From an information theory point of view, the capacity of the attacker to perform this task is related to the entropy of the resulting randomized identifiers. The more the identifiers look random, the harder it is for the attacker to reverse the protocol. Thus we use the entropy as a security metrics to evaluate the protection level of the randomization function against reverse engineering:

$$H(id_r) = \sum_{x \in [0, 2^n - 1]} P(id_r = x) \times \log_2 \left(\frac{1}{P(id_r = x)} \right). \tag{5}$$

3.2.2 Replay and Injection Attacks

In order for the attacker to successfully inject a message on the CAN bus with the presence of a randomization function, she needs to "predict" the next randomized identifier to be sent. If the attacker successfully conducts a reverse-engineering attacks, she should be able to predict the next original identifier to be sent. Knowing the original identifier, the attacker has to predict its randomized version. Since we suggested to combine the randomization function with a cryptographically secure pseudorandom number generator that has a uniform distribution, we suppose that the prediction capability of the attacker is not better than a simple "guess." Thus, the conditional entropy of the randomized identifier knowing the original identifier can be used as a metrics to evaluate the protection level of the randomization function

against replay and injection attacks:

$$H(id_r|id_o) = \sum_{x\in[0,2^n-1]} P(id_r = x|id_o) \times \log_2\left(\frac{1}{P(id_r = x|id_o)}\right). \quad (6)$$

3.3 The IA-CAN Approach

In [5, 6] Han et al. gave a method for identifier randomization of the CAN protocol called Identity-Anonymized CAN (IA-CAN). Their approach is to mix a part of the identifier (LSB part) and a part of the payload with a random variable generated at sender and receiver sides. Here we want to focus only on the randomization of the CAN identifier. This is motivated by the fact that if the attacker successfully injects an identifier that gets passed through the CAN filter, even if the rest of the payload is not correct, it will nevertheless exhaust the receiver ECU.

If we disregard the payload part of the anonymization in the IA-CAN approach, we can conclude that the randomization function being used is as follows: (We refer the reader to the original paper [5, 6] for further details.)

$$\begin{array}{ccccc} f_r : & [0, 2^a - 1] & \times & [2^a, 2^n - 1] & \rightarrow [0, 2^n - 1] \\ & r & & id & \rightarrow id_{MSB(n-a)} + id_{LSB(a)} \oplus r \end{array} \quad (7)$$

where

- n is the number of bits of the identifier ($n = 11$ for standard CAN, $n = 29$ for extended CAN).
- a is the number of bits that will be used for randomization ($a < n$).
- r is a random variable in $[0, 2^a - 1]$ generated at both sender and receiver sides.
- id is the original identifier of the message.
- $id_{MSB(\alpha)}$ is the identifier α *M*ost *S*ignificant *B*it.
- $id_{LSB(\alpha)}$ is the identifier α *L*east *S*ignificant *B*it.

We assume that the random number r is uniformly distributed over the randomization interval $[0, 2^a - 1]$. Figure 6 shows the principle of the transformation applied to the identifiers.

Obviously the choice of the variable a is bounded by the total number of original used identifiers N and the minimum of interspace between all identifiers:

$$1 \leq a \leq \left\lfloor \log_2(Min_{i,j\in[1,N]}|id_i - id_j|) \right\rfloor \quad (8)$$

In order to maximize the randomness of the identifiers, we have to choose the maximum possible a : this means that for better security performance, we have to choose:

$$a = \left\lfloor \log_2(Min_{i,j\in[1,N]}|id_i - id_j|) \right\rfloor \quad (9)$$

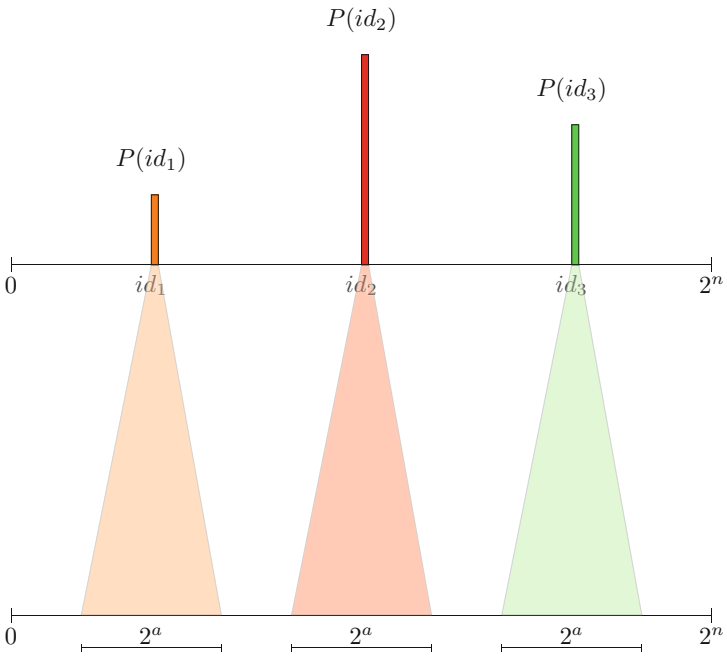


Fig. 6 IA-CAN identifier transformation

3.3.1 Particular Case

A particular case arises when the identifier interspace is constant between all original identifiers. The constant is then $\frac{2^n}{N}$. The upper bound of a is then expressed as the following:

$$\left\lfloor \log_2(Min_{i,j\in[1,N]}|id_i - id_j|) \right\rfloor = n - \left\lceil \log_2(N) \right\rceil \tag{10}$$

To measure the efficiency of this randomization function, we compute the entropy of the randomized identifiers:

$$H_{\mathrm{IA-CAN}}(id_r) = H(id_o) + a \tag{11}$$

And to quantify the attacker capacity to inject new frames, we compute the conditional entropy of the randomized identifiers knowing the original identifiers:

$$H_{\mathrm{IA-CAN}}(id_r|id_o) = a = \left\lfloor \log_2(Min_{i,j\in[1,N]}|id_i - id_j|) \right\rfloor \tag{12}$$

In case the identifier interspace is constant:

$$H_{\mathrm{IA-CAN}}(id_r|id_o) = a = n - \left\lceil \log_2(N) \right\rceil \tag{13}$$

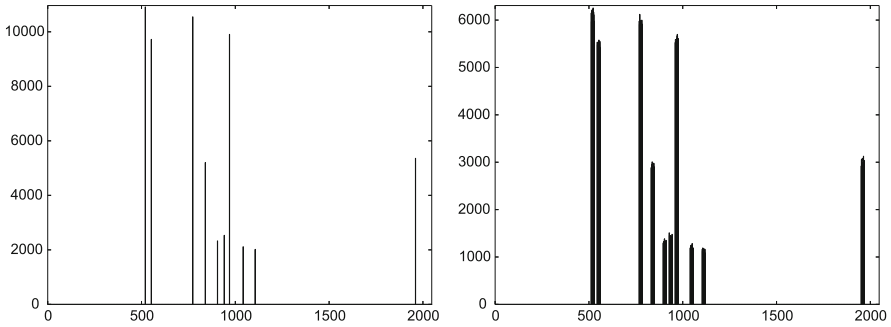


Fig. 7 IA-CAN transformation: Original (left) and randomized (right)

Proof of Eqs. (11) and (12) is presented in the Appendix.

3.3.2 Testing

To test this approach, we made a real acquisition on a vehicle CAN bus, which we used with this randomization procedure to assess its efficiency. Figure 7 shows the identifier histograms before and after randomization. On this particular example, the randomization was done over $a = 4$ bits which means that for each identifier, we allocated $2^a = 2^4 = 16$ substitute identifiers. The computed entropy of the original distribution is $H(id_o) = 3.05$. After randomization, the computed entropy of the randomized identifiers is $H_{\mathrm{IA-CAN}}(id_r) = 7.05$. The computed conditional entropy is $H_{\mathrm{IA-CAN}}(id_r|id_o) = 4$. We can observe from the randomized identifier distribution of Fig. 7 that the attacker can still distinguish frequencies of the messages. It is also clear from Eq. (11) that the entropy of randomized identifiers depends on the entropy of the original identifiers. Using this information the attacker can deduce the next original identifier to be sent and try to inject a frame within the observed randomization interval.

3.4 Equal Intervals

The first observation that we can make concerning the IA-CAN approach is that there is still room for amelioration in terms of entropy and conditional entropy. In fact, the randomization of IA-CAN is done only on the a least significant bits of the identifier, which makes the added entropy bounded by a which is also bounded by $\log_2(Min_{i,j\in[1,N]}|id_i - id_j|)$

A first possible improvement is to create a mapping function that assigns to each original identifier a substitute identifier. The set of substitute identifiers should satisfy the equidistance condition, mentioned in the previous section, that

maximizes random space. A second improvement is to change the randomization function from an XOR function to an arithmetic addition in order to increase the randomness and thus the entropy.

If we have N identifiers $id_1 < id_2 < \cdots < id_N$, we have to partition the identifier space $[0, 2^n - 1]$ over N intervals $I_i = [inf_i, sup_i]$ such that

- $inf_1 = 0, sup_N = 2^n - 1$
- For each $i \in [1, N-1] : inf_{i+1} = sup_i + 1$
- For each $i \in [1, N-1] : sup_i - inf_i = const = \frac{2^n}{N}$

Thus the identifier mapping function Map:

$$\begin{aligned} Map : [0, 2^{n-1} - 1] &\rightarrow [0, 2^n] \\ id_i \qquad &\rightarrow inf_i \end{aligned} \tag{14}$$

The Map function is designed to redefine the distribution of the identifier (by assigning a substitute identifier to the original one) in such a way that the new identifiers maximize the identifier interspace.

Given this new distribution, in the interval $I_i = [inf_i, sup_i]$, we have only one identifier id_i; we can exploit all the interval to randomize that identifier.

Thus the randomization function:

$$\begin{aligned} f_r : [0, 2^{n-1} - 1] &\rightarrow [0, 2^n] \\ id_i \qquad &\rightarrow Map(id_i) + r_{[0, sup_i - inf_i]} \end{aligned} \tag{15}$$

Figure 8 shows the transformation applied to the identifiers. To compare this proposed solution to the previous one, we compute the proposed security metrics:

Entropy:

$$H_{EI}(id_r) = H(id_o) + n - \log_2(N) \tag{16}$$

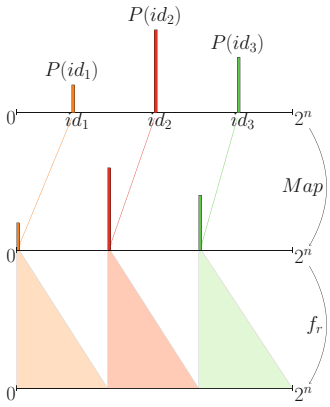


Fig. 8 Equal interval identifier transformation

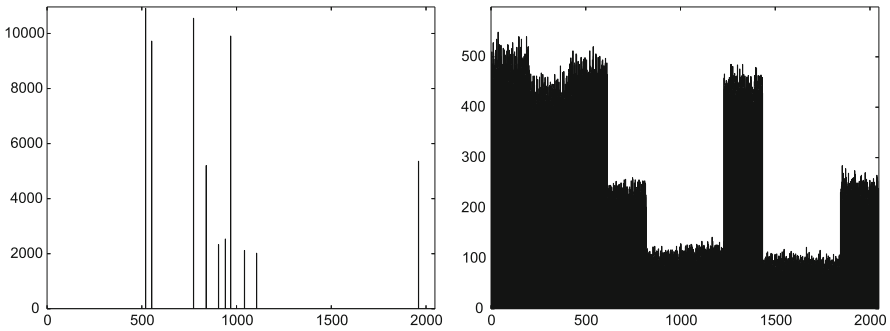


Fig. 9 Equal interval transformation: Original (left) and randomized (right)

Conditional entropy:

$$H_{EI}(id_r|id_o) = n - \log_2(N) \tag{17}$$

Proof of Eqs. (16) and (17) is presented in the Appendix.

In Sect. 4 we show that based on theoretical analysis of the proposed metrics, this randomization function is more secure than the state-of-the-art solution.

3.4.1 Testing

The randomization function is applied to the same identifier distribution used in the previous section. Figure 9 shows the identifier histograms before and after randomization.

We can see that compared to the IA-CAN approach, the equal interval randomization function (15) exploits all the available identifier space. The computed entropy of this particular example is $H_{EI}(id_r) = 10,72$. Compared to the IA-CAN randomization function ($H_{\text{IA}-\text{CAN}} = 7.05$), the equal interval randomization function generates more entropy. The conditional entropy of the randomized identifier knowing the original identifier is $H_{EI}(id_r|id_o) = 7.67$. We can also observe that compared to IA-CAN ($H_{\text{IA}-\text{CAN}}(id_r|id_o) = 4$), it has better conditional entropy. In Sect. 4, we formally prove that it is always the case. Nevertheless the attacker can still identify clusters of randomized identifiers that can guide him in the reverse-engineering process even if the probability of a successful injection is slightly smaller than the previous solution.

3.5 Frequency Intervals

The previous methods are not secure enough against reversing the original identifiers and periods. Indeed, given the histogram, the attacker can identify clusters

of identifiers and thus can identify the original identifier and its frequency. In this section we design a new randomization function whose aim is to overcome this limitation. The goal of this function is to make the randomized identifier distribution [histogram] as uniform as possible to improve the entropy of the randomized identifier by still preserving the priority order. In order to do that, a flattening of the peaks has to be done. We choose the randomization interval of each identifier to be proportional to its frequency of appearance on the CAN bus. Thus an identifier that has a high frequency (small period) will appear more frequently on the CAN bus; this identifier will be assigned a large interval of randomization. Similarly, an identifier that has a small frequency (large period) of appearance on the CAN bus will appear less frequently and thus will be assigned a small interval of randomization. In order for this strategy to be possible, we also need a mapping function that assigns substitute identifiers to the original identifier. Then we apply the randomization to the substitute identifier.

Suppose there are N identifiers $id_1 < id_2 < \cdots < id_N$, respectively, with a sending frequency of $f_1, f_2, \ldots, f_N$. We have to partition the identifier space $[0, 2^n - 1]$ over N intervals $I_i = [inf_i, sup_i]$ such that:

- $inf_1 = 0, sup_k = 2^n - 1$
- For each $i \in [1, N-1] : inf_{i+1} = sup_i + 1$
- For each $i \in [1, N-1] : sup_i - inf_i = \frac{2^n \times f_i}{\sum_{j=1}^{N} f_j} = P(id_i) \times 2^n$

where $P(id_i)$ is the probability of the identifier id_i to appear on the CAN bus.

We define an identifier mapping function Map that assigns substitute identifiers to the original ones such that the identifier interspace is proportional to the frequency of the smaller identifier:

$$\begin{aligned} Map : [0, 2^{n-1} - 1] &\rightarrow [0, 2^n] \\ id_i \qquad &\rightarrow inf_i \end{aligned} \tag{18}$$

The randomization function then assigns a randomized identifier to the substitute identifier. Each identifier is randomized in an interval proportional to its frequency:

$$\begin{aligned} f_r : [0, 2^{n-1} - 1] &\rightarrow [0, 2^n] \\ id_i \qquad &\rightarrow Map(id_i) + r_{[0, sup_i - inf_i]} \end{aligned} \tag{19}$$

Figure 10 shows the transformation applied to the identifiers.

To compare this proposed solution to the previous one, the proposed security metrics is computed:

Entropy:

$$H_{FI}(id_r) = n \tag{20}$$

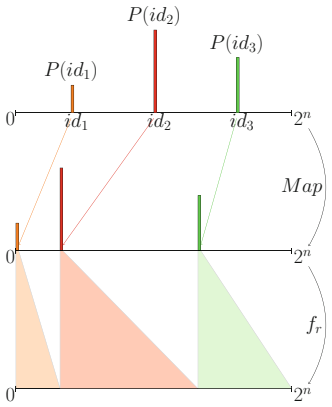


Fig. 10 Frequency interval identifier transformation

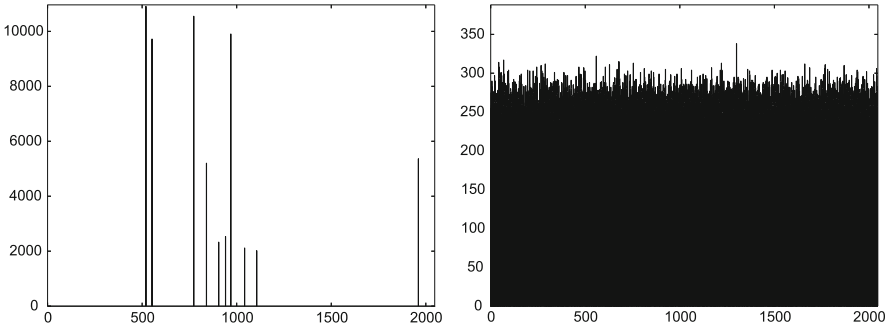


Fig. 11 Frequency interval transformation: Original (left) and randomized (right)

Conditional entropy:

$$H_{FI}(id_r|id_o) = n - H(id) \tag{21}$$

A first observation is that in terms of theoretical entropy, this solution reaches the maximum entropy which is n. Another interesting result is that it gives an enhancement of the conditional entropy as it is shown in Sect. 4. From a theoretical analysis, it is shown in Appendix section "Fixed Mapping Optimality Proof" that the maximum conditional entropy is optimal if the mapping is constant. We will see in the next section that a dynamic mapping will increase the conditional entropy.

3.5.1 Testing

To test this randomization strategy, we apply it to the identifier distribution used for the previous functions. Figure 11 shows the identifier histograms before and

after randomization. The computed entropy for this example is $H_{FI}(id_r) = 10.99$. The computed conditional entropy is $H_{FI}(id_r|id_o) = 7.94$. It is clear from the histogram and the computed entropy that the randomized identifier distribution is more uniform than the previous solutions. A uniform distribution of identifiers is a perfect protection against reverse engineering as it is harder for the attacker to distinguish clusters of identifiers. We can also observe that there is an enhancement in terms of conditional entropy compared to the previous solutions. That is to say, this solution has better security performance against injection attacks.

3.6 Dynamic Intervals

We can now raise the question to increase the conditional entropy obtained with the Frequency Intervals, by using a dynamic mapping. A practical observation of the CAN bus behavior shows that there is a strong dependency between consecutive identifiers: the majority of identifiers will have zero probability to appear right after id_i. This observation involves that using a fixed identifier mapping, after that identifier id_i has been sent, an important part of the allocated space for identifiers will not be used. Hence, if the mapping is changed dynamically after every sending of id_i, and according to the dependency between identifiers, we can increase the conditional entropy.

To construct such randomization function, the Markov matrix can be built to give the probabilities $p_{i,j} = P(id_j^{t+1}/id_i^t)$ of receiving an identifier id_j at iteration $t+1$ knowing that we received the identifier id_i at iteration t:

$$M = \begin{pmatrix} . & id_1^{t+1} & id_2^{t+1} & \dots & id_j^{t+1} & \dots & id_n^{t+1} \\ id_1^t & p(id_1^{t+1}/id_1^t) & p(id_2^{t+1}/id_1^t) & . & . & & p(id_n^{t+1}/id_1^t) \\ id_2^t & p(id_1^{t+1}/id_2^t) & p(id_2^{t+1}/id_2^t) & . & . & & p(id_n^{t+1}/id_2^t) \\ id_3^t & . & . & . & . & & . \\ \vdots & . & . & . & . & & . \\ id_i & . & . & . & p(id_j^{t+1}/id_i^t) & & . \\ \vdots & . & . & . & . & & . \\ id_n^t & . & . & . & . & & . \end{pmatrix} \tag{22}$$

Each time we receive an identifier id_i, immediately after, we have $p(id_j^{t+1}/id_i^t)$ probability to receive id_j.With this in mind, we opt for the Frequency Interval strategy to randomize the upcoming identifiers since it is the optimal strategy that guarantees the maximal entropy. We keep updating the interval partition according to Frequency Interval strategy and to the probabilities in the matrix.

We define the identifier mapping function as the following:

$$\begin{aligned} Map^{t+1} : [0, 2^n - 1] &\rightarrow [0, 2^n - 1] \\ id_{i+1} &\rightarrow id_i + 2^n \times p_{k,i} \end{aligned} \tag{23}$$

The Map function has to be updated every received identifier according to the Frequency Interval strategy. At an instant $t + 1$, knowing that the previous sent identifier is id_k, identifier id_i has an assigned randomization interval of I_i of width $W(I_i) = 2^n \times p_{k,i}$ The resulting randomization function is the following:

$$\begin{aligned} f_r^{t+1} : [0, 2^n - 1] &\rightarrow [0, 2^n - 1] \\ id_i &\rightarrow Map^{t+1}(id_i) + r_{[0,2^n \times p_{k,i}]} \end{aligned} \tag{24}$$

3.6.1 Illustrative Example

As an example, consider the following sequence of identifiers appearing on the CAN bus: $[id_2, id_3, id_1, id_2, id_3, id_1, id_2, id_3, id_2, id_1, id_2, id_3, id_2, id_1, id_2]$.

After analyzing the sequence, the following transition matrix can be established:

$$M = \begin{pmatrix} . & id_1^{t+1} & id_2^{t+1} & id_3^{t+1} \\ id_1^t & 0 & 1 & 0 \\ id_2^t & \frac{1}{3} & 0 & \frac{2}{3} \\ id_3^t & \frac{1}{2} & \frac{1}{2} & 0 \end{pmatrix} \tag{25}$$

This transition matrix is used to define new mapping upon reception of a new identifier. Figure 12 shows the transformation applied to the identifiers after reception of id_2 and then id_3.

The security metrics obtained with the Dynamic Interval strategy is the following:

Entropy:

$$H_{DI}(id_r) = n \tag{26}$$

Conditional entropy:

$$H_{DI}(id_r^{t+1} | id_o^{t+1}) = \sum_{x \in [0,2^n]} \sum_{id_j^{t+1}} \sum_{id_i^t} \frac{1}{W(I_{i,j})} P(id_i) \log_2 \left(\frac{1}{\sum_{id_k^t} \frac{1}{W(I_{k,j})} P(id_k)} \right) \tag{27}$$

3.6.2 Testing

To test this randomization strategy, we apply it to the identifier distribution used for the previous functions. The computed entropy for this example is $H_{DI}(id_r) = 11$.

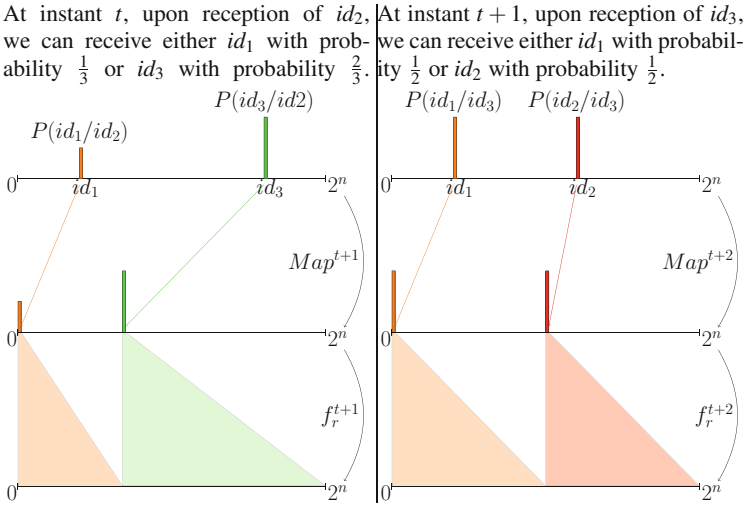


Fig. 12 Dynamic interval identifier transformation at $t+1$ (left) and $t+2$ (right)

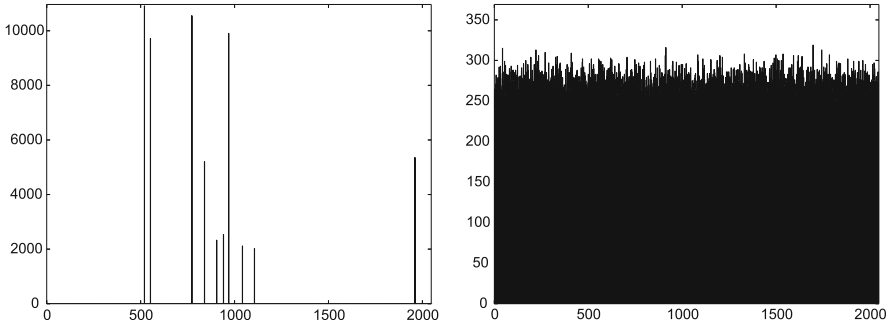


Fig. 13 Dynamic interval transformation: Original (left) and randomized (right)

The computed conditional entropy is $H_{DI}(id_r|id_o) = 10.24$. It is clear from the histogram and the computed entropy that the randomized identifier distribution is uniform as for the frequency interval randomization strategy. Moreover, this method provides a significant enhancement in terms of conditional entropy, compared to the previous solutions (Fig. 13).

3.7 Arithmetic Masking

All of the above-proposed solutions can be applied at software level (Layer 3). This subsection considers a hardware solution which can involve some change in the CAN controllers. The payoff of this choice is to eliminate the third constraint

imposed on Sect. 3.1 that states that the randomization function has to preserve priority over time. Here we consider that the new hardware at physical layer does not have any frame buffer. Hence there is a possibility that all the CAN controllers share the same random variable in a consistent manner. The internal changes of this random variable could be done by a pseudorandom number generator (PRNG) which is initialized identically in every CAN controller at start-up.

The hardware randomization proposal is based on Arithmetic Masking, meaning that the random variable is added arithmetically to the base identifier. The operations are the following:

- First a mapping function is defined. It assigns new substitute identifiers to the original identifiers.
- Then the randomization is performed by adding the random variable to the substitute identifier.
- The random variable is such that it is shared with all CAN controllers and the randomized identifier does not exceed 2^{11}. This allows to preserve the priority between identifiers.

Suppose there are N identifiers $id_1 < id_2 < \cdots < id_N$, with a sending frequencies of $f_1, f_2, \ldots, f_N$. A substitute and random identifier is assigned for each original identifier. The identifier mapping function is defined as the following:

$$\begin{array}{rcl} Map : [0, 2^{n-1} - 1] & \rightarrow & [0, 2^n] \\ id_i & \rightarrow & i - 1 \end{array} \tag{28}$$

The mapping function substitutes the original identifiers with the N first lowest identifiers. The rest of interval $[N, 2^n]$ is used for randomization:

$$\begin{array}{rcl} f_r : [0, 2^n - N] \times [0, 2^{n-1} - 1] & \rightarrow & [0, 2^n] \\ r \qquad\qquad id_i & \rightarrow & Map(id_i) + r \end{array} \tag{29}$$

Figure 14 shows the transformation applied to the identifiers.

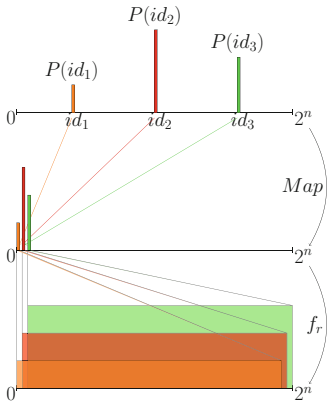


Fig. 14 Arithmetic masking identifier transformation

The security metrics applied to the Arithmetic Masking solution give the following results:

Entropy:

$$H_{AM}(id_r) = \log_2(2^n - N + 1) + \frac{1}{2^n - N + 1} \sum_{x \in [0, N-2]} \sum_{i=0}^{x} P(id_i) \times \log_2 \left(\frac{1}{\sum_{i=0}^{x} P(id_i)} \right) + \sum_{i=x+1}^{N-1} P(id_i) \times \log_2 \left(\frac{1}{\sum_{i=x+1}^{N-1} P(id_i)} \right)$$

Conditional entropy:

$$H_{AM}(id_r | id_o) = \log_2(2^n - N + 1) \tag{30}$$

3.7.1 Testing

To test this randomization strategy, we apply it to the identifier distribution used for the previous functions. The computed entropy for this example is $H_{AM}(id_r) = 10.99$. The computed conditional entropy is $H_{AM}(id_r | id_o) = 10.99$. The histogram and the computed entropy show that the randomized identifier distribution is approximately uniform. Moreover, we observe a significant enhancement in terms of conditional entropy compared to the previous solutions (Fig. 15).

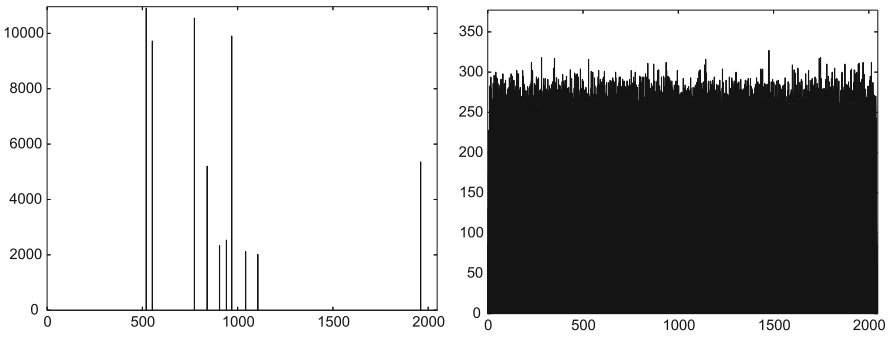


Fig. 15 Arithmetic masking transformation: Original (left) and randomized (right)

4 Comparison

In the previous section, we introduced the state-of-the-art solution for CAN identifier randomization, and we proposed solutions both at software and hardware layers. These solutions were tested on a real identifier trace captured from a real vehicle. In this section a comparison of the proposed solutions is applied to more identifier distributions by using the proposed security metrics.

Four reference identifier distributions are considered. Table 1 summarizes the obtained results. The visual inspection of the histograms indicates that frequency interval and dynamic interval randomization strategies have more uniform distribution than equal intervals and IA-CAN. Hence, these solutions should offer better protection against reverse-engineering attack, at first glance. This observation can be theoretically proven. By comparing the closed-form expressions of the respective entropies, we can establish the following:

$$H_{\mathrm{IA-CAN}}(id_r) \leq H_{EI}(id_r) \leq H_{FI}(id_r) = H_{DI}(id_r) \tag{31}$$

Proof

$$H(id_o) \leq \log_2(N)$$

And we can establish from Eqs. (8) and (10) that:

$$\begin{aligned} a \leq n - \left\lceil \log_2(N) \right\rceil \leq n - \log_2(N) \\ \Rightarrow H(id_o) + a \leq H(id_o) + n - \log_2(N) \\ \Rightarrow H_{\mathrm{IA-CAN}}(id_r) \leq H_{EI}(id_r) \end{aligned}$$

Since

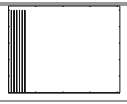
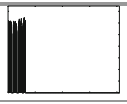
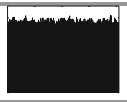
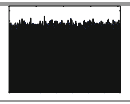
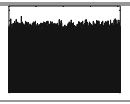
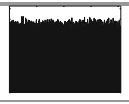
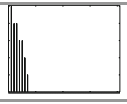
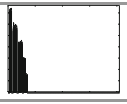
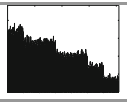
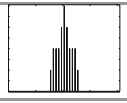
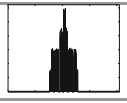
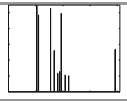
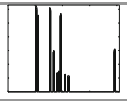
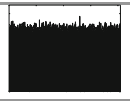
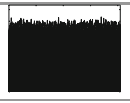
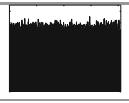
$$H_{DI}(id_r) = H_{FI}(id_r) = n$$

Then:

$$H_{\mathrm{IA-CAN}}(id_r) \leq H_{EI}(id_r) \leq H_{FI}(id_r) = H_{DI}(id_r)$$

It is clear that compared to Arithmetic Masking, Dynamic Intervals and Frequency Intervals have better performance in terms of entropy, involving a high robustness against reverse engineering. Comparing the Arithmetic Masking to IA-CAN and Equal Intervals is not trivial. This is mainly because established entropy expressions depend on the entropy of the original identifier distribution. If we consider that the original identifiers have equal probabilities (example of the first distribution), the Equal Interval solution has better entropy ($H_{EI}(id_r) = 10.9997$, $H_{AM}(id_r) = 10.9948$). Theoretically, the entropy of Equal Intervals for this example is maximal.

Table 1 Comparison between different randomization strategies based on identifier histograms (the X axis represents the identifiers space [0, 2^{11}-1], the Y axis represents the occurrence count of identifiers)

Reference distribution	IA-CAN	Equal intervals	Frequency intervals	Dynamic intervals	Arithmetic masking
					
$H(id_o) = 2.80$	$H(id_r) = 7.80$	$H(id_r) = 10.99$	$H(id_r) = 10.99$	$H(id_r) = 10.99$	$H(id_r) = 10.99$
					
$H(id_o) = 2.68$	$H(id_r) = 7.68$	$H(id_r) = 10.86$	$H(id_r) = 10.99$	$H(id_r) = 11$	$H(id_r) = 10.99$
					
$H(id_o) = 3.35$	$H(id_r) = 8.35$	$H(id_r) = 10.88$	$H(id_r) = 10.99$	$H(id_r) = 11$	$H(id_r) = 10.99$
					
$H(id_o) = 3.05$	$H(id_r) = 7.05$	$H(id_r) = 10.72$	$H(id_r) = 10.99$	$H(id_r) = 11$	$H(id_r) = 10.99$

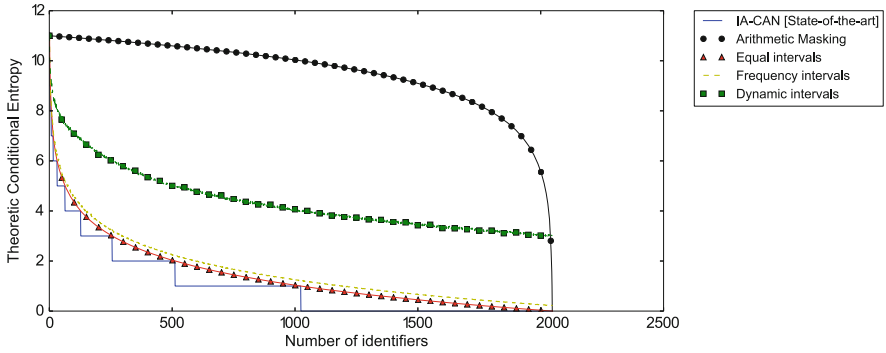


Fig. 16 Conditional entropy $H(id_r|id_o) = f(N)$

On the other side, concerning the second distribution, we can observe that the Arithmetic Masking performs better.

To compare the protection level against replay and injection attacks, the conditional entropy metrics is used. Based on the closed-form expressions established in the previous sections, we draw the curve showing the evolution of the conditional entropy as a function of the total number of identifiers. Figure 16 shows the results. From this graph we can conclude that all the proposed solutions outperform the IA-CAN strategy. Second, it appears that the hardware-based solution, namely, Arithmetic Masking, is the best against replay and injection attacks. However, as discussed previously, the Arithmetic Masking needs to be implemented in the CAN controller between the physical and data link layer, which makes it not easy to deploy. At software level, the Frequency Interval strategy performs the best, both against replay and injection attacks and against reverse engineering.

5 Conclusion

This chapter first presents the state-of-the-art solutions to protect the CAN bus from possible malicious attacks, namely, reverse-engineering, frame injection, and frame replay attacks. It appears that one of the most efficient classes of protection is based on the randomization of the CAN identifiers. Starting from the existing Identity-Anonymized CAN (IA-CAN), three major enhancements based on randomization have been proposed: with Equal Intervals, Frequency Intervals, and Dynamic Intervals. In case it is possible to change the CAN hardware, a randomization based on Arithmetic Masking has also been introduced. The security assessment has been carried out by using security metrics coming from the information theory: entropy (for the reverse-engineering attack) and conditional entropy (for the replay and injection attacks). It has been shown that the enhanced protections provide a significant gain compared to the IA-CAN approach. The entropy obtained from the new randomization solutions is very near the optimum (11 bits), thus presenting

a high robustness against reverse-engineering attacks. The conditional entropy is better achieved with the Arithmetic Masking and the Dynamic Intervals. This last solution has the interest not to modify the hardware of the CAN interface. Overall, the proposed solutions are much better than the existing IA-CAN, as proven by the resulting security gain formally expressed by means of information theory metrics.

Appendix

Let id_o be a random variable representing original identifiers whose outcome is $id_1, id_2, \ldots, id_N$ with probabilities $P(id_1), P(id_2), \ldots, P(id_N)$. We consider a second random variable id_r representing randomized identifiers whose outcome is in $[0, 2^n - 1]$.

Entropy of Fixed Mapping

The entropy of the fixed mapping solutions (IA-CAN, equal intervals, frequency intervals) is the following:

- IA-CAN: $H_{\mathrm{IA-CAN}}(id_r) = H(id_o) + a$
- Equal Intervals: $H_{EI}(id_r) = H(id_o) + n - \log_2(N)$
- Frequency Intervals: $H_{FI}(id_r) = n$

Proof According to the fixed mapping randomization functions (IA-CAN, equal intervals, frequency intervals), each identifier id_i is randomized over a fixed interval I_i of width $W(I_i)$. We begin by computing the probability that the random variable id_r takes the value $x \in [0, 2^n]$:

$$P(id_r = x) = \sum_{i=1}^{N} P(id_r = x | id_i) \times P(id_i)$$

The conditional probability of id_r knowing the original identifier $id_o = id_i$:

$$P(id_r = x | id_i) = \frac{1_{I_i}(x)}{W(I_i)}$$

$$H(id_r) = \sum_{x \in [0, 2^n - 1]} P(id_r = x) \times \log_2 \left(\frac{1}{P(id_r = x)} \right)$$

$$= \sum_{x\in[0,2^n-1]} \left[\sum_{i=1}^{N} P(id_i)\frac{1_{I_i}(x)}{W(I_i)}\right] \times \log_2\left(\frac{1}{\left[\sum_{j=1}^{N} P(id_j)\frac{1_{I_j}(x)}{W(I_j)}\right]}\right)$$

$$= \sum_{i=1}^{N} \sum_{x\in[0,2^n-1]} P(id_i)\frac{1_{I_i}(x)}{W(I_i)} \times \log_2\left(\frac{1}{\left[\sum_{j=1}^{N} P(id_j)\frac{1_{I_j}(x)}{W(I_j)}\right]}\right)$$

$$H(id_r) = \sum_{i=1}^{N} \sum_{x\in I_i} P(id_i)\frac{1_{I_i}(x)}{W(I_i)} \times \log_2\left(\frac{1}{\left[\sum_{j=1}^{N} P(id_j)\frac{1_{I_j}(x)}{W(I_j)}\right]}\right)$$

Since the intervals I_i are nonoverlapping: $\forall x \in I_i, \forall j \neq i \rightarrow 1_{I_j}(x) = 0$

We can thus simplify the expression: $\forall x \in I_i, \forall j \neq i \rightarrow \sum_{j=1}^{N} P(id_j)\frac{1_{I_j}(x)}{W(I_j)} = P(id_i)\frac{1_{I_i}(x)}{W(I_i)}$

$$H(id_r) = \sum_{i=1}^{N} \sum_{x\in I_i} P(id_i)\frac{1_{I_i}(x)}{W(I_i)} \times \log_2\left(\frac{1}{P(id_i)\frac{1_{I_i}(x)}{W(I_i)}}\right)$$

$$= \sum_{i=1}^{N} \sum_{x\in I_i} P(id_i)\frac{1}{W(I_i)} \times \log_2\left(\frac{1}{P(id_i)\frac{1}{W(I_i)}}\right)$$

- IA-CAN entropy: $\forall i \in [1, N],\ W(I_i) = 2^a$

$$H(id_r) = \sum_{i=1}^{N} \sum_{x\in I_i} P(id_i)\frac{1}{2^a} \times \log_2\left(\frac{1}{P(id_i)\frac{1}{2^a}}\right) = H(id_o) + a$$

- Equal interval entropy: $\forall i \in [1, N],\ W(I_i) = \frac{2^n}{N}$

$$H(id_r) = \sum_{i=1}^{N} \sum_{x\in I_i} P(id_i)\frac{1}{\frac{2^n}{N}} \times \log_2\left(\frac{1}{P(id_i)\frac{1}{\frac{2^n}{N}}}\right) = H(id_o) + n - \log_2(N)$$

- Frequency interval entropy: $\forall i \in [1, N],\ W(I_i) = 2^n \times P(id_i)$

$$H(id_r) = \sum_{i=1}^{N} \sum_{x \in I_i} P(id_i) \frac{1}{2^n \times P(id_i)} \times \log_2 \left(\frac{1}{P(id_i) \frac{1}{2^n \times P(id_i)}} \right) = n$$

□

Conditional Entropy of Fixed Mapping

The conditional entropy of randomized identifiers knowing the original identifiers of the fixed mapping solutions (IA-CAN, equal intervals, frequency intervals) is the following:

- IA-CAN: $H_{\text{IA}-\text{CAN}}(id_r|id_o) = a$
- Equal Intervals: $H_{EI}(id_r|id_o) = n - \log_2(N)$
- Frequency Intervals: $H_{FI}(id_r|id_o) = n - H(id_o)$

Proof

$$H(id_r|id_o) = H(id_r, id_o) - H(id_o)$$

$$H(id_r, id_o) = \sum_{x \in [0, 2^n - 1]} \sum_{i=0}^{N} P(id_r = x, id_o = id_i) \log_2 \left(\frac{1}{P(id_r = x, id_o = id_i)} \right)$$

$$P(id_r = x, id_o = id_i) = \begin{cases} P(id_i) \times \frac{1}{w(I_i)} & , x \in I_i \\ 0 & , \text{elsewhere} \end{cases}$$

$$H(id_r, id_o) = \sum_{i=0}^{N} \sum_{x \in I_i} \frac{P(id_i)}{w(I_i)} \log_2 \left(\frac{1}{P(id_i) \frac{1}{w(I_i)}} \right)$$

$$H(id_r, id_o) = H(id_r)$$

$$H(id_r|id_o) = H(id_r) - H(id_o)$$

- IA-CAN conditional entropy : $H(id_r|id_o) = a$
- Equal interval conditional entropy : $H(id_r|id_o) = n - \log_2(N)$
- Frequency interval conditional entropy : $H(id_r|id_o) = n - H(id_o)$

□

Entropy of Dynamic Intervals

Let id_o^t be a Markov chain over the space of original identifiers $(id_1, id_2, \ldots id_N)$. And the matrix M presented in Eq. (25) be its transition matrix. Let id_r be the random variable over $[0, 2^n - 1]$, generated using the dynamic interval randomization strategy applied to id_o^t. We have $H_{DI}(id_r) = n$

Proof

$$H(id_r) = \sum_{x \in [0, 2^n - 1]} P(id_r = x) \times \log_2 \left(\frac{1}{P(id_r = x)} \right)$$

$$P(id_r = x) = \sum_{i}^{N} P(id_r = x | id_o^t = id_i) \times P(id_o^t = id_i)$$

$$P(id_r = x) = \sum_{i}^{N} \sum_{j}^{N} P(id_r = x | id_i^t, id_j^{t+1}) \times P(id_j^{t+1} | id_i^t) \times P(id_i^t)$$

$$P(id_r = x | id_i^t, id_j^{t+1}) = \frac{1_{I_{i,j}}(x)}{W(I_{i,j})}$$

where $W(I_{i,j}) = P(id_j^{t+1} | id_i^t) \times 2^n is\ the\ width\ of\ the\ interval\ I_{i,j}$

$$\begin{aligned} P(id_r = x) &= \sum_{i}^{N} \sum_{j}^{N} \frac{1_{I_{i,j}}(x)}{W(I_{i,j})} \times P(id_j^{t+1} | id_i^t) \times P(id_i^t) \\ &= \sum_{i}^{N} \sum_{j}^{N} \frac{1_{I_{i,j}}(x)}{P(id_j^{t+1} | id_i^t) \times 2^n} \times P(id_j^{t+1} | id_i^t) \times P(id_i^t) \\ &= \sum_{i}^{N} \sum_{j}^{N} \frac{1_{I_{i,j}}(x)}{2^n} \times P(id_i^t) \end{aligned}$$

$\forall x \in [0, 2^n - 1], \sum_j^N 1_{I_{i,j}}(x) = 1$

$$P(id_r = x) = \sum_{i}^{N} \frac{1}{2^n} \times P(id_i^t) = \frac{1}{2^n}$$

$$H(id_r) = \sum_{x \in [0, 2^n - 1]} \frac{1}{2^n} \times \log_2 \left(\frac{1}{2^n} \right)$$

$$H(id_r) = n$$

□

Entropy of Arithmetic Masking

Proof

$$H(id_r) = \sum_{x \in [0, 2^n - 1]} P(id_r = x) \log_2 \left(\frac{1}{P(id_r = x)} \right)$$

$$P(id_r = x) = \begin{cases} \sum_{i=0}^{x} \frac{P(id_i)}{2^n - N + 1} & , x \in [0, N-2] \\ \frac{1}{2^n - N + 1} & , x \in [N-1, 2^n - N] \\ \sum_{i=x-2^n+N}^{N-1} \frac{P(id_i)}{2^n - N + 1} & , x \in [2^n - N + 1, 2^n - 1] \end{cases}$$

$$\begin{aligned} H(id_r) = & \sum_{x \in [N-1, 2^n - N]} \frac{1}{2^n - N + 1} \times \log_2(2^n - N + 1) \\ & + \sum_{x \in [0, N-2]} \left[\sum_{i=0}^{x} \frac{P(id_i)}{2^n - N + 1} \right] \times \log_2 \left(\frac{1}{\sum_{i=0}^{x} \frac{P(id_i)}{2^n - N + 1}} \right) \\ & + \sum_{x \in [2^n - N + 1, 2^n - 1]} \left[\sum_{i=x-2^n+N}^{N-1} \frac{P(id_i)}{2^n - N + 1} \right] \\ & \times \log_2 \left(\frac{1}{\sum_{i=x-2^n+N}^{N-1} \frac{P(id_i)}{2^n - N + 1}} \right) \end{aligned}$$

$$\begin{aligned} H(id_r) = & \frac{2^n - 2(N-1)}{2^n - N + 1} \log_2(2^n - N + 1) + \sum_{x \in [0, N-2]} \left[\sum_{i=0}^{x} \frac{P(id_i)}{2^n - N + 1} \right] \\ & \times \log_2 \left(\frac{1}{\sum_{i=0}^{x} \frac{P(id_i)}{2^n - N + 1}} \right) \\ & + \left[\sum_{i=x+1}^{N-1} \frac{P(id_i)}{2^n - N + 1} \right] \times \log_2 \left(\frac{1}{\sum_{i=x+1}^{N-1} \frac{P(id_i)}{2^n - N + 1}} \right) \end{aligned}$$

$$
\begin{aligned}
H(id_r) =& \frac{2^n - 2(N-1)}{2^n - N + 1} \log_2(2^n - N + 1) \\
&+ \sum_{x \in [0, N-2]} \frac{1}{2^n - N + 1} \log_2(2^n - N + 1) \\
&+ \sum_{x \in [0, N-2]} \sum_{i=0}^{x} \frac{P(id_i)}{2^n - N + 1} \times \log_2 \left(\frac{1}{\sum_{i=0}^{x} P(id_i)} \right) \\
&+ \sum_{i=x+1}^{N-1} \frac{P(id_i)}{2^n - N + 1} \times \log_2 \left(\frac{1}{\sum_{i=x+1}^{N-1} P(id_i)} \right) \\
H(id_r) =& \frac{2^n - 2(N-1)}{2^n - N + 1} \log_2(2^n - N + 1) + \frac{N-1}{2^n - N + 1} \log_2(2^n - N + 1) \\
&+ \frac{1}{2^n - N + 1} \sum_{x \in [0, N-2]} \sum_{i=0}^{x} P(id_i) \times \log_2 \left(\frac{1}{\sum_{i=0}^{x} P(id_i)} \right) \\
&+ \sum_{i=x+1}^{N-1} P(id_i) \times \log_2 \left(\frac{1}{\sum_{i=x+1}^{N-1} P(id_i)} \right) \\
H(id_r) =& \frac{2^n - N + 1}{2^n - N + 1} \log_2(2^n - N + 1) \\
&+ \frac{1}{2^n - N + 1} \sum_{x \in [0, N-2]} \sum_{i=0}^{x} P(id_i) \times \log_2 \left(\frac{1}{\sum_{i=0}^{x} P(id_i)} \right) \\
&+ \sum_{i=x+1}^{N-1} P(id_i) \times \log_2 \left(\frac{1}{\sum_{i=x+1}^{N-1} P(id_i)} \right) \\
H(id_r) =& \log_2(2^n - N + 1) + \frac{1}{2^n - N + 1} \sum_{x \in [0, N-2]} \sum_{i=0}^{x} P(id_i) \\
&\times \log_2 \left(\frac{1}{\sum_{i=0}^{x} P(id_i)} \right) + \sum_{i=x+1}^{N-1} P(id_i) \\
&\times \log_2 \left(\frac{1}{\sum_{i=x+1}^{N-1} P(id_i)} \right)
\end{aligned}
$$

□

Conditional Entropy of Arithmetic

The arithmetic masking conditional entropy is:

$$H_{AM}(id_r|id_o) = \log_2(2^n - N + 1)$$

Proof

$$P(id_r = x|id_o = id_i) = \frac{1_{[i-1,2^n-N+i-1]}}{2^n - N + 1}$$

$$H_{AM}(id_r|id_o) = \sum_{i=0}^{N} P(id_i) H(id_r|id_o = id_i)$$

$$H_{AM}(id_r|id_o) = \sum_{i=0}^{N} P(id_i) \sum_{x\in[i-1,2^n-N+i-1]} P(id_r = x|id_o = id_i) \times \log_2\left(\frac{1}{P(id_r = x|id_o = id_i)}\right)$$

$$H_{AM}(id_r|id_o) = \sum_{i=0}^{N} P(id_i) \sum_{x\in[i-1,2^n-N+i-1]} \frac{1}{2^n - N + 1} \log_2\left(\frac{1}{\frac{1}{2^n-N+1}}\right)$$

$$H_{AM}(id_r|id_o) = \sum_{i=0}^{N} P(id_i) \log_2(2^n - N + 1)$$

$$H_{AM}(id_r|id_o) = \log_2(2^n - N + 1)$$

□

Fixed Mapping Optimality Proof

If we adopt a fixed mapping randomization strategy, the optimal solution in terms of conditional entropy is the frequency interval solutions.

Proof In the context of fixed mapping, we want to find the best decomposition of intervals that maximizes the conditional entropy. We previously showed that the conditional entropy of all fixed mapping solutions can be expressed as $H(id_r|id_o) = \sum_{i\in[1,N]} P(id_i) \times \log_2(W_i)$, where I_i is the randomization interval of id_i of width $W(I_i)$. For the fixed mapping solutions, the intervals are nonoverlapping. Besides the width of each interval I_i is positive ($W(I_i) \geq 0$) and their sum equals 2^n. Thus we define the following problem:

$$\underset{\{I_i\},i\in[1,N]}{\text{Argmax}}\ H(id_r|id_o) = \sum_i P(id_i) \times \log_2(W_i)$$

Subject to the following constraints:

$$\begin{aligned} h_0 &: \textstyle\sum_{i\in[1,N]} W_i - 2^n = 0 \\ h_i &: \forall i \in [1, N], -W_i \leq 0 \end{aligned}$$

To find the solution to this problem, we use the Lagrangian multiplier:

$$\mathscr{L}(W_1, \ldots, W_N, \lambda_1, \ldots \lambda_N, \lambda_0) = H(id_r|id_o) + \sum_{j=0}^{N} \lambda_j h_j$$

and solve the equation system: $\frac{\partial \mathscr{L}}{\partial W_i} = 0, \quad \forall i \in [1, N]$

$$\frac{\partial \mathscr{L}}{\partial W_i}(W_1, \ldots, W_N, \lambda_1, \ldots \lambda_N, \lambda_0) = \frac{\partial H}{\partial W_i} + \sum_{j=0}^{N} \lambda_j \frac{\partial h_j}{\partial W_i}$$

$$\forall i \in [0, N] : \lambda_i \times h_i = 0$$

$$\begin{aligned} h_0 &: \textstyle\sum_{i\in[1,N]} W_i - 2^n = 0 \\ h_i &: \forall i \in [1, N], -W_i \leq 0 \end{aligned}$$

We have: $\frac{\partial H}{\partial W_i} = P(id_i) \times \frac{1}{W_i}$ and $\frac{\partial h_0}{\partial W_i} = 1$ and $\frac{\partial h_j}{\partial W_i} = -1$ if $(i = j)$, 0 otherwise

$$\forall i \in [1, N] : P(id_i) \times \frac{1}{W_i} + \lambda_0 - \lambda_i = 0$$

$$\forall i \in [1, N] : \lambda_i \times h_i = 0$$

Resolving this system of equations gives:

$$\lambda_i = 0, \quad \forall i \in [1, N]$$

$$\lambda_0 = \frac{-1}{2^n}$$

Hence:

$$\Rightarrow \forall i \in [1, N] : W_i = P(id_i) \times 2^n$$

□

References

1. S. Checkoway, D. McCoy, B. Kantor, D. Anderson, H. Shacham, S. Savage, K. Koscher, A. Czeskis, F. Roesner, T. Kohno et al., Comprehensive experimental analyses of automotive attack surfaces, in *USENIX Security Symposium*, San Francisco, 2011
2. S. Dario, M. Mirco, C. Michele, Detecting attacks to internal vehicle networks through hamming distance, in *IEEE 2017 AEIT International Annual Conference-Infrastructures for Energy and ICT (AEIT 2017)*, 2017
3. E. de Chérisey, S. Guilley, A. Heuser, O. Rioul, On the optimality and practicability of mutual information analysis in some scenarios. Cryptogr. Commun. **10**(1), 101–121 (2018)
4. I.D. Foster, A. Prudhomme, K. Koscher, S. Savage, Fast and vulnerable: a story of telematic failures, in *WOOT*, 2015
5. K. Han, A. Weimerskirch, K.G. Shin, Automotive cybersecurity for in-vehicle communication, in *IQT Quarterly*, vol. 6 (2014), pp. 22–25
6. K. Han, A. Weimerskirch, K.G. Shin, A practical solution to achieve real-time performance in the automotive network by randomizing frame identifier, in *Escar Conference*, Cologne, Germany, 2015
7. O. Hartkopp, R. Schilling, MaCAN - Message authenticated CAN, in *Escar Conference*, Berlin, 2012
8. T. Hoppe, S. Kiltz, J. Dittmann, Security threats to automotive CAN networks–practical examples and selected short-term countermeasures, in *International Conference on Computer Safety, Reliability, and Security* (Springer, Berlin, 2008), pp. 235–248
9. A. Humayed, B. Luo, Using ID-hopping to defend against targeted DoS on CAN, in *Proceedings of the 1st International Workshop on Safe Control of Connected and Autonomous Vehicles* (ACM, New York, 2017), pp. 19–26
10. ISO, *11898-1–Road Vehicles–Controller Area Network (CAN)–Part 1: Data Link Layer and Physical Signalling* (International Organization for Standardization, Geneva, 2003)
11. ISO, *11898-2–Road Vehicles–Controller Area Network (CAN)–Part 2: High-Speed Medium Access Unit* (International Organization for Standardization, Geneva, 2003)
12. ISO, *11898-3–Road Vehicles–Controller Area Network (CAN)–Part 2: Fault Tolerant Medium Access Unit* (International Organization for Standardization, Geneva, 2003)
13. K. Koscher, A. Czeskis, F. Roesner, S. Patel, T. Kohno, S. Checkoway, D. McCoy, B. Kantor, D. Anderson, H. Shacham et al., Experimental security analysis of a modern automobile, in *2010 IEEE Symposium on Security and Privacy (SP)* (IEEE, Piscataway, 2010), pp. 447–462

14. M. Marchetti, D. Stabili, Anomaly detection of CAN bus messages through analysis of ID sequences, in *2017 IEEE Intelligent Vehicles Symposium (IV)* (IEEE, Piscataway, 2017), pp. 1577–1583
15. C. Miller, C. Valasek, Adventures in automotive networks and control units. *DEF CON* **21**, 260–264 (2013)
16. C. Miller, C. Valasek, Remote exploitation of an unaltered passenger vehicle. Black Hat USA, 2015
17. M. Müter, N. Asaj, Entropy-based anomaly detection for in-vehicle networks, in *2011 IEEE Intelligent Vehicles Symposium (IV)* (IEEE, Piscataway, 2011), pp. 1110–1115
18. D.K. Nilsson, U.E. Larson, E. Jonsson, Efficient in-vehicle delayed data authentication based on compound message authentication codes, in *IEEE 68th Vehicular Technology Conference, 2008. VTC 2008-Fall* (IEEE, Piscataway, 2008), pp. 1–5
19. C. Smith, *The Car Hacker's Handbook: A Guide for the Penetration Tester* (No Starch Press, San Francisco, 2016)
20. A. Taylor, N. Japkowicz, S. Leblanc, Frequency-based anomaly detection for the automotive CAN bus, in *2015 World Congress on Industrial Control Systems Security (WCICSS)* (IEEE, Piscataway, 2015), pp. 45–49
21. Testing CAN Network with help of CANtoolz. https://www.slideshare.net/AlexeySintsov/testing-can-network-with-help-of-cantoolz, 2016. Accessed 1 Jan 2018

Public Key-Based Lightweight Swarm Authentication

Simon Cogliani, Bao Feng, Houda Ferradi, Rémi Géraud, Diana Maimuţ, David Naccache, Rodrigo Portella do Canto, and Guilin Wang

Abstract We describe a lightweight algorithm performing whole-network authentication in a distributed way. This protocol is more efficient than one-to-one node authentication: it results in less communication, less computation and overall lower energy consumption. The proposed algorithm is provably secure and achieves zero-knowledge authentication of a network in a time logarithmic in the number of nodes.

1 Introduction

A growing market focuses on lightweight devices, whose low cost and easy production allow for creative and pervasive uses. The Internet of Things (IoT) consists in spatially distributed nodes that form a network, able to control or monitor physical or environmental conditions (such as temperature, pressure, image and sound), perform computations or store data. IoT nodes are typically low-cost devices with limited computational resources and limited battery. They transmit the data they acquire through the network to a gateway, also called the transceiver, which collects information and sends it to a processing unit. Nodes are usually deployed in hostile environments and are therefore susceptible to physical attacks, harsh weather conditions and communication interferences.

S. Cogliani (✉) · H. Ferradi · R. Géraud · D. Naccache · R. Portella do Canto
École normale supérieure, Paris, France
e-mail: simon.cogliani@ens.fr; houda.ferradi@ens.fr; remi.geraud@ens.fr; david.naccache@ens.fr; rodrigoportella.docanto@ens.fr

B. Feng · G. Wang
Huawei Technologies Co. Ltd., Shenzhen, China
e-mail: bao.feng@huawei.com; guilin.wang@huawei.com

D. Maimuţ
École normale supérieure, Paris, France

Advanced Technologies Institute, Bucharest, Romania
e-mail: diana.maimut@ens.fr; ati@dcti.ro

Ç. K. Koç (ed.), *Cyber-Physical Systems Security*,
https://doi.org/10.1007/978-3-319-98935-8_12

Due to the open and distributed nature of the IoT, security is key to the entire network's proper operation [14]. However, the lightweight nature of sensor nodes heavily restricts the type of cryptographic operations that they can perform, and the constrained power resources make any communication costly.

This chapter describes an authentication protocol that establishes network integrity and leverages the distributed nature of computing nodes to alleviate individual computational effort. This enables the base station to identify which nodes need replacement or attention.

This is most useful in the context of wireless sensor networks and the IoT, but applies equally well to mesh network authentication and similar situations.

1.1 Related Work

Zero-knowledge (ZK) protocols have been considered for authentication of wireless sensor networks. For instance, Anshul and Roy [1] describe a modified version of the Guillou–Quisquater identification scheme [8], combined with the μTesla protocol [11] for authentication broadcast in constrained environments. We stress that the purpose of the scheme of [1], and similar ones, is to authenticate the base station.

Aggregate signature schemes such as [2, 15] may be used to achieve the goal pursued here—however they are intrinsically noninteractive—and the most efficient aggregate constructions use elliptic curve pairings, which require powerful devices.

Closer to our concerns, [13] describes a ZK network authentication protocol, but it only authenticates two nodes at a time, and the base station acts like a trusted third party. As such it takes a very large number of interactions to authenticate the network as a whole.

What we propose instead is a collective perspective on authentication and not an isolated one.

1.2 Structure of This Chapter

Section 2 recalls the Fiat–Shamir authentication scheme and presents a distributed algorithm for topology-aware networks. We describe the core idea of our chapter, a distributed Fiat–Shamir protocol for IoT authentication, in Sect. 3. We analyse the security of the proposed protocol in Sect. 4. Section 5 provides several improvements and explores trade-offs between security, transmission and storage.

2 Preliminaries

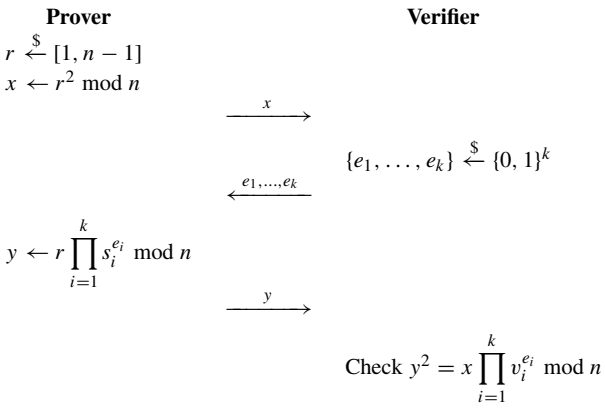
2.1 Fiat–Shamir Authentication

The Fiat–Shamir authentication protocol [5] enables a prover $\mathscr{P}$ to convince a verifier $\mathscr{V}$ that $\mathscr{P}$ possesses a secret key without ever revealing the secret key [4, 7].

The algorithm first runs a one-time setup, whereby a trusted authority publishes an RSA modulus $n = pq$ but keeps the factors p and q private. The prover $\mathscr{P}$ selects a secret $s < n$ such that $\gcd(n, s) = 1$ computes $v = s^2 \bmod n$ and publishes v as its public key.

When a verifier $\mathscr{V}$ wishes to identify $\mathscr{P}$, he uses the protocol of Protocol 1. $\mathscr{V}$ may run this protocol several times until $\mathscr{V}$ is convinced that $\mathscr{P}$ indeed knows the square root s of v modulo n.

Protocol 1: Fiat–Shamir authentication



Protocol 1 describes the original Fiat–Shamir authentication protocol [5], which is *honest verifier* zero knowledge[1] and whose security is proven assuming the hardness of computing arbitrary square roots modulo a composite n, which is equivalent to factoring n.

As pointed out by [5], instead of sending x, $\mathscr{P}$ can hash it and send the first bits of $H(x)$ to $\mathscr{V}$, for instance, the first 128 bits. With that variant, the last step of the protocol is replaced by the computation of $H(y^2 \prod_{i=1}^{k} v_i^{a_i} \bmod n)$, truncated to the first 128 bits and compared to the value sent by $\mathscr{P}$. Using this "short commitment" version reduces somewhat the number of communicated bits. However, it comes at the expense of a reduced security level. A refined analysis of this technique is given in [6].

[1]This can be fixed by requiring $\mathscr{V}$ to commit on the a_i before $\mathscr{P}$ has sent anything, but this modification will not be necessary for our purpose.

2.2 Topology-Aware Distributed Spanning Trees

Due to the unreliable nature of sensors, their small size and wireless communication system, the overall network topology is subject to change. Since sensors send data through the network, a sudden disruption of the usual route may result in the whole network shutting down.

2.2.1 Topology-Aware Networks

A *topology-aware* network detects changes in the connectivity of neighbours, so that each node has an accurate description of its position within the network. This information is used to determine a good route for sending sensor data to the base station. This could be implemented in many ways, for instance, by sending discovery messages (to detect additions) and detecting unacknowledged packets (for deletions). Note that the precise implementation strategy does not impact the algorithm.

Given any graph $G = (V, E)$ with a distinguished vertex B (the base station), the optimal route for any vertex v is the shortest path from v to B on the minimum degree spanning tree $S = (V, E')$ of G. Unfortunately, the problem of finding such a spanning tree is **NP**-hard [12], even though there exist optimal approximation algorithms [9, 12]. Any spanning tree would work for the proposed algorithm; however the performance of the algorithm gets better as the spanning tree degree gets smaller.

2.2.2 Mooij–Goga–Wesselink's Algorithm

The network's topology is described by a spanning tree W constructed in a distributed fashion by the Mooij–Goga–Wesselink algorithm [10]. We assume that nodes can locally detect whether a neighbour has appeared or disappeared, i.e. graph edge deletion and additions.

W is constructed by aggregating smaller subtrees together. Each node in W is attributed a "parent" node, which already belongs to a subtree. The complete tree structure of W is characterized by the parenthood relationship, which the Mooij–Goga–Wesselink algorithm computes. Finally, by topological reordering, the base station $\mathscr{T}$ can be put as the root of W.

Each node in W has three local variables {parent, root, dist} that are initially set to a null value $\perp$. Nodes construct distributively a spanning tree by exchanging "M-messages" containing a root information, distance information and a type. The algorithm has two parts:

- *Basic*: maintains a spanning tree as long as no edge is removed (it is a variant of the union-find algorithm [3]). When a new neighbour w is detected, a discovery M-message (root, dist) is sent to it. If no topology change is detected for w, and

an M-message is received from it, it is processed by Algorithm 1. Note that a node only becomes active upon an event such as the arriving of an M-message or a topology change.
- *Removal*: intervenes after the edge deletion so that the basic algorithm can be run again and give correct results.

Algorithm 1: Mooij–Goga–Wesselink algorithm (Basic part)

Receive: An M-message (r, d) coming from a neighbour w.

1. $(\mathsf{parent}, \mathsf{root}, \mathsf{dist}) \leftarrow (\perp, \perp, \perp)$
2. if $(r, d+1) < (\mathsf{root}, \mathsf{dist})$
3. $\quad \mathsf{parent} \leftarrow w$
4. $\quad \mathsf{root} \leftarrow r$
5. $\quad \mathsf{dist} \leftarrow d + 1$
6. $\quad$ send the M-message $(\mathsf{root}, \mathsf{dist})$ to all neighbours except w

Algorithm 1 has converged once all topology change events have been processed. At that point we have a spanning tree [10].

For our purposes, we may assume that the network was set up and that Algorithm 1 is running on it, so that at all times the nodes of the network have access to their parent node. Note that this incurs very little overhead as long as topology changes are rare.

3 Distributed Fiat–Shamir Authentication

3.1 The Approach

Given a k-node network $\mathscr{N}_1, \ldots, \mathscr{N}_k$, we may consider the nodes $\mathscr{N}_i$ as users and the base station as a trusted centre $\mathscr{T}$. In this context, each node will be given only an[2] s_i. To achieve collective authentication, we propose the following Fiat–Shamir-based algorithm:

- *Step 0:* Wait until the network topology has converged and a spanning tree W is constructed with Algorithm 1 presented in Sect. 2.2. When that happens, $\mathscr{T}$ sends an authentication request message (AR-message) to all the $\mathscr{N}_i$ directly connected to it. The AR-message may contain a commitment to e (*cf.* Step 2) to guarantee the protocol's zero-knowledge property even against dishonest verifiers.

[2]This is for clarity. It is straightforward to give each node several private keys and adapt the algorithm accordingly.

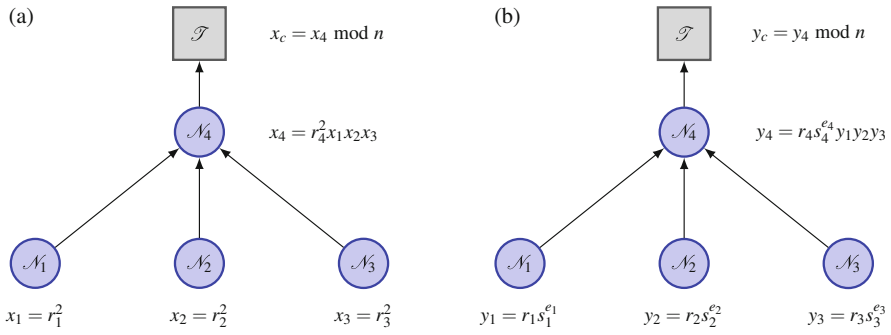


Fig. 1 The proposed algorithm running on a network: computation of x_c (left) and of y_c (right). Each parent node aggregates the values computed by its children and adds its own information before transmitting the result upwards to the base station

- *Step 1:* Upon receiving an AR-message, each $\mathscr{N}_i$ generates a private r_i and computes $x_i \leftarrow r_i^2 \bmod n$. $\mathscr{N}_i$ then sends an A-message to all its children, if any. When they respond, $\mathscr{N}_i$ multiplies all the x_j sent by its children together, and with its own x_i, and sends the result up to its own parent. This recursive construction enables the network to compute the product of all the x_is and send the result x_c to the top of the tree in d steps (where $d = \deg\ W$). This is illustrated for a simple network including four nodes and a base station in Fig. 1.
- *Step 2:* $\mathscr{T}$ sends a random e as an authentication challenge (AC-message) to the $\mathscr{N}_i$ directly connected to it.
- *Step 3:* Upon receiving an AC-message e, each $\mathscr{N}_i$ computes $y_i \leftarrow r_i s_i^{e_i}$. $\mathscr{N}_i$ then sends the AC-message to all its children, if any. When they respond, $\mathscr{N}_i$ multiplies the y_j values received from all its children together, and with its own y_i, and sends the result to its own parent. The network therefore computes collectively the product of all the y_i's and transmits the result y_c to $\mathscr{T}$. This is illustrated in Fig. 1.
- *Step 4:* Upon receiving y_c, $\mathscr{T}$ checks that $y_c^2 = x_c \prod v_i^{e_i}$, where $v_1, \ldots, v_k$ are the public keys corresponding to $s_1, \ldots, s_k$, respectively.

Note that the protocol may be interrupted at any step. In the version of the algorithm that we have just described, this results in a failed authentication.

3.2 Backup Authentication

Network authentication may fail for many reasons described and analysed in detail in Sect. 4.3.3. As a consequence of the algorithm's distributed nature that we have just described, a single defective node suffices for authentication to fail.

This is the intended behaviour; however there are contexts in which such a brutal answer is not enough, and more information is needed. For instance, one could wish to know *which* node is responsible for the authentication failure.

A simple backup strategy consists in performing *usual* Fiat–Shamir authentication with all the nodes that still respond, to try and identify where the problem lies. Note that, as long as the network is healthy, using our distributed algorithm instead is more efficient and consumes less bandwidth and less energy.

Since all nodes already embark the hardware and software required for Fiat–Shamir computations, and can use the same keys, there is no real additional burden in implementing this solution.

4 Security Proofs

In this section we wish to discuss the security properties relevant to our construction. The first and foremost fact is that algorithm given in Sect. 3 is *correct*: a legitimate network will always succeed in proving its authenticity, provided that packets are correctly transmitted to the base station $\mathscr{T}$ (possibly hopping from node to node) and that nodes perform correct computations.

The interesting part, therefore, is to understand what happens when such hypotheses do *not* hold.

4.1 Soundness

Lemma 1 (Soundness) *If the authentication protocol succeeds, then with overwhelming probability the network nodes are genuine.*

Proof Assume that an adversary $\mathscr{A}$ simulates the whole network, but does not know the s_i, and cannot compute in polynomial time the square roots of the public keys v_i. Then, as for the original Fiat–Shamir protocol [5], the base station will accept $\mathscr{A}$'s identification with probability bounded by 2^k where k is the number of nodes.

4.2 Zero Knowledge

Lemma 2 (Zero Knowledge) *The distributed authentication protocol of Sect. 3.1 achieves statistical zero knowledge.*

Proof Let $\mathscr{P}$ be a prover and $\mathscr{A}$ be a (possibly cheating) verifier, who can use any adaptive strategy and bias the choice of the challenges to try and obtain information about the secret keys.

Consider the following simulator $\mathscr{S}$:

Step 1. Choose $\overline{e} \xleftarrow{\$} \{0,1\}^k$ and $\overline{y} \xleftarrow{\$} [0, n-1]$ using any random tape ω'.
Step 2. Compute $\overline{x} \leftarrow \overline{y}^2 \prod v_i^{\overline{e}_i}$ and output $(\overline{x}, \overline{e}, \overline{y})$.

The simulator $\mathscr{S}$ runs in polynomial time and outputs triples that are indistinguishable from the output of a prover that knows the corresponding private key.

If we assume the protocol is run N times, and that $\mathscr{A}$ has learnt information which we denote η, then $\mathscr{A}$ chooses adaptively a challenge using all information available to it $e(x, \eta, \omega)$ (where ω is a random tape). The proof still holds if we modify $\mathscr{S}$ in the following way:

Step 1. Choose $\overline{e} \xleftarrow{\$} \{0,1\}^k$ and $\overline{y} \xleftarrow{\$} [0, n-1]$ using any random tape ω'.
Step 2. Compute $\overline{x} \leftarrow \overline{y}^2 \prod v_i^{\overline{e}_i}$.
Step 3. If $e(\overline{x}, \eta, \omega) = \overline{e}$, then go to Step 1; else output $(\overline{x}, \overline{e}, \overline{y})$.

Note that the protocol is also "locally" ZK, in the sense that an adversary simulating ℓ out of k nodes of the network still has to face the original Fiat–Shamir protocol.

4.3 Security Analysis

4.3.1 Choice of Parameters

Let λ be a security parameter. To ensure this security level, the following constraints should be enforced on parameters:

- The identification protocol should be run $t \geq \lceil \lambda / k \rceil$ times (according to Lemma 1), which is reasonably close to one as soon as the network is large enough.
- The modulus n should take more than $2^{\lambda t}$ operations to factor.
- Private and public keys are of size comparable to n.

4.3.2 Algorithmic Complexity

The number of operations required to authenticate the network depends on the exact topology at hand, but can safely be bounded above:

- Number of modular squarings: $2kt$
- Number of modular multiplications $\leq 3kt$

In average, each $\mathscr{N}_i$ performs only a constant (a small) number of operations. Finally, only $O(d)$ messages are sent, where d is the degree of the minimum spanning tree of the network. Pathological cases aside, $d = O(\log k)$, so that only a logarithmic number of messages are sent during authentication.

All in all, for $\lambda = 256$, $k = 1024$ nodes and $t = 1$, we have $n \geq 2^{1024}$ and up to 5 modular operations per node.

4.3.3 Root Causes of Authentication Failure

Authentication may fail for several reasons. This may be caused by network disruption, so that no response is received from the network—at which point not much can be done.

However, more interestingly, $\mathscr{T}$ may have received an invalid value of y_c. The possible causes are easy to spot:

1. A topology change occurred during the protocol:
   - If all the nodes are still active and responding, the topology will eventually converge and the algorithm will get back to Step 0.
   - If, however, the topology change is due to nodes being added or removed, the network's integrity has been altered.
2. A message was not transmitted: this is equivalent to a change in topology.
3. A node sent a wrong result. This may stem from low battery failure or when errors appear within the algorithm the node has to perform (fault injection, malfunctioning, etc). In that case authentication is expected to fail.

4.3.4 Effect of Network Noise

Individual nodes may occasionally receive incorrect (ill-formed, or well-formed but containing wrong information) messages, be it during topology reconstruction (M-messages) or distributed authentication (A-messages). Upon receiving incorrect A- or M-messages, nodes may dismiss them or try and acknowledge them, which may result in a temporary failure to authenticate. An important parameter which has to be taken into account in such an authentication context is the number of children of a node. When a node with many children starts failing, all its children are disconnected from the network and cannot be contacted or authenticated anymore. While a dysfunction at the leaf level might be benign, the failure of a fertile node is catastrophic.

4.3.5 Man-in-the-Middle Attacks

An adversary could instal itself between nodes, or between nodes and the base station, and try to intercept or modify communications. Lemma 2 proves that a passive adversary cannot learn anything valuable, and Lemma 1 shows that an active adversary cannot fool authentication.

It is still possible that the adversary *relays* information, but any attempt to intercept or send messages over the network would be detected.

5 Variants and Implementation Trade-Offs

The protocol may be adapted to better fit operational constraints: in the context of IoT, for instance, communication is a very costly operation. We describe variants that aim at reducing the amount of information sent by individual nodes while maintaining security.

5.1 Shorter Challenge Variant

In the protocol, the *long* (say, 128-bit) challenge e is sent throughout the network to all individual nodes. One way to reduce the length of e without compromising security is the following:

- A *short* (say, 80-bit) value e is sent to the nodes.
- Each node i computes $e_i \leftarrow H(e \| i)$ and uses e_i as a challenge.
- The base station also computes e_i the same way and uses this challenge to check authentication.

This variant does not impact security, assuming an ideal hash function H, and it can be used in conjunction with the other improvements described below.

5.2 Multiple-Secret Variant

Instead of keeping one secret value s_i, each node could have multiple-secret values $s_{i,1}, \ldots, s_{i,\ell}$. Note that these additional secrets need not be stored: they can be derived from a secret seed.

The multiple-secret variant is described here for a single node, for the sake of clarity. Upon receiving a challenge e_i (assuming, for instance, that e_i was generated by the above procedure), each node computes a response:

$$y_i \leftarrow r_i s_{i,1}^{e_{i,1}} s_{i,2}^{e_{i,2}} \cdots s_{i,\ell}^{e_{i,\ell}} \bmod n.$$

This can be checked by the verifier by checking whether:

$$y_i^2 \stackrel{?}{=} x_i v_{i,1}^{e_{i,1}} v_{i,2}^{e_{i,2}} \cdots v_{i,\ell}^{e_{i,\ell}} \bmod n.$$

To do swarm authentication, it suffices to perform aggregation as described in the protocol of Sect. 3 at intermediate nodes.

Using this approach, one can adjust the memory-communication trade-off, as the security level is $\lambda = t\ell$ (single-node compromission). Therefore, if $\ell = 80$, for instance, it suffices to authenticate once to get the same security as $t = 80$ authentications with $\ell = 1$ (which is the protocol of Sect. 3). This drastically cuts bandwidth usage, a scarce resource for IoT devices.

Furthermore, computational effort can be reduced by using batch exponentiation techniques to compute y_i.

5.3 Precomputed Alphabet Variant

A way to further reduce computational cost is the following: each node chooses an alphabet of m words $w_0, \ldots, w_{m-1}$ (a word is a 32-bit value) and computes once and for all the table of all pairwise products $p_{i,j} = m_i m_j$. Note that each $p_{i,j}$ entry is 64 bits long.

The values s_i are generated by randomly sampling from this alphabet. Put differently, s_i is built by concatenating u words (bit patterns) taken from the alphabet only. We thus see that the s_i, which are mu-bit integers, can take m^u possible values.

Example 1 For instance, if $m = u = 32$, then s_i is a 1024-bit number chosen amongst $32^{32} = 2^{160}$ possible values. Thanks to the lookup table, most multiplications need not be performed, which provides a substantial speed-up over the naive approach.

The size of the lookup table is moderate, for the example given; all we need to store is $32 \times 31/2 + 32 = 528$ values. This can be further reduced by noting that the first lines in the table can be removed: 32 values are zeros, 31 values are the results of multiplications by 1, 30 values are left shifts by 1 of the previous line, 29 values are the sum of the previous 2 and 28 values are left shifts by 2. Hence all in all the table can be compressed into $528 - 32 - 31 - 29 - 28 = 408$ entries. Because each entry is a word, this boils down to 1632 bytes only.

5.4 Precomputed Combination Variant

The idea is that computational cost can be cut down if we precompute and store some products, only to assemble them online during Fiat–Shamir authentication: the values of $s_{i,1,2} \leftarrow s_{i,1}s_{i,2}$, $s_{i,2,3} \leftarrow s_{i,2}s_{i,3}$, ... are stored in a lookup table.

The use of combined values $s_{i,a,b}$ in the evaluation of y results in three possible scenarios for each:

1. $s_a s_b$ appears in y—the probability of this occurring is 1/4—in which case one additional multiplication must be performed.
2. $s_a s_b$ does not appear in y—the probability of this occurring is 1/4—in which case no action is performed.
3. s_a or s_b appears, but not both—this happens with probability 1/2—in which case one single multiplication is required.

As a result the expected number of multiplications is reduced by 25%, to wit $\frac{3}{4} \times 2^{m-1}$, where m is the size of e.

The method can be extended to work in a window of size $\kappa \geq 2$; for instance, with $\kappa = 3$, we would precompute:

$$
\begin{aligned}
s_{i,3n,3n+1} &\leftarrow s_{i,3n} s_{i,3n+1} \\
s_{i,3n+1,3n+2} &\leftarrow s_{i,3n+1} s_{i,3n+2} \\
s_{i,3n,3n+2} &\leftarrow s_{i,3n} s_{i,3n+2} \\
s_{i,3n,3n+1,3n+2} &\leftarrow s_{i,3n} s_{i,3n+1} s_{i,3n+2}
\end{aligned}
$$

Following the same analysis as above, the expected number of multiplications during the challenge-response phase is $\frac{7}{8} \times \frac{2^m}{3}$. The price to pay is that larger values of κ claim more precomputing and memory. More precisely, we have the following trade-offs, writing $\mu = 2^m \bmod \kappa$:

$$
\begin{aligned}
\text{Multiplications(expected)} &= 2^m \left(\frac{2^\kappa - 1}{2^\kappa} \left(\left\lfloor \frac{2^m}{\kappa} - 1 \right\rfloor \right) - \frac{2^\mu - 1}{2^\mu} \right) \\
\text{Premultiplications} &= \ell - 1 + \left((2^\kappa - \kappa - 1) \left\lfloor \frac{2^m}{\kappa} \right\rfloor \right) + (2^\mu - \mu - 1) \\
\text{Storedvalues} &= (2^\kappa - 1) \left\lfloor \frac{2^m}{\kappa} \right\rfloor + (2^\mu - 1)
\end{aligned}
$$

where ℓ is the number of components of s_i.

6 Conclusion

In this work we describe a distributed Fiat–Shamir authentication protocol that enables network authentication using very few communication rounds, thereby alleviating the burden of resource-limited devices such as wireless sensors and other IoT nodes. Instead of performing one-on-one authentication to check the network's integrity, our protocol gives a proof of integrity for the whole network at once.

References

1. D. Anshul, S. Roy, A ZKP-based identification scheme for base nodes in wireless sensor networks, in *Proceedings of the 2005 ACM Symposium on Applied Computing, SAC '05* (ACM, New York, 2005), pp. 319–323
2. D. Boneh, C. Gentry, B. Lynn, H. Shacham, Aggregate and verifiably encrypted signatures from bilinear maps, in *Advances in Cryptology – EUROCRYPT 2003*, ed. by E. Biham. Lecture Notes in Computer Science, vol. 2656 (Springer, Heidelberg, 2003), pp. 416–432
3. T.H. Cormen, C. Stein, R.L. Rivest, C.E. Leiserson, *Introduction to Algorithms*, 2nd edn. (McGraw-Hill Higher Education, New York, 2001)
4. U. Feige, A. Fiat, A. Shamir, Zero-knowledge proofs of identity. J. Cryptol. **1**(2), 77–94 (1988)
5. A. Fiat, A. Shamir, How to prove yourself: practical solutions to identification and signature problems, in *Advances in Cryptology – CRYPTO'86*, ed. by A.M. Odlyzko. Lecture Notes in Computer Science, vol. 263 (Springer, Heidelberg, 1987), pp. 186–194
6. M. Girault, J. Stern, On the length of cryptographic hash-values used in identification schemes, in *Advances in Cryptology – CRYPTO'94*. ed. by Y. Desmedt. Lecture Notes in Computer Science, vol. 839 (Springer, Heidelberg, 1994), pp. 202–215
7. S. Goldwasser, S. Micali, C. Rackoff, The knowledge complexity of interactive proof-systems (extended abstract), in *Proceedings of the 17th Annual ACM Symposium on Theory of Computing, May 6–8, 1985, Providence, Rhode Island, USA*, ed. by R. Sedgewick (ACM, New York, 1985), pp. 291–304. http://doi.acm.org/10.1145/22145.22178
8. L.C. Guillou, J.J. Quisquater, A practical zero-knowledge protocol fitted to security microprocessor minimizing both trasmission and memory, in *Advances in Cryptology – EUROCRYPT'88*, ed. by C.G. Günther. Lecture Notes in Computer Science, vol. 330 (Springer, Heidelberg, 1988), pp. 123–128
9. C. Lavault, M. Valencia-Pabon, A distributed approximation algorithm for the minimum degree minimum weight spanning trees. J. Parallel Distrib. Comput. **68**(2), 200–208 (2008)
10. A.J. Mooij, N. Goga, J.W. Wesselink, A distributed spanning tree algorithm for topology-aware networks. Technische Universiteit Eindhoven, Department of Mathematics and Computer Science (2003)
11. A. Perrig, R. Szewczyk, J.D. Tygar, V. Wen, D.E. Culler, Spins: security protocols for sensor networks. Wirel. Netw. **8**(5), 521–534 (2002)
12. M. Singh, L.C. Lau, Approximating minimum bounded degree spanning trees to within one of optimal, in *Proceedings of the Thirty-Ninth Annual ACM Symposium on Theory of Computing* (ACM, New York, 2007), pp. 661–670
13. S.K. Udgata, A. Mubeen, S.L. Sabat, Wireless sensor network security model using zero knowledge protocol, in *2011 IEEE International Conference on Communications* (IEEE, Piscataway, 2011), pp. 1–5
14. M. Hashim, G. Santhosh Kumar, A. Sreekumar, Authentication in wireless sensor networks using zero knowledge protocol, in *Computer Networks and Intelligent Computing*. vol. 157 (Springer, Berlin, 2011), pp. 416–421
15. L. Zhang, B. Qin, Q. Wu, F. Zhang, Efficient many-to-one authentication with certificateless aggregate signatures. Comput. Netw. **54**(14), 2482–2491 (2010)

Physical Security Versus Masking Schemes

Jean-Luc Danger, Sylvain Guilley, Annelie Heuser, Axel Legay, and Tang Ming

Abstract Numerous masking schemes have been designed as provable countermeasures against side-channel attacks. However, currently, several side-channel attack models coexist, such as "probing" and "bounded moment" models, at bit or word levels. From a defensive standpoint, it is thus unclear which protection strategy is the most relevant to adopt.

In this survey article, we review adversarial hypotheses and challenge masking schemes with respect to practical attacks. In a view to explain in a pedagogical way how to secure implementations, we highlight the key aspects to be considered when implementing a masking scheme.

1 Context About the Protection Problem

Sensitive computations must be secured against non-invasive attacks, which attempt to correlate the leakage of some operations with a hypothetical model [13].

A protection against this threat is the *masking* [13, Chap. 9] countermeasure. Masking consists in changing the intermediate variables of the computation into

J.-L. Danger (✉)
Télécom ParisTech, Paris, France

Secure-IC S.A.S., Cesson-Sévigné, France
e-mail: jean-luc.danger@telecom-paristech.fr

S. Guilley
Télécom ParisTech, Paris, France

Secure-IC, Paris, France

École normale supérieure, Paris, France

A. Heuser
CNRS, IRISA, Rennes, France

A. Legay
INRIA, IRISA, Rennes, France

T. Ming
Wuhan University, Wuhan, China

Ç. K. Koç (ed.), *Cyber-Physical Systems Security*,
https://doi.org/10.1007/978-3-319-98935-8_13

randomized versions, which are thus decorrelated from the unprotected variables, each being a potential target for a side-channel attack. Indeed, leakage localization tests have been put forward to pinpoint leakages [4, 10, 14]; hence masking schemes must have a full coverage.

In particular, in the field of symmetrical encryption, the mainstream approach consists in the use of purely Boolean structures. This is the case of widely used (and standardized) ciphers, such as DES [15] and AES [16]. In both these examples, the operations (except for simple data move, which will simply amplify the leakage signal-to-noise ratio, but not create new leakage model) simply consist in XORs and in look-up tables (LUTs). It is therefore natural to restrict to so-called Boolean masking, where the only operation in terms of masking is the XOR. This is innately compatible with the functional XOR operations, and LUTs are also easy to protect, e.g., using recomputation [20].

1.1 Nature of Computation

Complex computations, like cryptographic algorithms, can be seen as a sequence of parallel basic operations. In hardware, the basic operations are *logic gates*. They take as input a small amount of bits (e.g., 1, 2, 3, up to maximum 6 usually) and yield another bit according to a Boolean function. In software, the basic operations are *instructions*. They take a couple of operands and yield another one, computed through a deterministic function. For instance, `xor r1 r2 r3` computes the exclusive-or of 32-bit registers `r2` and `r3` and saves the result `r2 xor r3` in `r1`.

1.2 Combinational or Sequential?

We notice the spatio-temporal nature of computation: many bits are manipulated *in parallel*, and sometimes, the computation has loops. In hardware, this is called an *iterative implementation*, for instance, the instantiation of gates for one round of AES-128, which are evaluated *ten* times. In software, interestingly, the loops (in the underlying hardware, i.e., the processor) cannot be unrolled, because all operations pass through a unique *integer unit*. Typically, the accumulated register is updated again and again at each instruction (at least, for most instructions—with the exception of instructions where the result is saved directly in memory).[1]

[1]This highlights a very paradoxical modelization of software, even when is it straight line. A straight-line code is seen as sequential in software where it indeed consists in looping of the hardware accumulator register into itself when operations are chained in series. Obviously, the looping of the accumulator into itself only holds for basic controllers. Performance-oriented processors may behave in a more complex way—typically, the pipeline in a processor can break those loops.

1.3 Outline of the Article

The rest of this article is structured as follows: First of all, the mainstream masking algorithms are presented and challenged from a security standpoint in Sect. 2. Second, masking is analyzed vis-à-vis technological and logical high-order leakage function in Sect. 3. A new definition of realistic security objectives is given in Sect. 4. Eventually, this chapter is concluded in Sect. 5.

2 Definition of t-Order Security by ISW [12]

2.1 Revisiting of ISW Definition

Ishai, Sahai, and Wagner (ISW) define as *stateless circuits* [12, Sec. 2] fully combinational circuits, which are acyclic. This models fully unrolled hardware implementations, which are usually prohibitive in cost, but all the same implemented in some contexts like for extremely low latency or for some sort of side-channel resistance [3]. On the other hand, *stateful circuits* are circuits with loops.

Definition 1 (t-Order Security, as per ISW) According to ISW, a circuit is t-order secure if the attacker can get no information about the unprotected variables by:

1. Using t probes at arbitrary positions in one loop[2]
2. But with the possibility to re-probe (and even to move the t probes) for free at every loop[3]

2.2 Ill-Formed Definition

The problem with this definition of probing security is that it does not characterize well some countermeasures. For instance, *perfect masking* [5] of order t can be either secure or insecure depending on the implementation. The secure

[2] We quote [12, p. 464]: "*a* t-limited adversary is one that can observe at most t wires of the circuit within a certain time period (such as during one clock cycle)."

[3] We quote footnote 6 page 464 of [12]: "By default, we allow the adversary to adaptively move its t probes between time periods, but not within a time period." See also the complement given in [12, pp. 466–467]; we quote next: "Prior to each invocation, the adversary may fix an arbitrary set of t internal wires to which it will gain access in that invocation. We stress that while this choice may be adaptive between invocations, i.e., may depend on the outputs and on wire values observed in previous invocations, the adversary is assumed to be too slow to move its probes while the values propagate through the circuit."

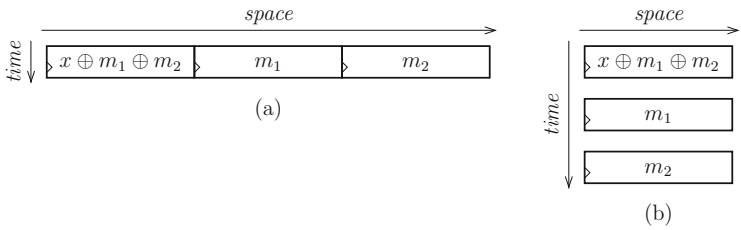


Fig. 1 Parallel (**a**) vs. sequential (**b**) implementation of *perfect masking* for $t = 2$

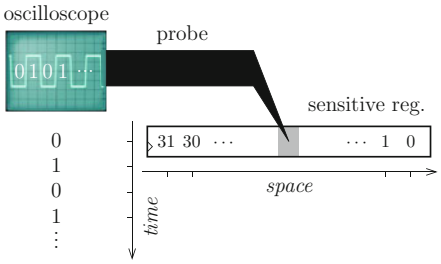


Fig. 2 Linear readout of one bit in a word of $n = 32$ bits, using one prober tip

implementation is the parallel one (see Fig. 1a), because indeed t probes are needed, whereas the insecure implementation is the sequential one (see Fig. 1b), because one probe suffices to read out one bit of the t shares over t clock cycles (even without changing the probe location). This kind of *linear readout* attack scenario is illustrated in Fig. 2; once the probe tip is installed on top on the register bit to probe, the consecutive values held in this DFF (Data Flip-Flop) are read out non-invasively, one after the other. This definition shall not be confused with the more general *(word) probing model* where whole words can be read out at once.

Admittedly (this was the hypothesis in seminal paper of ISW [12]), the relevant security parameter in probing is the number of probes. Indeed:

1. The probe tips are very small, but the probe itself is large (see Fig. 3a); hence only few of them can be placed over a circuit.
2. The step consisting in placing the probe is costly, for at least two reasons. First of all, the identification of the probe's location is time-consuming (it consists, as to say, to identify a needle in a haystack). Second, the positioning of the probe (see Fig. 3a) is slightly invasive, in that it requires to scratch the chip surface to get a reliable electrical contact with the resource to spy (see Fig. 3b).

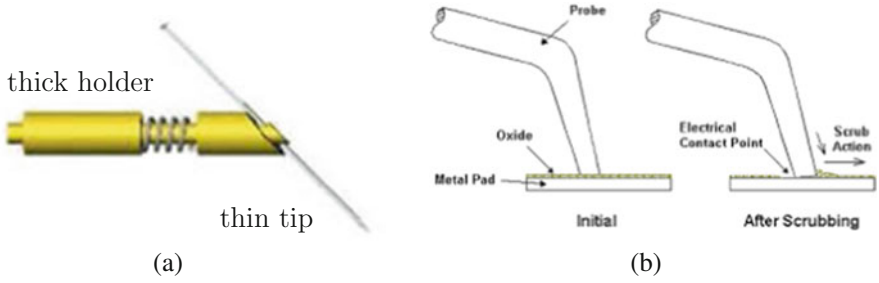


Fig. 3 Probe example, courtesy of http://www.bridgetec.com/holders.html (**a**) and operation for the probe tip to properly contact the targeted area (**b**)

2.3 Attack on Coron's Higher-Order Masking of Look-Up Tables [8]

Coron's higher-order masking of look-up tables [8] is a *software* variant of ISW scheme [12] (more precisely, it is a variant of the word-level variant of ISW, namely [22]). This scheme is proven high-order secure, but the proof is incorrect, because the given implementation and the one assumed in the proof do not match: in the given implementation (algorithms), some resources are reused over time, hence creating a security weakness, as we shall detail in this section.

We recall Coron's masked computation of look-up table $S : \mathbb{F}_2^n \to \mathbb{F}_2^n$ in Algorithm 1, which makes use of masks sharing refresh procedure given in Algorithm 2. In this latter algorithm, the operator "$\leftarrow_{\mathscr{R}}$" stands for uniformly randomized affectation.

We show that there is a second-order attack (in the sense of ISW) on Coron's scheme:

- The attacker probes at line 2 of Algorithm 2; then [one bit of] all the random numbers injected in Algorithm 1 are known. Let us call them $r_{i,j}$ for $0 \leq i \leq t$ (the ith time the RefreshMasks function is called) and for $1 \leq j \leq t$ (the jth fresh mask in invocation i of RefreshMasks).
- In parallel, the attacker also probes [one bit of] y_0 at line 12 of Algorithm 1. This value is equal to $S(\bigoplus_{i=0}^{t} x_i) \oplus \bigoplus_{i'=0}^{i} \bigoplus_{j=1}^{t} r_{i',j}$. As the attacker knows [one bit of] all the $r_{i,j}$, he can deduce [one bit of] $S(\bigoplus_{i=0}^{t} x_i) = S(x)$.

With one probe, only one bit of the $n = 8$ bit register can be probed, which allows nonetheless to recover one bit of $S(x)$ in the clear. However, this is sufficient information to break the AES: after knowing about n values of $S(x)$ targeted bit for $x = p \oplus k$ (plaintext $p \in \mathbb{F}_2^n$ xored with the key byte $k \in \mathbb{F}_2^n$) knowing the values of p, a unique k can be derived.

Input : $x_0, \ldots, x_t$ such that $x = x_0 \oplus \ldots \oplus x_t$
Output : $y_0, \ldots, y_t$ such that $y = S(x) = y_0 \oplus \ldots \oplus y_t$

1 **for** $u \in \mathbb{F}_2^n$ **do**
2 | $T(u) \leftarrow (S(u), 0, \ldots, 0)$ ▷ $\bigoplus_{j=0}^{t} T(u)[j] = S(u)$
3 **end**
4 **for** $i = 0$ to $t-1$ **do**
5 | **for** $u \in \mathbb{F}_2^n$ **do**
6 | | **for** $j = 0$ to t **do** $T'(u)[j] \leftarrow T(u \oplus x_i)[j]$
7 | **end**
| ▷ $\bigoplus_{j=0}^{t} T'(u)[j] = S(u \oplus x_0 \oplus \ldots \oplus x_i)$
8 | **for** $u \in \mathbb{F}_2^n$ **do**
9 | | $T(u) \leftarrow$ RefreshMasks$(T'(u))$
10 | **end**
| ▷ $\bigoplus_{j=0}^{t} T(u)[j] = S(u \oplus x_0 \oplus \ldots \oplus x_i)$
11 **end**
12 $(y_0, \ldots, y_t) \leftarrow$ RefreshMasks$(T(x_t))$ ▷ $\bigoplus_{j=0}^{t} T(x_n)[j] = S(x)$
13 **return** $(y_0, \ldots, y_t)$

Algorithm 1: Masked computation of $y = S(x)$ (Alg. 1 in [8])

Input : $z_0, \ldots, z_t$ such that $z = z_0 \oplus \ldots \oplus z_t$
Output : $z_0, \ldots, z_t$ such that $z = z_0 \oplus \ldots \oplus z_t$

1 **for** $j = 1$ to t **do**
2 | $tmp \leftarrow_{\mathscr{R}} \mathbb{F}_2^n$
3 | $z_0 \leftarrow z_0 \oplus tmp$
4 | $z_j \leftarrow z_j \oplus tmp$
5 **end**
6 **return** $(z_1, \ldots, z_t)$

Algorithm 2: The RefreshMasks function (Alg. 2 in [8])

The values to probe can be found as well in the reference code of https://github.com/coron/htable/blob/master/src/aes_htable.c (hash `a9e88df`, put online on 25 Sep 2015):

- Line 39, in function `subbyte_table` (line 12 of Algorithm 1)
- Line 51, in function `refreshword` (line 2 of Algorithm 2)

The online version of file `aes_htable.c` shall thus not be used as is. Its security problem can be fixed easily, by avoiding the reuse $t-1$ times of tables T and T' and $(t-1) \times t$ times of variable tmp. The corrected algorithm is Algorithm 3 (which calls Algorithms 4 and 5 as subfunctions).

Input : $x_0, \ldots, x_t$ such that $x = x_0 \oplus \ldots \oplus x_t$
Output : $y_0, \ldots, y_t$ such that $y = S(x) = y_0 \oplus \ldots \oplus y_t$

Initialize table $r_{i,j}$ $(0 \leq i \leq t, 1 \leq j \leq t)$ with InitRefreshMasks ▷ using Algorithm 5
for $u \in \mathbb{F}_2^n$ **do**
  $T(u) \leftarrow (S(u), 0, \ldots, 0)$ ▷ $\bigoplus_{j=0}^{t} T(u)[j] = S(u)$
end
for $i = 0$ to $t - 1$ **do**
  for $u \in \mathbb{F}_2^n$ **do**
    for $j = 0$ to t **do** $T'(u + i \times 2^n)[j] \leftarrow T((u \oplus x_i) + i \times 2^n)[j]$
  end
  ▷ $\bigoplus_{j=0}^{t} T'(u + i \times 2^n)[j] = S(u \oplus x_0 \oplus \ldots \oplus x_i)$
  for $u \in \mathbb{F}_2^n$ **do**
    $T(u + (i + 1) \times 2^n) \leftarrow$ RefreshMasks$(i, T'(u + i \times 2^n))$ ▷ using Algorithm 4
  end
  ▷ $\bigoplus_{j=0}^{t} T(u + (i + 1) \times 2^n)[j] = S(u \oplus x_0 \oplus \ldots \oplus x_i)$
end
$(y_0, \ldots, y_t) \leftarrow$ RefreshMasks$(t, T(x_t + t \times 2^n))$ ▷ using Algorithm 4
▷ $\bigoplus_{j=0}^{t} T(x_n + t \times 2^n)[j] = S(x)$
return $(y_0, \ldots, y_t)$

Algorithm 3: Masked computation of $y = S(x)$ (fixed version of Algorithm 1)

Input : Index i, $0 \leq i \leq t$, and $z_0, \ldots, z_t$ such that $z = z_0 \oplus \ldots \oplus z_t$
Output : $z_0, \ldots, z_t$ such that $z = z_0 \oplus \ldots \oplus z_t$

for $j = 1$ to t **do**
  $z_0 \leftarrow z_0 \oplus r_{i,j}$
  $z_j \leftarrow z_j \oplus r_{i,j}$
end
return $(z_1, \ldots, z_t)$

Algorithm 4: The RefreshMasks function (fixed version of Algorithm 2)

Input : None
Output : A table of $t \times (t - 1)$ random numbers

for $i = 0$ to t **do**
  for $j = 1$ to t **do**
    $r_{i,j} \leftarrow_{\mathscr{R}} \mathbb{F}_2^n$
  end
end
return $r_{i,j}$

Algorithm 5: The generation of internal masks InitRefreshMasks

2.4 Motivation for Bit-Mixing Masking Schemes

The attack in the previous Sect. 2.3 has revealed structural weaknesses in state-of-the-art masking schemes. In particular, the attack exploiting the consecutive values taken by one bit allows to break arbitrary high-order masking schemes such as perfect masking scheme [5] or Coron's table-based masking [8].

In reaction to this weakness, so-called *inner product* masking schemes have been proposed [1, 2, 18, 23], which make such attack more chancy. A comprehensive analysis between probing security at bit *versus* word levels is carried out in [7, 19].

3 Analysis of the Security Issue

In the previous section, we showed how one single probe is able to defeat *at bit level* high-order masking schemes proved *at word level*. Therefore, we recommended in Sect. 2.4 masking schemes which combine, by design, several bits together. In this section, we examine how *multi-bit* high-order leakage might arise, created either by the hardware or the software themselves (to the free benefit of the attacker).

3.1 Hardware Case

In the hardware case, coupling between bits can be due:

- *Spatially*, to:
  - *Glitches*: in combinational logic, gates do not evaluate in their order in the netlist (since they are non-synchronizing); for more information on how glitches appear in combinational circuits and contribute to lower the security with respect to side-channel attacks, we refer the reader to the didactic explanations provided in section 4 of [11] devoted to this topic.
  - *IR drop*: individual gates cannot be considered independent, since they share the same power/ground network; the effect is well illustrated in Fig. 9(b) of [9, Sec. 4.2.3].
  - *Capacitive coupling*: some gates, physically placed close one to each other, can have a capacitive coupling of their output nets; the effect is well illustrated in Fig. 9(c) of [9, Sec. 4.2.3].
  - *Unselected gates*: some gates are instantiated in a netlist and supposed to be have a useful functionality only at some times. But actually, being there (i.e., being instantiated and thus activated), they contribute to the leakage continuously, even when they handle data which is eventually not selected (i.e., not used downstream). For example, Fig. 4a illustrates a complete masking scheme, made up of $four$ algorithms: (1) data masking, (2) operation

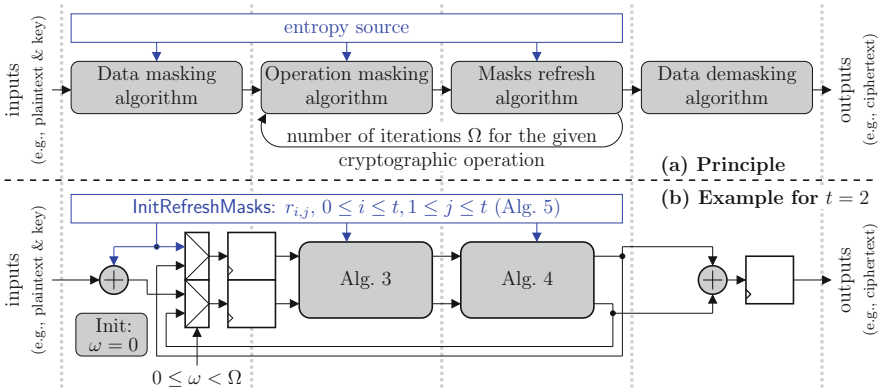


Fig. 4 Masking scheme steps (**a**) and illustration of a netlist for perfect masking with $t = 2$ shares (**b**), with an *unselected gate's* leakage flaw

masking, (3) masks refresh (*optional*), and (4) data unmasking. The last algorithm should, obviously, be executed only at the end of the computation. In the example of a masked iterative block cipher with Ω rounds, the shares can be combined only to recover the ciphertext. However, Fig. 4b shows a faulty implementation of a perfect masking scheme, wherein the data unmasking logic is executed at each round $0 \leq \omega \leq \Omega$ of the block cipher, thereby leaking information on all intermediate rounds.[4]

- *Temporally*, when some gates are reused over time, as explained in the *linear probing* issue (recall Fig. 2 of Sect. 2.2).

3.2 Software Case

In this section, we tackle the question of software security with respect to side-channel leakage. Let us first precise what is implied under the term "software." Software means that some control is written in a memory, but the execution is carried out by one (or several) processor(s). Now, processors are pieces of hardware and hence suffer from the same leakage sources as mentioned in previous Sect. 3.1.

Let us recall two optimizations occurring at *compilation* stage, which make software execution more amenable to side-channel attacks [17]:

[4]Notice that Fig. 4b purposely represents an incorrect masking scheme for an iterative block cipher (for the sake of counterexample) and shall not be implemented this way. Rather, in a secure version, the XOR demasking gate shall be enabled only for the last round (i.e., when round counter ω is equal to its maximal value Ω).

- Register packing consists in regrouping several variables into one register. Indeed, registers are usually wide and hence can accommodate the concatenation of several *words* (say *shares*), to be processed in parallel, e.g., using bitslice operations.
- In Static Single Assignment (SSA) mode, any new variable is affected to a new virtual register. However, in the next pass, registers are allocated. Dead registers are considered as fresh resources and hence are reused, which opens the door to *linear probing* issues.

For the sake of pedagogy, let us make explicit some unusual sources of leakage occurring in software. One important point to make clear is that **glitches do exist in software**. In particular, we find in CPUs the case of leakage of *unselected gates*, owing to unselected logic mentioned in the previous section. Let us illustrate this on the example of two CPUs: 1. 6502, 2. LEON3. For the sake of legibility, lines which are too long (ending by a "\" sign) have been folded.

6502

Let us analyze the integer unit of 6502 processor, described in VHDL in https://github.com/chenxiao07/vhdl-nes/tree/master. Lines 765–772 of source file `vhdl-nes-master/src/free6502.vhd` are recalled below:

```
-- The ALU adder
alu_add <= alu_in1 + alu_add_in2 + alu_add_cin;

-- The ALU itself    This is purely combinatorial logic
process (alu_add, alu_in1, alu_in2, alu_op, c_flag)
begin
  -- default for alu_add inputs
  alu_add_in2 <= alu_in2;
  alu_add_cin <= c_flag;

  case alu_op is
    -- [...]
    when MC_BIT_AND => alu_out <= alu_in1 and alu_in2;
    when MC_BIT_OR =>  alu_out <= alu_in1 or  alu_in2;
    when MC_BIT_XOR => alu_out <= alu_in1 xor alu_in2;
    when MC_BIT_ASL => alu_out <= alu_in1(7 downto 0) & "0";
    when MC_BIT_LSR => alu_out <= alu_in1(0) & \
                                  "0" & alu_in1(7 downto 1);
    when MC_BIT_ROL => alu_out <= alu_in1(7 downto 0) & c_flag;
    when MC_BIT_ROR => alu_out <= alu_in1(0) \
                                  & c_flag & alu_in1(7 downto 1);
    when others =>     alu_out <= alu_in1;
  end case;
end process;
```

LEON3

Let us also analyze the integer unit of LEON3 processor, described in VHDL in iu3.vhd excerpt below:

```
procedure logic_op(r : registers; aluin1, aluin2, mey : word;
      ymsb : std_ulogic; logicres, y : out word) is
variable logicout : word;
begin
  case r.e.aluop is
  when EXE_AND   => logicout := aluin1 and aluin2;
  when EXE_ANDN  => logicout := aluin1 and not aluin2;
  when EXE_OR    => logicout := aluin1 or aluin2;
  when EXE_ORN   => logicout := aluin1 or not aluin2;
  when EXE_XOR   => logicout := aluin1 xor aluin2;
  when EXE_XNOR  => logicout := aluin1 xor not aluin2;
  when EXE_DIV   =>
    if DIVEN then logicout := aluin2;
    else logicout := (others => '-'); end if;
  when others => logicout := (others => '-');
  end case;
  if (r.e.ctrl.wy and r.e.mulstep) = '1' then
    y := ymsb & r.m.y(31 downto 1);
  elsif r.e.ctrl.wy = '1' then y := logicout;
  elsif r.m.ctrl.wy = '1' then y := mey;
  elsif MACPIPE and (r.x.mac = '1') then \
    y := mulo.result(63 downto 32);
  elsif r.x.ctrl.wy = '1' then y := r.x.y;
  else y := r.w.s.y; end if;
  logicres := logicout;
end;
```

Analysis of the 6502 and LEON3 Codes

It clearly appears that both CPUs (6502 and LEON3) do compute all the possible bitwise operations *in parallel* before selecting the one actually relevant for the computation indicated by the current instruction. This behavior is explained in Fig. 5 and corresponds to an *unselected gate's* flaw. In particular, the arithmetic addition combines all the bits (since there is a carry propagation, i.e., the last bit depends on all previous bits) and hence defeats the *register packing* strategy. This is illustrated in Fig. 6. The structure of the full adder (FA) is recalled in Fig. 7.

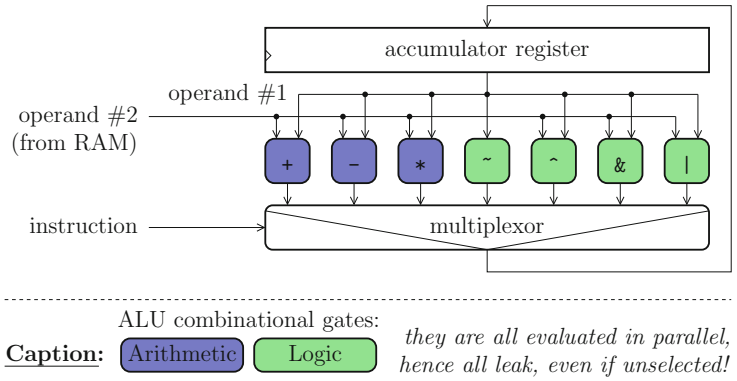


Fig. 5 Illustration of the *unselected gate's* flaw on the ALU (Arithmetic and Logic Unit) of a CPU

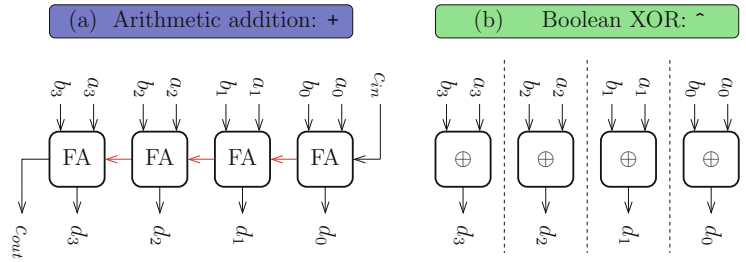


Fig. 6 Illustration addition (+) and exclusive-or (^) operations: (**a**) the addition ($a + b + c_{\text{in}} = 2^n c_{\text{out}} + d$) couples bits of the accumulator (here assumed $n = 4$ bit), through carries represented as red arrows, (**b**) the exclusive-or operation ($a \oplus b = d$) is bitslice, meaning that each bit is computed independently (as illustrated by the "⋮" separation line)

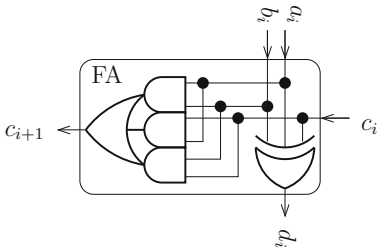


Fig. 7 Full adder, as used in the addition of Fig. 6a

4 New Definition of Security Order

Therefore, in the probing model of ISW, the defender has only one option to enhance the security: disallow the adversary from probing more than once with one probe by

- In hardware: unrolling circuits, hence designing fully combinational logic (as in [3, 24])
- In software: unrolling loops, hence using n^2 times more memory than announced in [8]

For this reason, we propose a new definition of security order. We call this definition the security order in the Noisy Non-Injective (NNI) model.

Definition 2 (t-Order Security, in the NNI Model) The implementation is t-order secure in the NNI model if no information can be recovered by measuring

- t_{space} different bits, at
- t_{time} different times,

where $t_{\text{space}} \times t_{\text{time}} \leq t$.

In this definition,

- t_{space} relates to the "high-order" aspect of side-channel attacks in the NNI model.
- t_{time} relates to the "multivariate" aspect of side-channel attacks.

We introduce the following result to motive for the definition:

Proposition 1 *Let $\mathscr{L}$ be a pseudo-Boolean function $\mathbb{F}_2^n \to \mathbb{R}$, of degree one. Then, assuming the attacker performs a zero-offset attack (i.e., $t_{\text{time}} = 1$), we have that*

$$\forall i, 0 \leq i \leq t, \quad \mathbb{E}(\mathscr{L}(Z)^i | X = x) \quad \text{does not depend on } x$$

if the implementation is t-order secure.

Proof See, for instance, Theorem 2 and/or Proposition 3 in [6].

This means that the attacker will need to raise the leakage traces to the power t_{space}; hence a noise variance raised to the power t_{space}. Assuming that the noise is independent from sample to sample, then we also have that the noise variance is raised to the power t_{time} in t-variate attacks [21].

Thus, the relevant quantity is indeed the product between $t_{\text{space}} \times t_{\text{time}}$.

This model is more satisfactory, as it allows for the designer to do:

- In hardware, fully combinational circuits ($t_{\text{time}} = 1$)
- In software, fully sequential bitslice implementations ($t_{\text{space}} = 1$))

The security notion of Definition 2 is thus more flexible in terms of design solutions to thwart attacks and also more realistic than Definition 1.

However, this model is relevant only in the case where the noise is minimal, so that taking two consecutive measurements decreases the effectiveness of the attack.

5 Conclusion

In this article, we reviewed some masking schemes of the scientific literature. We present their effectiveness with respect to real-world analysis methods and suggest some adaptations. The goal of this article is mostly to underline the discrepancy which can exist between attacks and designs. As of today, attacks arise from the academic world and are pretty virulent. The protection of sensitive circuits is evolving less fast, but a key for a good protection is to understand the risk. The analysis of several adversarial models is performed, and the attacks are confronted to real implementations. We clearly identify a gap between attacks and countermeasures and contribute to bridge it.

Acknowledgements This work was supported in part by the National Natural Science Foundation of China under Grant (61472292, 61332019) and by the key technology research of new-generation high-speed and high-level security chip for smart grid (526816160015).

References

1. J. Balasch, S. Faust, B. Gierlichs, Inner product masking revisited, in *Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26–30, 2015, Proceedings, Part I*, ed. by E. Oswald, M. Fischlin. Lecture Notes in Computer Science, vol. 9056 (Springer, Berlin, 2015), pp. 486–510
2. J. Balasch, S. Faust, B. Gierlichs, C. Paglialonga, F.-X. Standaert, Consolidating inner product masking, in *Advances in Cryptology - ASIACRYPT 2017 - 23rd International Conference on the Theory and Applications of Cryptology and Information Security, Hong Kong, China, December 3–7, 2017, Proceedings, Part I*, ed. by T. Takagi, T. Peyrin. Lecture Notes in Computer Science, vol. 10624 (Springer, Berlin, 2017), pp. 724–754
3. S. Bhasin, S. Guilley, L. Sauvage, J.-L. Danger, Unrolling cryptographic circuits: a simple countermeasure against side-channel attacks, in *Topics in Cryptology - CT-RSA 2010, The Cryptographers' Track at the RSA Conference 2010, San Francisco, CA, USA, March 1–5, 2010. Proceedings*, ed. by J. Pieprzyk. Lecture Notes in Computer Science, vol. 5985 (Springer, Berlin, 2010), pp. 195–207
4. S. Bhasin, J.-L. Danger, S. Guilley, Z. Najm, Side-channel leakage and trace compression using normalized inter-class variance, in *Proceedings of the Third Workshop on Hardware and Architectural Support for Security and Privacy, HASP '14* (ACM, New York, 2014), pp. 7:1–7:9
5. J. Blömer, J. Guajardo, V. Krummel, Provably secure masking of AES, in *Selected Areas in Cryptography*, ed. by H. Handschuh, M.A. Hasan. Lecture Notes in Computer Science, vol. 3357 (Springer, Berlin, 2004), pp. 69–83
6. J. Bringer, C. Carlet, H. Chabanne, S. Guilley, H. Maghrebi, Orthogonal direct sum masking: a smartcard friendly computation paradigm in a code, with builtin protec-

tion against side-channel and fault attacks. Cryptology ePrint Archive, Report 2014/665, 2014. http://eprint.iacr.org/2014/665/ (extended version of conference paper (J. Bringer, C. Carlet, H. Chabanne, S. Guilley, H. Maghrebi, Orthogonal direct sum masking – a smartcard friendly computation paradigm in a code, with builtin protection against side-channel and fault attacks, in *WISTP International Conference on Information Security Theory and Practice*. Lecture Notes in Computer Science, vol. 8501 (Springer, Berlin, 2014), pp. 40–56. Heraklion, Greece))
7. C. Carlet, S. Guilley, Statistical properties of side-channel and fault injection attacks using coding theory. Cryptogr. Commun. **10**(5), 909–933 (2018). https://doi.org/10.1007/s12095-017-0271-4
8. J.-S. Coron, Higher order masking of look-up tables, in *Annual International Conference on the Theory and Applications of Cryptographic Techniques, EUROCRYPT*, ed. by P.Q. Nguyen, E. Oswald. Lecture Notes in Computer Science, vol. 8441 (Springer, Berlin, 2014), pp. 441–458
9. J.-L. Danger, S. Guilley, P. Nguyen, R. Nguyen, Y. Souissi, Analyzing security breaches of countermeasures throughout the refinement process in hardware design flow, in *Design, Automation & Test in Europe Conference & Exhibition, DATE 2017, Lausanne, Switzerland, March 27–31, 2017*, ed. by D. Atienza, G. Di Natale (IEEE, Piscataway, 2017), pp. 1129–1134
10. R.J. Easter, J.-P. Quemard, J. Kondo, Text for ISO/IEC 1st CD 17825 – Information technology – Security techniques – Non-invasive attack mitigation test metrics for cryptographic modules, March 22 2014. Prepared within ISO/IEC JTC 1/SC 27/WG 3. http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=60612
11. S. Guilley, A. Heuser, O. Rioul, Codes for side-channel attacks and protections, in *Codes, Cryptology and Information Security - Second International Conference, C2SI 2017, Rabat, Morocco, April 10–12, 2017, Proceedings - In Honor of Claude Carlet*, ed. by S. El Hajji, A. Nitaj, E.M. Souidi. Lecture Notes in Computer Science, vol. 10194 (Springer, Berlin, 2017), pp. 35–55
12. Y. Ishai, A. Sahai, D. Wagner, Private circuits: securing hardware against probing attacks, in *Annual International Cryptology Conference, CRYPTO*. Lecture Notes in Computer Science, vol. 2729 (Springer, Berlin, 2003), pp. 463–481. Santa Barbara, California
13. S. Mangard, E. Oswald, T. Popp, *Power Analysis Attacks: Revealing the Secrets of Smart Cards* (Springer, Berlin, 2006). http://www.springer.com/. ISBN 0-387-30857-1, http://www.dpabook.org/
14. A. Moradi, S. Guilley, A. Heuser, Detecting hidden leakages, in *Applied Cryptography and Network Security*, ed. by I. Boureanu, P. Owesarski, S. Vaudenay, vol. 8479 (Springer, Berlin, 2014). 12th International Conference on Applied Cryptography and Network Security, Lausanne, Switzerland
15. NIST/ITL/CSD, Data Encryption Standard. FIPS PUB 46-3, Oct 1999. http://csrc.nist.gov/publications/fips/fips46-3/fips46-3.pdf
16. NIST/ITL/CSD, Advanced Encryption Standard (AES). FIPS PUB 197, Nov 2001. http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.197.pdf (also ISO/IEC 18033-3:2010)
17. K. Papagiannopoulos, N. Veshchikov, Mind the gap: towards secure 1st-order masking in software, in *Constructive Side-Channel Analysis and Secure Design: 8th International Workshop, Paris, France, COSADE* (Springer, Berlin, 2017)
18. R. Poussier, Q. Guo, F.-X. Standaert, C. Carlet, S. Guilley, Connecting and improving direct sum masking and inner product masking, in *Smart Card Research and Advanced Applications - 16th International Conference, CARDIS 2017, Lugano, Switzerland, November 13–15, 2017, Revised Selected Papers*, ed. by Y. Teglia, T. Eisenbarth. Lecture Notes in Computer Science (Springer, Berlin, 2017)
19. R. Poussier, Q. Guo, F.-X. Standaert, C. Carlet, S. Guilley, Connecting and improving direct sum masking and inner product masking, in *Smart Card Research and Advanced Applications - 16th International Conference, CARDIS 2017, Lugano, Switzerland, November 13–15, 2017, Revised Selected Papers*, ed. by T. Eisenbarth, Y. Teglia. Lecture Notes in Computer Science, vol. 10728 (Springer, Berlin, 2017), pp. 123–141

20. E. Prouff, M. Rivain, A generic method for secure SBox implementation, in *International Workshop on Information Security Applications WISA*, ed. by Sehun Kim, Moti Yung, and Hyung-Woo Lee. Lecture Notes in Computer Science, vol. 4867 (Springer, Berlin, 2007), pp. 227–244
21. E. Prouff, M. Rivain, Masking against side-channel attacks: a formal security proof, in *Advances in Cryptology - EUROCRYPT 2013, 32nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Athens, Greece, May 26–30, 2013. Proceedings*, ed. by T. Johansson, P.Q. Nguyen. Lecture Notes in Computer Science, vol. 7881 (Springer, Berlin, 2013), pp. 142–159
22. M. Rivain, E. Prouff, Provably secure higher-order masking of AES, in *International Workshop on Cryptographic Hardware and Embedded Systems, CHES*, ed. by S. Mangard, F.-X. Standaert. *Lecture Notes in Computer Science*, vol. 6225 (Springer, Berlin, 2010), pp. 413–427
23. W. Wang, F.-X. Standaert, Y. Yu, S. Pu, J. Liu, Z. Guo, D. Gu, Inner product masking for bitslice ciphers and security order amplification for linear leakages, in *Smart Card Research and Advanced Applications - 15th International Conference, CARDIS 2016, Cannes, France, November 7–9, 2016, Revised Selected Papers*, ed. by K. Lemke-Rust, M. Tunstall. Lecture Notes in Computer Science, vol. 10146 (Springer, Berlin, 2016), pp. 174–191
24. V. Yli-Mäyry, N. Homma, T. Aoki, Improved power analysis on unrolled architecture and its application to PRINCE block cipher, in *Lightweight Cryptography for Security and Privacy - 4th International Workshop, LightSec 2015, Bochum, Germany, September 10–11, 2015, Revised Selected Papers*, ed. by T. Güneysu, G. Leander, A. Moradi. Lecture Notes in Computer Science, vol. 9542 (Springer, Berlin, 2015), pp. 148–163

Embedded Classifiers for Energy-Constrained IoT Network Security

Jennifer Hasler

Abstract We discuss the impact of physical computing techniques to classifying network security issues for ultra-low power networked IoT devices. Energy-constrained IoT systems, such as wearable devices, are already sensor rich and processing/computation constrained. The digital energy efficiency wall constrains the amount of signal processing possible at energy-constrained nodes. One rarely has any computational resources left to consider network security, leaving devices exposed. Fortunately many of these devices have infrequent wireless communication with very constrained command structures, but they still exhibit a system vulnerability, particularly when monitoring or controlling physical infrastructure. Physical computing approaches enable at least a factor of 1000 improvement in computational energy efficiency empowering a new generation of local computational structures for embedded IoT devices. These techniques offer computational capability to address network security concerns.

1 Sensor Nodes Empowered by SoC FPAA Devices

Analog Computing has grown up, fueled through the emergence of large-scale Field-Programmable Analog Array (FPAA) devices (e.g., SoC FPAA [11]), the generalization of FPGAs. Physical computing [12], which includes analog computing, enables both improved computational efficiency (speed and/or larger complexity) of $\times 1000$ or more compared to digital solutions (as predicted by [27]) and potential improvements in area efficiency of x100. Physical computing is now programmable and configurable (e.g., [11]). The rise of programmable and configurable analog techniques (e.g., [11]), integrated with digital processing, enables a wide use of physical computing techniques, not limiting to a few analog IC design specialists [11, 12].

J. Hasler (✉)
Georgia Institute of Technology, Atlanta, GA, USA
e-mail: jennifer.hasler@ece.gatech.edu

Ç. K. Koç (ed.), *Cyber-Physical Systems Security*,
https://doi.org/10.1007/978-3-319-98935-8_14

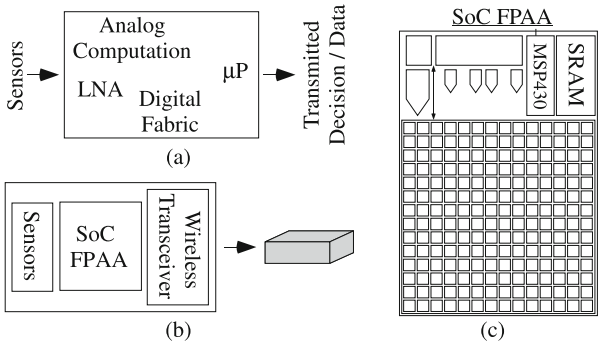


Fig. 1 Embedded configurable physical computation. (**a**) Physical computation in an embedded platform enabling a sea of analog and digital interacting and computation enable significant computing resources moving from sensors to decisions to be communicated. (**b**) Overview picture of the recently published SoC FPAA device [11]. (**c**) A wireless sensor node using this FPAA device, heavily utilizing context-aware techniques. The data from these experimentally measured structures will guide further scaling efforts (size, energy consumed). One application for this sensor network would be for ground-level monitoring of people, cars, trucks, machineries, or other elements through acoustic or MEMS vibration/accelerometer sensors. A second application for this sensor network would be for a body-level sensing network, monitoring the behavior of knees, heart, and other internal organs through a combination of vibrational and acoustic sensors

Figure 1 illustrates discussions on a wireless sensor node utilizing FPAA. FPAA devices allow the user to investigate many physical computing designs within a few weeks of time. The alternative for one design would require years of IC design by potentially multiple individuals. The sensor node could classify (e.g., [11]) and learn (e.g., [15]) from original sensor signals, performing all of the computation required for the computation and operating the entire system in its real-world application environment. Embedded learning approaches, implemented in a single FPAA device, illustrate the small area and ultra-low power capabilities of configurable physical computing. Section 2 discusses the context-aware opportunities in FPAA architectures.

Although the low-power physical computing could have huge impacts for network of autonomous sensor nodes, these FPAA-enabled nodes often require secure operation. Although FPAAs are a recent technology, widespread adoption of these devices eventually requires some level of security measures against malicious users. This discussion overviews low-power context-aware FPAA architectures (Sect. 2) and then addresses FPAAs as physical computing devices for low-power embedded applications (Sect. 3). The conversation moves to secure FPAA devices (Sect. 4), showing positive FPAA security attributes (Sect. 4.1) and addressing FPAA security issues (Sect. 4.2). FPAA devices can be used to investigate security of analog/mixed-signal capabilities (Sect. 5), as well as be part of the resulting secure computation, such as implementing unique functions (Sect. 5.3). The final section summarizes the discussions as well as addresses remaining issues for secure ultra-low power embedded FPAA devices.

2 Low-Power Context-Aware FPAA Architectures

Many portable and wearable devices are constrained by their energy efficiency. Figure 2 illustrates the energy impact for cloud computation, on-device digital computation, and FPAA-assisted computation. The digital communication typically dominates the overall energy consumption [14].

Cloud-based computing removes issues of real-time embedded (e.g., fixed-point arithmetic) to be done on some far away (and supposedly free) server using MATLAB-style coding. Computation done off-device is not seen and considered effectively endless; eventually that resulting energy and resulting infrastructure required still have significant impacts. The host system still must constantly transmit and receive data through its wireless communication system to perform these computations. The network connectivity must have a minimum quality at all times; otherwise performance noticeably drops. One often assumes the cloud is nearly free for a small number of users; as the product scales to the consumer market, these assumptions can break down. Although the local digital device computation (for a good wireless network) requires similar energy for cloud and on-device computation (at a 100MMAC(/s) level), physical computation, such as FPAA-empowered devices, enables factors of 1000x improvement in the overall power requirements.

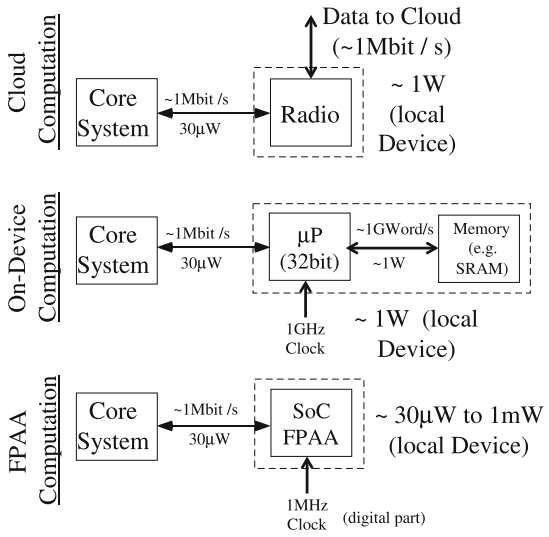


Fig. 2 Comparison of cloud computation, on-device computation, and FPAA computation. For cloud computation and for on-device computation, we only consider the energy required for communication. All devices might have an RF radio; we consider just the part required for this core computation. For FPAA computation we include the entire device. If cloud computation were considered free, then cloud and on-device computation would appear of similar complexity. FPAA computation dramatically decreases the resulting on-device computation

These approaches enable ultra-low power physical computing that enables small devices capable of computational-intensive (>10MMAC(/s)) context-aware processing. This approach requires low-power components, continuously operating or operating frequently, that can decide when to *wake up* the expensive hardware components. A node requiring $100\,\mu W$ average power could operate for several months on a single battery. These computing components are enabled by the x1000 energy improvement (and $\times 100$ area improvement) from FPAA classification algorithms. The always-on computation in stage one requires being physical computation, both because of its computational power and its close proximity to sensor inputs.

The need for low-average power consumption requires that higher-power devices, like wireless transceivers and even embedded μP, must be shut down most of the time. These devices should be active only in those rare cases where they are needed, such as when messages need to be passed between nodes. Similarly high-power sensors and actuators (e.g., acoustic speaker) need to be shut down except when it is being used. Without using physical computation, the sensor node would be a simple, low-speed data acquisition node and likely cannot stay under 1mW given the power constraints of the embedded processor and wireless transceiver. The rest of the processing probably still needs to take place on some other digital system.

Physical computing in context-aware architectures enables potential energy-harvesting opportunities. Most energy-harvesting devices supply $\approx 10\,\mu$W of power per cm^2 except in unusual environments. Figure 3b shows the device lifetime (due to average power consumption) for a single coin cell battery (0.1–0.5 Ah). A $10\,cm^2$ energy-harvesting device could supply $100\,\mu$W of average power, a manageable area for an embedded sensor node.

3 FPAAs as Physical Computation Devices

FPAA devices are our vehicle for discussing ultra-low energy computing. FPAA devices allow the user to investigate many physical computing designs within a few weeks of time. The alternative for one design would require years of IC design by potentially multiple individuals. These FPAAs compare favorably against custom designs, and unlike FPGA designs, FPAA architectures are open to the academic community.

Floating-gate (FG) devices empower FPAA by providing a ubiquitous, small, dense, nonvolatile memory element [19] (Fig. 4). A single device can store a weight value, compute signal(s) with that weight value, and program or adapt that weight value, all in a single device available in standard CMOS [17, 18]. The circuit components involve FG-programmed transconductance amplifiers and transistors (and similar components) with current sources programmable over six orders of magnitude in current (and therefore time constant) [24]. Devices not used are programmed to require virtually zero power. FG devices enable programming

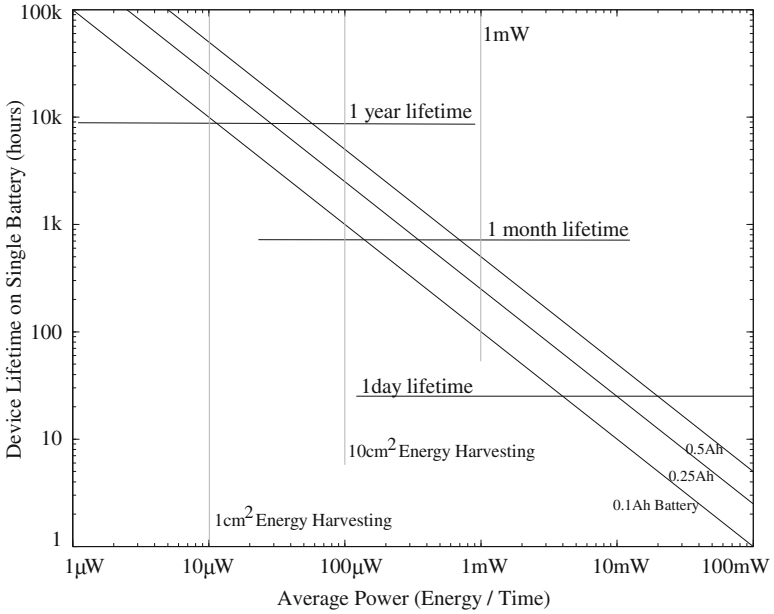


Fig. 3 Small sensor nodes require energy-efficient computation, computation enabled through physical computing approaches. Device lifetime available for wireless sensor nodes. The graph shows the opportunity for energy-harvesting systems at the size of 1 and 10 cm^2 form factors; most energy-harvesting systems output 10 μW of power per cm^2, with the exception of solar cells in direct sunlight in a desert

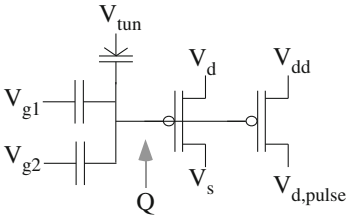


Fig. 4 Circuit diagram for basic FG device. The device can store a nonvolatile charge (Q), enables feedforward computation of functions involving Q, can program, and can adapt Q, all in a compact device structure. Multiple transistors sometimes ease the programming infrastructure for generic programming architectures

around device mismatch characteristics, enabling each device in a batch of ICs to perform similarly.

The SoC FPAA [11] ecosystem represents a device to system user-configurable system. An SoC FPAA implemented a command-word acoustic classifier utilizing hand-tuned weights demonstrating command-word recognition in less than 23 μW power utilizing standard digital interfaces (Fig. 5) [11]. Multiple analog signal processing functions are a factor of 1000× more efficient than digital processing,

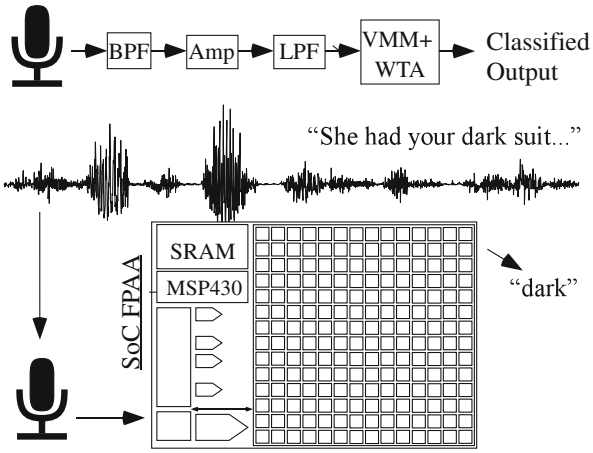


Fig. 5 High-level picture of low-power classification using an acoustic classifier for command-word classification. This approach enables computation from sensor to classified output in a single structure, handling all of the initial sensor processing and early-stage signal processing. The SoC FPAA device classified the word, such as *dark* from the TIMIT database phrases. This analog computation (<23 μW) is radically different than the class of expected analog operations

such as Vector-Matrix Multiplication (VMM), frequency decomposition, adaptive filtering, and classification (e.g., [11] and references within). Embedded classifiers have found initial success using this SoC FPAA device toward acoustic classification and learning (e.g., [11, 15]) in 10–30 μW average power consumption. The circuits compute from sensor to classified output in a single structure, handling all of the initial sensor processing and early-stage signal processing. This ecosystem will scale with newer ICs built to this standard, as expected by all future FPAA devices [20].

This new capability creates opportunities, but also creates design stress to address the resulting large co-design problem. The designer must choose the sensors as well as where to implement algorithms between the analog front end, analog signal processing blocks, classification (mixed-signal computation) which includes symbolic (e.g., digital) representations, digital computation blocks, and resulting μP computation. Moving heavy processing to analog computation tends to have less impact on signal line and substrate coupling to neighboring elements compared to digital systems, an issue often affecting the integration of analog components with mostly digital computing systems. Often the line between digital and analog computation is blurred, for example, for data converters or their more general concepts, analog classifier blocks that typically have digital outputs. The digital processor will be invaluable for bookkeeping functions, including interfacing, memory buffering, and related computations, as well as serial computations that are just better understood at the time of a particular design. Some heuristic concepts have been used previously, but far more research is required in building applications and the framework of these applications to enable co-design procedures in this space.

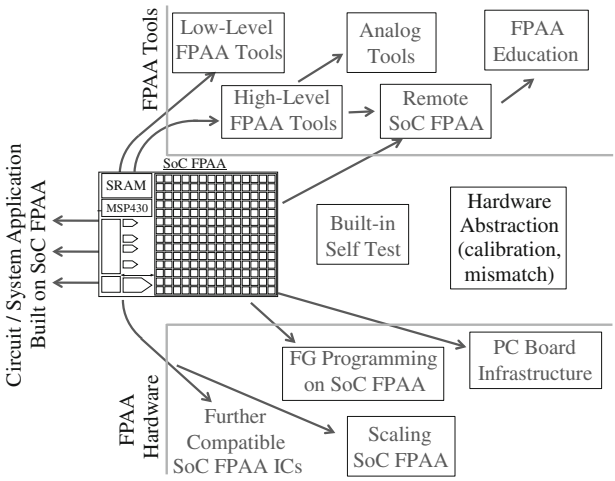


Fig. 6 SoC FPAA approach consists of key innovations in FPAA hardware, innovations and developments in FPAA tool structure, as well as innovations in the bridges between them. One typically focuses on what circuit and system applications can be built on the FPAA platform, but every solution is built up for a large number of components ideally abstracted away from the user

Analog computation [12] becomes relevant with the advent of FPAA devices, particularly the SoC FPAA devices [11]. Figure 6 shows a high-level view of the demonstrated infrastructure and tools for the SoC FPAA, from FG programming, device scaling, and PC board infrastructure through system-enabling technologies as calibration and built-in self-test methodologies and through high-level tools for design as well as education (e.g., [21]). This framework is essential for application-based system design using physical systems particularly given modern comfort with structured and automated digital design from code to working application. This framework utilizes abstractions in a mixed analog–digital framework [13], as well as development of high-level tools to enable non-device/analog circuit designers to effectively use these approaches [9] (Fig. 6). High-level design tools are implemented in Scilab/Xcos that enable automated compilation to working FPAA hardware [9]. These tools give the user the ability to create, model, and simulate analog and digital designs. Physical algorithms may show improvements beyond just energy efficiency for digital computing machines [12].

4 Embedded FPAA Security Concerns

The FPAA opportunities presented in the last section, particularly the ultra-low energy and small size characteristics, require consideration to make these embedded nodes secure. This section discusses multiple opportunities toward secure FPAA devices. Sections 4.1 and 4.1 discuss the positive characteristics and security issues,

in turn, for the FPAA device family. Sections 5.1 and 5.2 discuss the aspects of using FPAA devices to develop training and procedures for deconstructing custom ICs, giving a sense of the security of a compromised FPAA device. Section 5.3 discusses using the FPAA infrastructure to build a unique function for security, potentially in securing the given FPAA device.

4.1 Positive FPAA Security Attributes

The FPAA structure has a number of good security aspects. The FPAA uses FG devices to store the device state without any SRAM loading vulnerability, particularly from an external IC. Once the FG values on the chip are programmed and loaded, the FPAA code is secure, unless one can scan out the states of the FG elements. FG programming an IC will have minimal changes over the lifetime (e.g., 10-year rating) of the part. The programming code is not the IC μP SRAM, but only used for programming, and then purged after programming. Analog values can be hard to measure without disturbing the values significantly, and digital computation can be encoded with analog computation and storage. Further, very low-power circuits are challenging to externally measure due to the low-circuit currents (e.g., pA and nA). These transistors do not have enough current or field to generate light to measure transistor behavior and become very hard to measure the external fields.

On the other hand, the FPAA structure is a platform for creating secure applications. The SoC FPAA structure is a generic structure, openly published, and built from general components. None of the particular components are unknown or confidential. IC layout says almost nothing about the programmed IC functions. The motivation to *steal* the knowledge of on-chip FPAA circuits is minimal. The infrastructure can measure the analog behavior at any given node in the FPAA. FPAAs allow for scanning every hardware node internally to the circuit (e.g., [11, 33]). If the core FG programming on the IC is verified, effectively part of the calibration procedure and measurement [25], then the entire IC can be verified. Secure analog and digital code can be programmed in a secure space.

The IC could have intelligence, using internal signals and voltages, to choose to erase its contents. If tampering is suspected, the operating device could pull up on the tunneling voltage line(s) in an attempt to erase the previous operating code. The device parallel erase occurs from a combination of electron tunneling and reverse tunneling. The result leaves little chance of recovering any previous code even with a short erase cycle. One is more likely to pick up device mismatch patterns rather than anything of the previous code.

4.2 Addressing FPAA Security Issues

FPAA devices are far from safe from a potential malicious agent, even with a number of good starting properties. For example, the current FPAA devices do not have encryption and related security on the input control of the device. If an actor could connect to the particular control connections, even if the IC pins are disconnected or disabled, they could get direct control of the device and programming infrastructure. Future FPAA devices will have encryption on the control structure, particularly as they move to a wider user community. The encrypted access can make use of a PUF from the particular FPAA, such as the approach shown in Sect. 5.3. Encryption is a straightforward solution used on secure FPGA devices. This section will consider the resulting issues for these devices.

Figure 7 illustrates possible security issues and types of attacks for an embedded system built with SoC FPAA device. The FPAA attacks could happen by physical tampering with an existing device, as well as electronic attacks through the communication port, such as a transceiver port. In a physical FPAA attack, the device is obtained while avoiding self-destruct sequence to be explicitly deconstructed. If the internal code can be obtained, likely at considerable expense, one could potentially reconstruct the FPAA function. Mismatch encoded functions would require additional computational and measurement structures. An alternate physical FPAA attack could use a compiled digital serial port to gain access to the digital control and resulting programming interface. When digital interfaces (e.g., SPI) are controlled by the processor, getting control of the processor is unlikely. A more likely situation is finding a way to stall the computation resulting from a physical attack on the clock structure. Many systems are far less secure due to physical tampering if the device has been obtained, and any self-destruct/erase mechanism was somehow avoided. A more likely situation is a nonphysical attack through the transceiver interface into the IC. These can include attacks to gain control of the FPAA device to reprogram the device or constantly attacking a device to drain the node battery power.

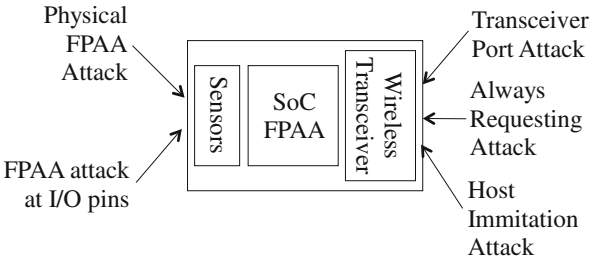


Fig. 7 Possible security issues for an embedded system built with SoC FPAA device. Some attacks could occur through the known communication path, such as through the wireless transceiver port, and other attaches could occur through direct physical access to the device

Table 1 Summary classification of IoT systems

Category	RAM	ROM
Class 2 (C_2)	50 kB	250 kB
Class 1 (C_1)	10 kB	100 kB
Class 0 (C_0)	<10 kB	<100 kB

Low-energy computation opens application opportunities at 10 mW, 1 mW, and lower-average power consumption, and yet the low power consumption constrains the system security capabilities. Embedded FPAA applications have limited digital memory because of the system cost. Network security is characterized in terms of classes of networked devices, summarized in Table 1 [6, 22, 26, 29, 31]. SoC FPAA is a C_0 device having only 32 kB total digital memory. Digital memory is expensive in terms of relative on-chip area, complexity, and energy dissipation. Many systems going forward might have less total digital memory, as well as many systems that will not rise to the C_1 memory level. FPAAs enable a whole opportunity of C_0 devices, devices many assume are impossible to secure over a network. Running a minimal OS and security code may exceed the rest of system energy budget.

So how do we have an ultra-low power secure IoT system? Part of the opportunity is coding systems outside of a minimal OS, consistent with the rest of the event-based FPAA μP code, as well as enabling tight secure stack and security aspects in MSP 430 assembly language. Digital FPAA event code is coded in assembly language and encapsulated in graphical code for easy user reuse.

5 FPAAs for Investigating IC Validation

When a user has a programmed FPAA device, it looks like any other custom IC that performs one or a set of functions. Further, the IC layout says nothing about the actual device performance. If the user knows it is an FPAA device and has sufficient knowledge of its programming functions, they might have additional information to figure out the function; otherwise, all they have is the device to characterize.

FPAA devices become good test platforms to investigate how individuals might deconstruct a particular IC. FPAA devices allow for many reprogrammed circuits, so the approach can be repeated many times. In the following subsections, we will discuss two such cases. First we will overview the inspiration of this study, the Black Box (BB) exam at Caltech (CNS 182). Second, we will discuss how this approach was modified, in an academic setting enabled by FPAA devices, to Training IC Deconstruction.

5.1 Black Box (BB) Exam: CNS 182

Academic groups are not usually interested in deconstructing known ICs, and when they are, the details are rarely discussed. One particular exception I personally experienced, both as a student (1993) and a teacher (1994–1996), was the Black Box (BB) exam at Caltech (CNS 182). This particular exercise was the final exam for the second quarter (Winter quarter) for CNS 182, Analog VLSI, and Neural Systems, between 1989 and 1996.

The exam consisted of a 2-h lab session followed by several days (4–5) to write up the results discovered during the lab session. The students in the class spent every week for two quarters measuring custom-built ICs, starting with transistors through small systems, using typical computer-controlled bench equipment. When the students arrived in the lab for the BB exam, a particular circuit consisting of 3–5 pins (besides power (V_{dd}) and ground (GND)) was operating correctly in one possible mode. Typically the circuit was a single transconductance amplifier (TA) or 2 TA circuit with a couple of transistors and known to be somewhat related to course topics over the first two quarters. This circuit was part of a 40-pin chip custom fabricated for the course; the students did not have access to any layout information. At least one element was a bias, set by a potentiometer. No FG devices were used. In the end, roughly half of the students would correctly guess the correct circuit with various levels of experimental justification.

5.2 Training IC Deconstruction Using FPAA BB Approach

The BB experience was recreated between 2011 and 2012 using currently available FPAA devices. The FPAA enables investigating deconstructing circuits, by providing a structured platform to instantiate a large number of circuits and systems. Each case would look from the IC pins to be some custom IC device and could be tested accordingly. The deconstruction capabilities can be quantified for different amounts of IC knowledge, such as routing information or netlists. These techniques could be used to verify a desired circuit implementation, as well as search for any additional component that was placed in the circuit. The FPAAs used for these experiments were designed between 2007 and 2010 (e.g., [4, 33]); the results should directly extend to using the SoC FPAA devices.

A group of graduate student IC designers were trained through a set of six BB events (Table 2) over a 9-month timeframe to eventually deconstruct a custom-fabricated IC. This BB approach arose from the constant interaction between courses and research. One person designed, compiled, and experimentally characterized the design completely without the knowledge of others. The groups had no idea of the functionality of the circuits before they arrived in the lab. Each person on the student team was previously familiar with measuring the FPAA devices. Between events students developed additional tools to assist in deconstructing the IC design.

Table 2 Summary of FPAA black box experiments

	Components to find	Group info	Analysis techniques	Teams	Time
BB1	Analog Amps/muxes	Only IC	DC I/V, scopes	3 (2 people)	>8 h
BB2	Am Demod + hidden	Switch list	Switch list analysis + DC I/V, scopes	2 (3 people)	≈ 4 h
BB3	DAC: 5ibt R-22R + 3-bit V-mode = 8 bits	Netlist ≈100	Low-level netlists + DC I/V, scopes	2 (3 people)	7–8 h
BB4	Low-frequency transciever circuit	Netlist (spice)	Netlists, clustering, + DC I/V, scopes	2(4 people)	6–8 h
BB5	VCO controlled by 7-bit DAC	Netlist (spice)	Netlists, clustering, + DC I/V, scopes	2 (3 people)	5–6 h
BB6	Multiplexed 1 8bit DAC two in, two out	Netlist (spice)	Netlists, clustering, + DC I/V, scopes	2 (3 people)	4–5 h

Different events had different level of information (Table 2). The first case paralleled the Caltech experience to get a baseline performance, but with roughly double the number of chip pins and number of components, as well as the students involved did not prepare before this starting exercise. The groups did a number of I–V measurements at the chip pins to identify the resulting circuit. In the second case, the groups had a switch list (Fig. 8b), similar in format to the SoC FPAA approach [24]. The group made extensive use of the routing visualization tool, Routing Activity Tool (RAT), to uncover the resulting circuits. Whiteboard pictures prove this solution approach. Figure 8b shows the expected demodulation circuit which all groups found; the groups also found an unexpected extra oscillator that was explicitly added. In later cases the groups were given a form of netlist, compatible with the existing tools, for their analysis. All of the groups developed clustering algorithms to assist with grouping and identifying the resulting circuits. At each level, the speed to fully recognize and experimentally verify a particular circuit increased with the increasing circuit complexity.

The final goal was to extract and verify an entire custom IC developed by another group. A group of four Ph.D. students is involved in the BB experiences, and Dr. Hasler would spend three isolated days together to analyze this IC. Although the promised information varied throughout, in the end, the group was given (approximate) delayered information extracted from the IC, not including n-type or p-type selections. After 3 days and 2 additional days to write the report, the group found all four interleaved DACs, although only one was populated fully. The group discovered an error on the VCO due to a misplaced GND line.

This process showed FPAAs could be used to train individuals to deconstruct the circuitry on a particular device, as well as important insights to secure a particular FPAA device. Nonvolatile analog FG storage makes discovering the internal code of a programmed device extremely difficult without huge expenses. The approach showed some unique aspects of using physical computation related to security; the

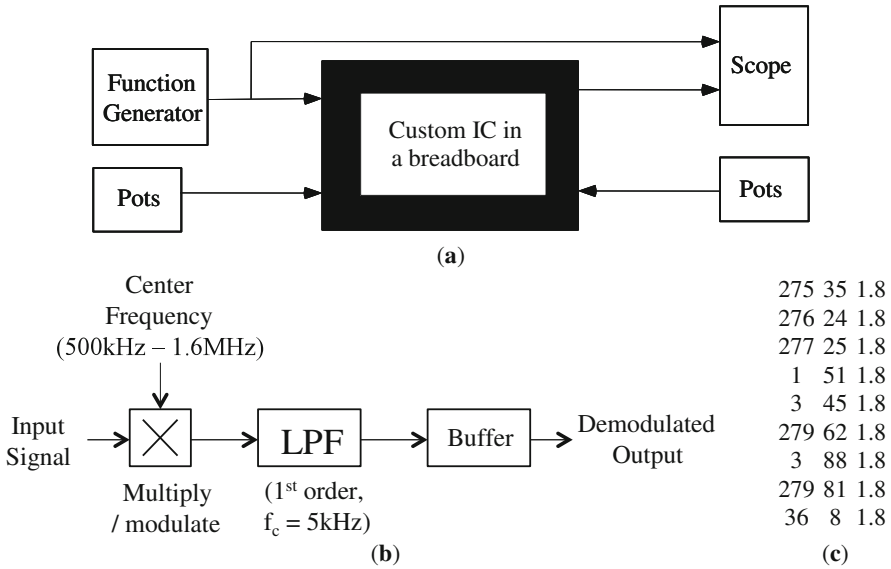


Fig. 8 Illustration of the Black Box exam setup. (**a**) A student would arrive into the lab with a working device demonstrating some characteristic of the circuit. The question is to find the entire circuit, a circuit inside an integrated circuit with a few, e.g., I/O pins, in a finite amount of time (e.g., 2 h). This experience has parallels to security issues when deconstructing an unknown analog or mixed-mode circuit. (**b**) A low-frequency signal demodulator is an example system (BB2) to deconstruct from the available data. Each of these components was built using available CAB components and routed into the FPAA infrastructure. Typical electrical engineers might predict such an architecture when faced with multiple components. If an additional component is sitting in this circuit, it might create confusion or might just be overlooked. (**c**) In BB2, the groups had the switch list programmed into the FPAA device. The switch list communicates the physical routing on the FPAA IC. The first two columns are position in *x* and *y* direction on chip. The third column is log-encoded value for current level; 1.8 is a value to program as a switch

wider opportunities in physical computing [12] show these items are just scratching the surface of what is possible.

5.3 FPAAs for Unique Functions

Unique functions in FPAA IC devices are rich platforms to construct unique functions, particularly for security. The FPAA device allows for the selection of many devices, devices that have mismatch specific to a particular IC and mismatch that can be selected and compiled into a particular circuit. The mismatch between pFETs for a FG device enables almost 1M mismatched components.

Unique functions and PUFs have been implemented in FPGAs (digital) [28, 37] and analog circuits [8, 32]. For example, [37] uses delay variability in the FPGA

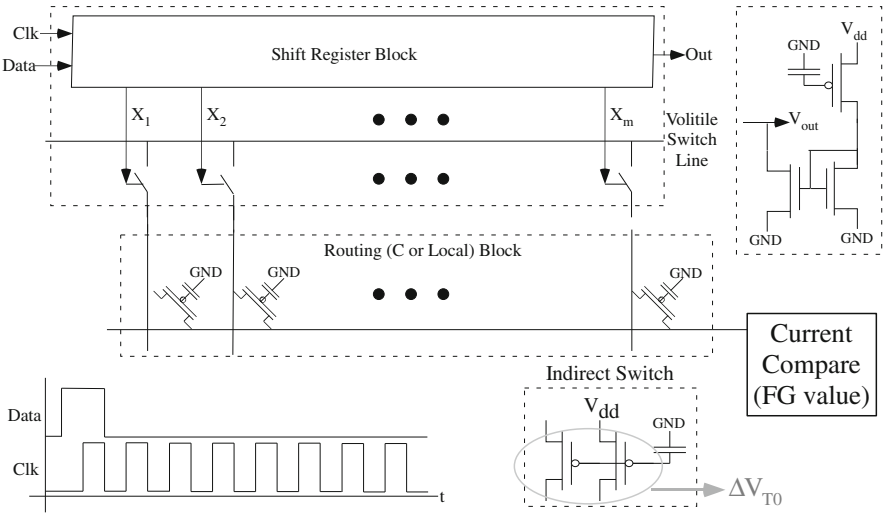


Fig. 9 Example of generating unique functions for secure codes implemented in an SoC FPAA. Threshold voltage (ΔV_{T0}) mismatch at a chosen location in nearly one million FG devices gives the resulting code

to create a specific code directly affected by the component variability. All of these functions are based on the mismatch of the resulting device, whether custom fabricated or compiled in the structure [36]. This FPAA approach is similar to the FPGA approach for making unique functions and PUF, in that a function is compiled on the device and utilized to create a unique output code for a particular input stimulus code.

Figure 9 shows an example of FPAA circuit for generating a unique function [16]. This approach utilizes the mismatch available in the FPAA circuit, mismatch we typically remove from the device. The structure yields a code for encryption of data, enabled by programming the desired code by the user. One use for the input code (stimulation) is the address of the FG elements to measure. The resulting outputs, scanned through shift registers available throughout the IC, would be thresholded to yield a digital code (Fig. 10). The FG elements would be programmed to bias the resulting code as desired, modulating the mismatch pattern. Typically one would program all elements to the same current to bring out the mismatch pattern (e.g., [11]). The programmed values would be retained for the operation of the FPAA IC, showing μV shift over a typical 10-year lifetime. The function could be compiled right into the rest of the circuitry, where implementation and routing of other circuits would obfuscate the resulting devices. This technique allows for an evolution of the codes through secure FG updates. If a code was suspected to be discovered, one could easily just move the sensing circuitry to an open circuit area. This unique function circuit may not have to be on the chip, but can be compiled onto a particular IC when needed [16]. If the IC is erased, knowledge of the PUF is erased except in the secure space originally used.

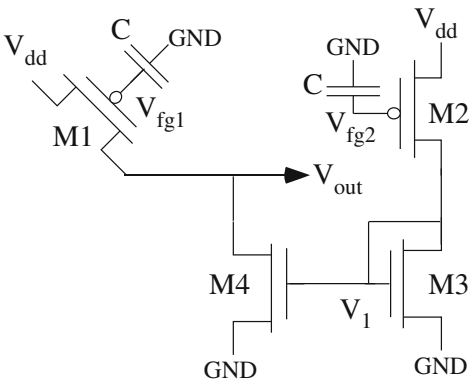


Fig. 10 Effective circuit diagram for the PVT analysis of the unique function circuit. The volatile switch line set to Vdd for this circuit

6 Summary and Next Directions

Physical computing opens great opportunities in energy-constrained IoT environments while creating significant security challenges for these IoT devices. FPAA devices enable the large-scale deployment of physical computation, and yet, these FPAA-enabled nodes often require secure operation against malicious users. Low-power context-aware FPAA architectures enable a number of autonomous sensor nodes. FPAA devices have a number of positive security attributes and security issues. FPAA devices can be used to investigate security and be part of the resulting secure computation.

We want to summarize current issues for building and deploying secure ultra-low power embedded FPAA devices. These directions include:

- Encrypt the control (and therefore programming) data stream, likely using a PUF circuit for the encryption code as part of the FPAA IC.
- Develop ultra-small security framework in dedicated assembly code + mixed-signal classification that integrates with event-based μP operation.

Network traffic attacks on FPAA-based systems are likely to be a point of vulnerability, requiring building tables and metrics of proper and improper network activity and classifying the resulting responses [1–3, 5, 7, 10, 23, 30, 34, 35]. These functions must be done in as low computational energy as possible. The functions require a minimal digital energy in parsing and creating these tables. Classification energy would be minimized using learning classifiers compiled on the FPAA infrastructure [15].

Security for ultra-low power embedded computing platforms based on FPAA devices is possible and is a space rich in potential research opportunities. The need for secure ultra-low power embedded computing platforms will likely only grow in the near future.

Acknowledgements This chapter resulted from a conference on IoT and embedded network device security in Paris in July 2017. My first exposure to security of embedded network devices started in Paris, not too far from the conference in 2017, when I (J. Hasler) visited Francois Bayen and his family in Paris in 1991 as a young graduate student. We had a number of technical discussions on embedded integrated circuit (IC) security related to a number of his European projects. These discussions gave me the background and interest to dig deeper in these areas when the time came; I am thankful for everything I learned from him during that visit and later visits. Twenty-six years later these discussions have come full circle. Writing this chapter on security of embedded devices, particularly FPAA devices, has brought me back to the knowledge source. Their family has remained close friends with my family to this day.

References

1. V. Asharani, B.N. Veerappa, M. Rafi, Security evaluation of pattern classifiers in adversarial environments. Int. J. Comput. Sci. Mobile Comput. **4**(4), 768–774 (2015)
2. T. Bakhshi, B. Ghita, On internet traffic classification: a two-phased machine learning approach. J Comput. Netw. Commun. 1–21 (2016)
3. R. Bar-Yanai, M. Langberg, D. Peleg, L. Roditty, Realtime classification for encrypted traffic, in *SEA* (2010), pp. 373–385
4. A. Basu, S. Brink, C. Schlottmann, S. Ramakrishnan, C. Petre, S. Koziol, F. Baskaya, C. Twigg, P. Hasler, A floating-gate-based field-programmable analog array. IEEE J. Solid-State Circuits **45**(9), 1781–1794 (2010)
5. B. Biggio, G. Fumera, F. Roli, Security evaluation of pattern classifiers under attack. IEEE Trans. Knowl. Data Eng. **26**(4), 984–996 (2014)
6. C. Bormann, M. Ersue, A. Ericsson, Terminology for constrained-node networks, in *RFC 7228*. Internet Engineering Task Force (IETF) (2014)
7. A.A. Cardenas, J.S. Baras, Evaluation of classifiers: practical considerations for security applications (2006)
8. R. Chakraborty, C. Lamech D. Acharyya, J. Plusquellic, A transmission gate physical unclonable function and on-chip voltage- to-digital conversion technique, in *IEEE DAC* (2013), pp. 59:1–59:10
9. M. Collins, J. Hasler, S. George, An open-source toolset enabling analog–digital software codesign. J. Low Power Electron. Appl. **6**(1), 3 (2016)
10. A. Dainotti, A. Pescape, K.C. Claffy, Issues and future directions in traffic classification. IEEE Netw. **26**(1), 35–40 (2012)
11. S. George, S. Kim, S. Shah, J. Hasler, M. Collins, F. Adil, R, Wunderlich, S. Nease, S. Ramakrishnan, A programmable and configurable mixed-mode FPAA SoC. IEEE Trans. Very Large Scale Integr. **24**(6), 2253–2261 (2016)
12. J. Hasler, Opportunities in physical computing driven by analog realization, in *IEEE ICRC, San Diego* (2016)
13. J. Hasler, Analog Abstraction, Computation, and Numerical Analysis. Accepted to ISCAS 2018, Florance, May 2018
14. J. Hasler, B. Marr, Finding a roadmap to achieve large neuromorphic hardware systems. Front. Neuromorphic Eng. (2013), 1–29. https://doi.org/10.3389/fnins.2013.00118
15. J. Hasler, S. Shah, SoC FPAA hardware implementation of a VMM+WTA embedded learning classifier, in *IEEE ECAS* (2018)
16. J. Hasler, S. Shah, Security implications for ultra-low power configurable SoC FPAA embedded systems. J. Low-Power Electron. Appl. **8**(2), 1–18 (2018)
17. P. Hasler, C. Diorio, B.A. Minch, C.A. Mead, Single transistor learning synapses, in *Advances in Neural Information Processing Systems*, ed. by G. Tesauro, D.S. Touretzky, T.K. Leen, vol. 7 (MIT Press, Cambridge, 1994), pp. 817–824

18. P. Hasler, C. Diorio, B. Minch, C. Mead, Single transistor learning synapse with long term storage. IEEE Int. Symp. Circuits Syst. **3**, 1660–1663 (1995)
19. P. Hasler, B. Minch, C. Diorio, Adaptive circuits using pFET floating-gate devices, in *Advanced Research in VLSI* (1999), pp. 215–229
20. J. Hasler, S. Kim, F. Adil, Scaling floating-gate devices predicting behavior for programmable and configurable circuits and systems. J. Low Power Electron. Appl. **6**(3), 13 (2016)
21. J. Hasler, A. Natarajan, S. Shah, S. Kim, SoC FPAA immersed junior level circuits course, in *MSE* (2017)
22. T. Heer, O. Garcia-Morchon, R. Hummen, S.L. Keoh, S.S. Kumar, K. Wehrle, Security challenges in the IP-based internet of things? Wirel. Pers. Commun. **61**(3), 527–542 (2011)
23. Z. Khorshidpour, S. Hashemi, A. Hamzeh, Learning a secure classifier against evasion attack, in *IEEE Data Mining Workshops* (2016), pp. 295–302
24. S. Kim, J. Hasler, S. George, Integrated floating-gate programming environment for system-level ICs. EEE Trans. Very Large Scale Integr. **24**(6), 2244–2252 (2016)
25. S. Kim, S. Shah, J. Hasler, Calibration of floating-gate SoC FPAA system. IEEE Trans. Very Large Scale Integr. (2017)
26. J. King, A.I. Awad, A distributed security mechanism for resource-constrained IoT devices. Informatica **40**, 133–143 (2016)
27. C. Mead, Neuromorphic electronic systems. Proc. IEEE **78**, 1629–1636 (1990)
28. N. Muthumeenakshi, S. Sharma, M. Farjanaaameera, R. Rajaprabha, LSI design of low cost (PUF) physical unclonable function using FPGA and highly secured clock network. IOSR J. VLSI Signal Process. **4**(1), 16–21 (2014)
29. G. Oikonomou, I. Phillips, Experiences from porting the Contiki operating system to a popular hardware platform, in *IEEE DCOSS Workshop, Barcelona* (2011)
30. T. Omrani, A. Dallali, B.C. Rhaimi, J. Fattahi, Fusion of ANN and SVM classifiers for network attack detection, in *Sciences and Techniques of Automatic Control* (2018)
31. R. Roman, P. Najera, J. Lopez, Securing the Internet of Things. IEEE Comput. **44**(9), 51–58 (2011)
32. T. Saha, V. Sehwag, TV-PUF : a fast lightweight analog physical unclonable function, in *IEEE Nanoelectric and Information Systems* (2016)
33. C. Schlottmann, S. Shapero, S. Nease, P. Hasler, A digitally enhanced dynamically reconfigurable analog platform for low-power signal processing. IEEE J. Solid-State Circuits **47**(9), 2174–2184 (2012)
34. P. Velan, M. Cermak, P. Celeda, M. Drasar, A survey of methods for encrypted traffic classification and analysis. Int. J. Netw. Manage. **25**(5), 355–374 (2014)
35. Z. Wang, The applications of deep learning on traffic identification
36. Z. Wang, Y. Chen, A. Patil, J. Jayabalan, X. Zhang, C.H. Chang, A. Basu, Current mirror array: a novel circuit topology for combining physical unclonable function and machine learning, in *IEEE TCAS I* (2017)
37. T. Xu, M. Potkonjak, Robust and flexible FPGA-based digital PUF, in *FPL* (2014)

Challenges in Cyber Security: Ransomware Phenomenon

Vlad-Raul Paşca and Emil Simion

Abstract Ransomware has become one of the major threats nowadays due to its huge impact and increased rate of infections around the world. According to https://www.adaware.com/blog/cryptowall-ransomware-cost-users-325-million-in-2015, just one family, CryptoWall 3, was responsible for damages of over 325 millions of dollars, since its discovery in 2015. Recently, another family of ransomware appeared in the cyberspace which is called WannaCry, and according to https://www.cnet.com/news/wannacry-wannacrypt-uiwix-ransomware-everything-you-need-to-know, over 230,000 computers around the world, in over 150 countries, were infected. This type of ransomware exploited a vulnerability which is present in the Microsoft Windows operating systems called EternalBlue, an exploit which was developed by the US National Security Agency (NSA) and released by The Shadow Brokers on April 14, 2017.

Spora ransomware is a major player in the field of ransomware families and is prepared by professionals. It has the ability to encrypt files offline like other families of ransomware, DMA Locker 3.0, Cerber, or some editions of Locky. Currently, there is no decryptor available in the market for the Spora ransomware.

Spora is distributed using phishing e-mails and infected websites which drops malicious payloads. There are some distribution methods which are presented in http://malware-traffic-analysis.net/2017/02/14/index2.html (the campaign from February 14, 2017) and http://malware-traffic-analysis.net/2017/03/06/index.html (the campaign from March 6, 2017).

Once the infection has begun, Spora runs silently and encrypts files with a specific extension, not all extensions are encrypted. This type of ransomware is interested in office documents, PDF documents, Corel Draw documents, database files, images, and archives and is important to present the entire list of extension in order to warn people about this type of attack: xls, doc, xlsx, docx, rtf, odt, pdf, psd, dwg, cdr, cd, mdb, 1cd, dbf, sqlite, accdb, jpg, jpeg, tiff, zip, rar, 7z, backup, sql,

V.-R. Paşca (✉) · E. Simion
Faculty of Automatic Control and Computers, University POLITEHNICA of Bucharest, Bucharest, Romania
e-mail: emil.simion@upb.ro

Ç. K. Koç (ed.), *Cyber-Physical Systems Security*,
https://doi.org/10.1007/978-3-319-98935-8_15

and bak. One crucial point here is that everybody can rename the files in order to avoid such infections, but the mandatory requirement is to back up the data.

Spora doesn't add extensions to the encrypted files, which is really unusual in the case of ransomware, for example, Locky adds .locky extension, TeslaCrypt adds .aaa extension, and WannaCry appends .WNCRY extension. In this case, each file is encrypted with a separate key, and it is a nondeterministic encryption (files with an identical content are encrypted in different ciphertexts); the content which was encrypted has a high entropy and visualization of an encrypted file, which suggests that a stream cipher or chained block was used (AES in CBC mode is suggested, because of the popularity of this mode of operation in ransomware's encryption schemes).

There are some methods which are used frequently to assure that a single copy of a malware is running, for example, the creation of a mutex, which means that the encrypted data is not encrypted again; therefore, we have a single step of encryption. Of course, there are some folders which are excluded from encryption, because the system must remain in a working state in order to make a payment, so Spora doesn't encrypt the files which are located in the following directories: windows, program files, program files (x86), and games.

Spora uses Windows Crypto API for the whole encryption process. Firstly the malware comes with a hardcoded AES 256 key, which is being imported using CryptImportKey (the parameters which are passed to this function reveal that an AES 256 key is present). The AES key is further used to decrypt another key, which is a RSA public key, using a CryptDecrypt function (a ransom note is also decrypted using the AES key, as well as a hardcoded ID of the sample).

For every computer, Spora creates a new pair of RSA keys. This process uses the function CryptGenKey with some parameters which are specific for RSA keys, after that the private key from the pair is exported using the function CryptExportKey and Base64 encoded using the function CryptBinaryToString. A new AES 256 key is generated using CryptGenKey, is exported using CryptExportKey, and is used to encrypt the generated private RSA key (finally, the key is encrypted using the hardcoded RSA public key and stored in the ransom note). For every file a new AES key is generated which is used to encrypt the file, is encrypted using the generated public RSA key, and is stored at the end of every encrypted file.

Spora is a professional product created by skilled attackers, but the code is not obfuscated or packed, which makes the analysis a little bit easier. The implementation of cryptographic algorithms uses the Windows Crypto API and seems to be consistent; nonetheless the decryption of files is not really possible without paying the ransom. The ability to handle a complex process of encryption offline makes Spora ransomware a real danger for unprepared clients.

Ransomware usually uses the RSA algorithm to protect the encryption key and AES for encrypting the files. If these algorithms are correctly implemented, then it is impossible to recover the encrypted information.

Some attacks, nonetheless, work against the implementation of RSA. These attacks are not against the basic algorithm, but against the protocol. Examples of such attacks on RSA are chosen-ciphertext attack, common modulus attack, low

encryption exponent attack, low decryption exponent attack, attack on encryption and signing with the same pair of keys, and attack in case of small difference between prime numbers p and q.

The attacks on AES implementation include ECB attack, CBC implementation without HMAC verification and oracle padding attack.

In the following sections, we present the fully analysis on three representative ransomware: Spora, DMA Locker, and WannaCry.

1 Spora Ransomware

Name: 9ae49d4a4202b14efe.exe
md5: 116d339b412cd1baf48bcc8e4124a20b
Type: encrypting ransomware

In Fig. 1 a detection report by VirusTotal scanner mechanism is presented, which shows that the malware is known and most vendors already offer a protection mechanism for it. Figure 2 shows us that the malware itself is not packed; nonetheless later results will show that the malware is obfuscated and hence the complexity of the analysis grows.

Figure 3 shows a string which is pushed on the stack 699 times; this trick is used to obfuscate the code.

In Fig. 4 it is shown that a function is called 700 times (the function calls **OpenMutexA**, which tries to open an existing mutex), which doesn't make sense

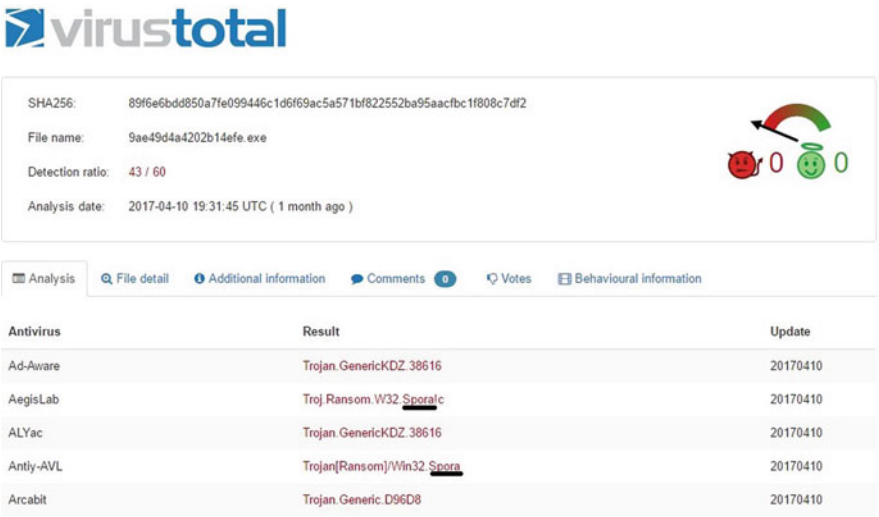


Fig. 1 VirusTotal report

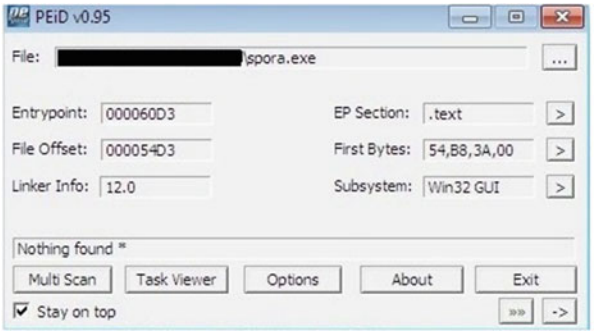


Fig. 2 PEiD report

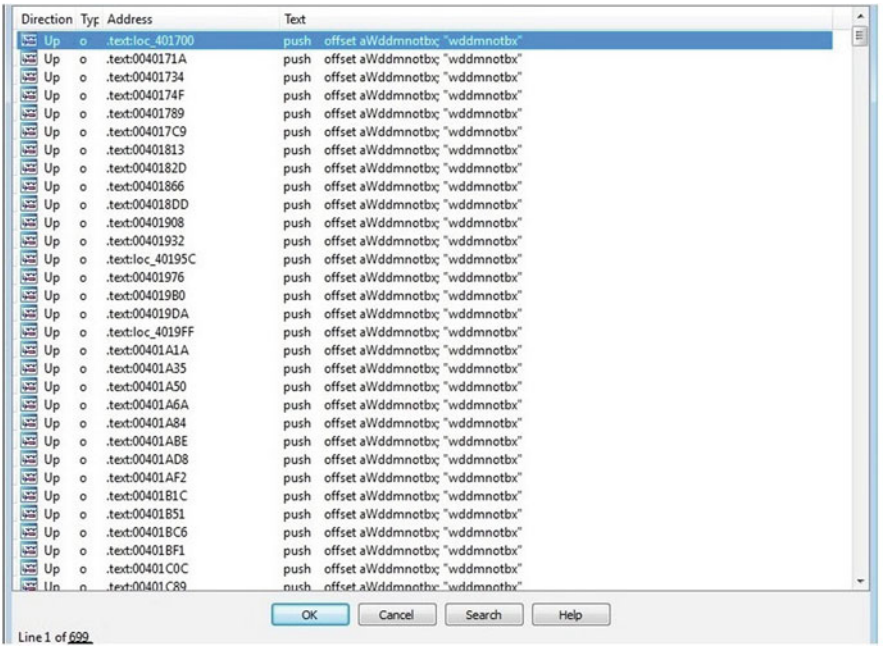


Fig. 3 IDA Pro 1

in this case, because the malware doesn't call **CreateMutexA**; this is another trick used to complicate the analysis. The malware uses the function **VirtualAlloc** to allocate space in the process address space, and then it writes the actual payload in that space. The initial conclusion is that the initial executable is just a packer and the actual malicious code is contained in the newly executable, which has the md5 97e84cc8afca475d15d8c3e1f38d deba.

The malware calls **GetVolumeInformationW** to get information about the file system and volume associated with the root directory, as shown in Fig. 5.

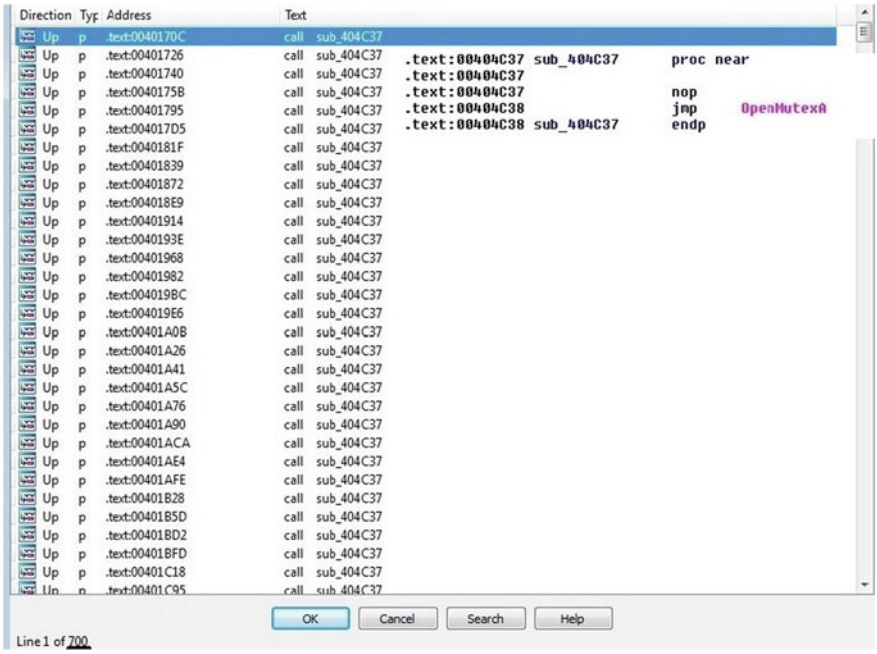


Fig. 4 IDA Pro 2

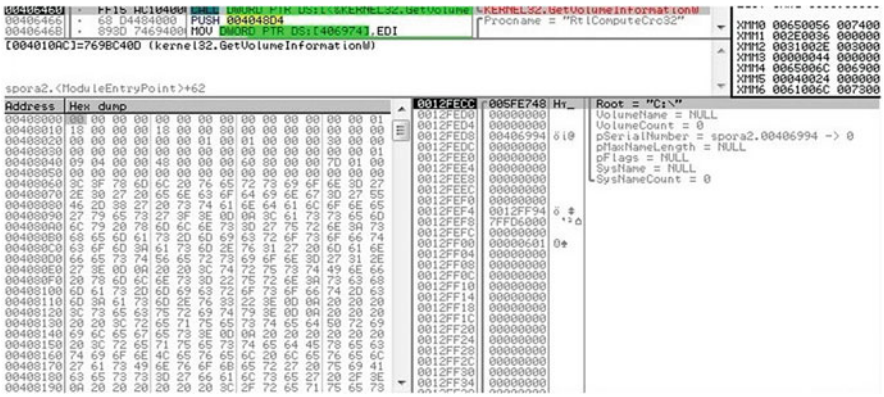


Fig. 5 GetVolumeInformationW call

A mutex is created and it has the following format: m⟨GetVolumeInformationResult⟩ (in decimal), to ensure that the malware runs only once. The sample creates a file which has the following name: C:\Users\⟨*user*⟩\AppData\Roaming\⟨*Mutex*⟩. The malware comes with a hardcoded key, which is being imported using the function **CryptImportKey**, as shown in Fig. 6. It represents an AES256 key, stored in a form of a blob. The

Fig. 6 CryptImportKey call

explanation of the fields is: 08 represents PLAINTEXTKEYBLOB and means that the key is a session key; 02 CUR_BLOB_VERSION, 0x00006610 which represents Alg_ID: CALG_AES_256, 0x20=32 represents key length.

The AES key is used to decrypt another key, which is a RSA key embedded in the binary, as shown in Fig. 7. The AES key is also used to decrypt the ransom note and the binary's hardcoded ID. The malware uses **GetLogicalDrives** to obtain the currently available disk drives and then a loop, which selects the files that have a specified extension which is attacked by this ransomware, is created. The malware also uses **WNetOpenEnum** and **WNetEnumResource** APIs to enumerate the network resources, and the created file is used to store temporary data, like the files which will be encrypted.

Fig. 7 CryptDecrypt calls

The attacked extensions are presented in the table below:

.xls	.doc	.xlsx	.docx	.rtf	.odt	.pdf	.ppt	.pptx
.psd	.dwg	.cdr	.cd	.mdb	.1cd	.dbf	.sqlite	.accdb
.jpg	.jpeg	.tiff	.zip	.rar	.7z	.backup	.sql	.bak

The next folders are excluded from the attack:

windows	programfiles	programfiles(x86)	games

For every victim, the malware creates a pair of RSA keys. The fragment which generates the RSA key pair (1024 bits) is shown in Fig. 8.

The relevant parameters for **CryptGenKey** are 0xA400 which represents AlgId: CALG_RSA_KEYX and 0x04000001 which represents RSA1024BIT_KEY | CRYPT_EXPORTABLE. The private RSA key is exported and Base64 encoded, as shown in Fig. 9. The encryption of the private RSA key is stored into a buffer alongside the data regarding the machine and the infection, like date, username, country code, malware ID, and statistics of encrypted file types. An example is shown in Fig. 10. The malware uses a MD5 algorithm to hash the buffer which contains the private RSA key (the hash is used to create the user ID) as shown in Fig. 11. Another AES key is generated; then it's exported and encrypted using public RSA key that was hardcoded. In Fig. 12 this process is shown. The generated AES key is used to encrypt the data (including the RSA private key), as shown in Fig. 13. Finally, all encrypted data is Base64 encoded and stored in the ransom note.

Fig. 8 CryptGenKey call

Fig. 9 RSA key is Base64 encoded

For every file a new AES256 key is generated, as shown in Fig. 14. The AES key is encrypted using the generated public RSA key, and it is appended to the encrypted file; also the CRC32 is being computed and stored in the file (Fig. 15). Each file is encrypted using the AES key, as shown in Fig. 16.

In order to decrypt a file, a ransom note is uploaded to the server giving the attacker access to all information needed. He uses the private RSA key corresponding to the hardcoded public RSA key to decrypt the first AES key, and then the key is used to decrypt the generated private RSA key. Because of the fact that each AES256 key is encrypted using the corresponding public RSA key and stored at the end of each file, it is possible to decrypt each key and then decrypt each file individually.

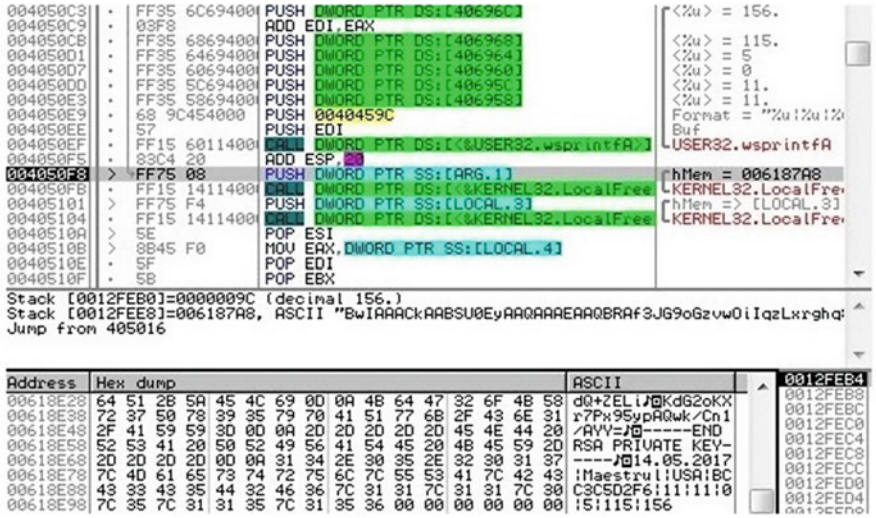


Fig. 10 Buffer contains information about the system

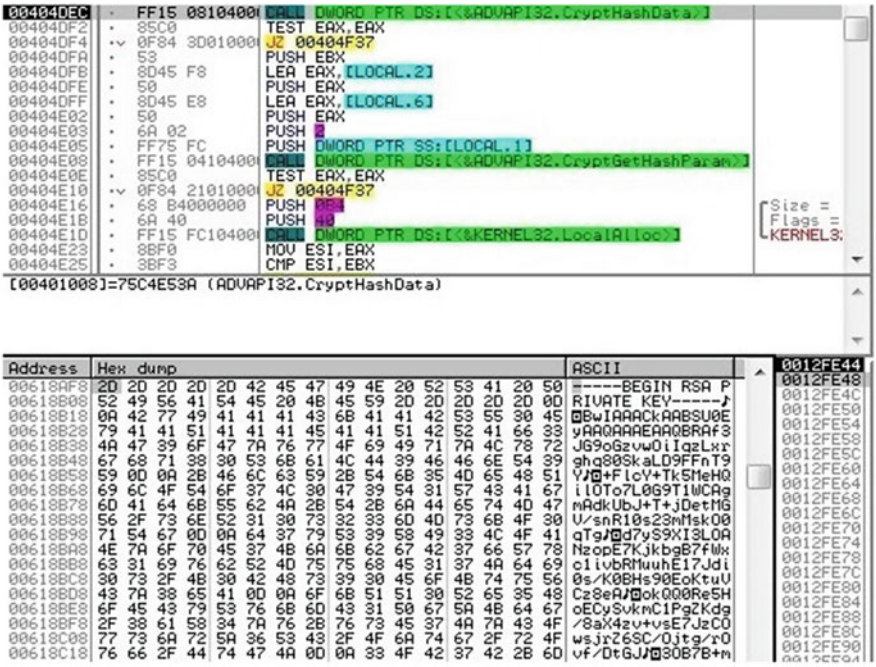


Fig. 11 MD5 Algorithm is used to hash the buffer

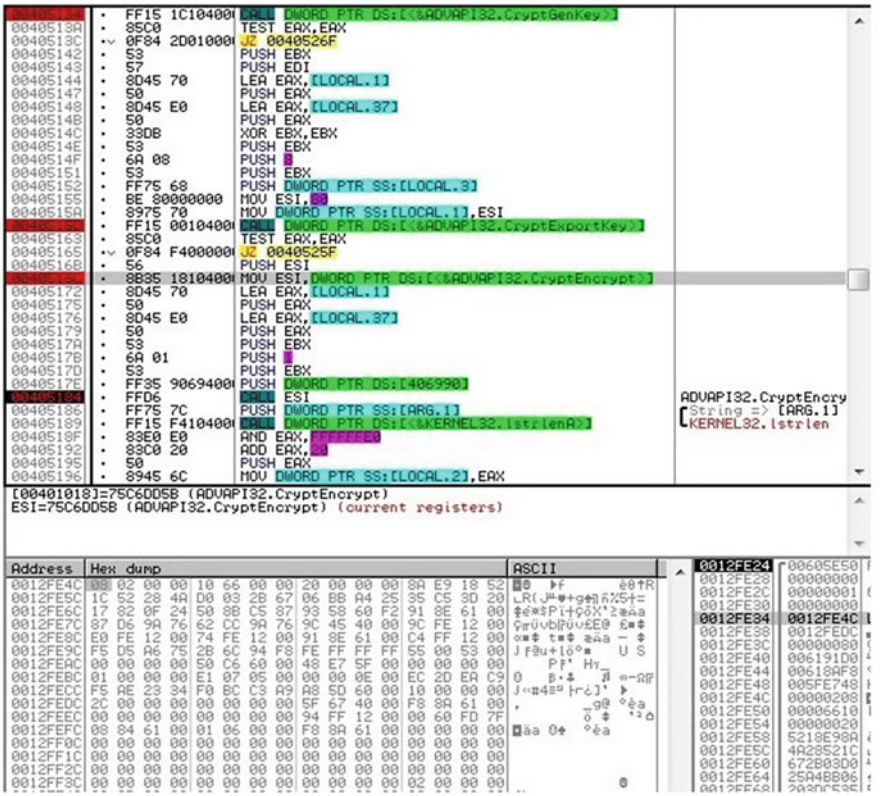


Fig. 12 Another AES key is generated, exported, and encrypted using the embedded RSA key

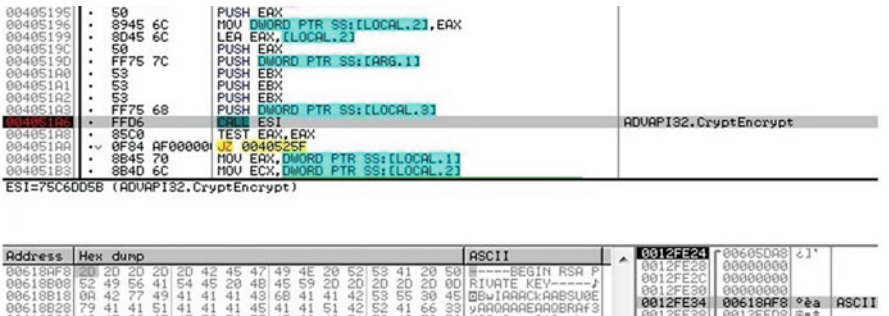


Fig. 13 The AES key, which was generated, is used to encrypt a private RSA key

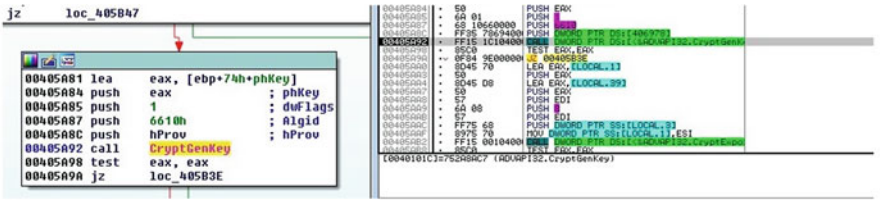


Fig. 14 Another AES256 key is generated

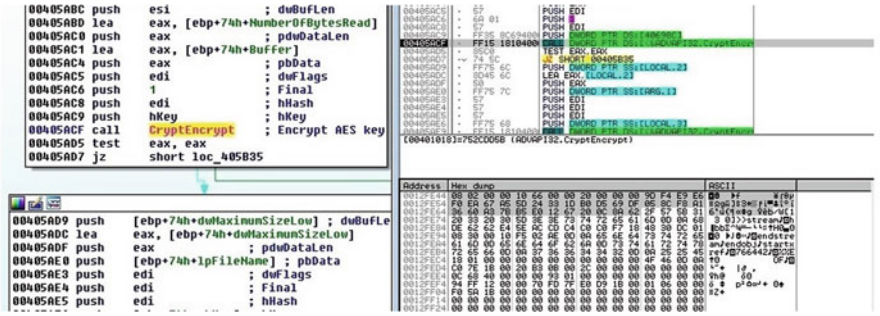


Fig. 15 The AES key is encrypted using RSA key

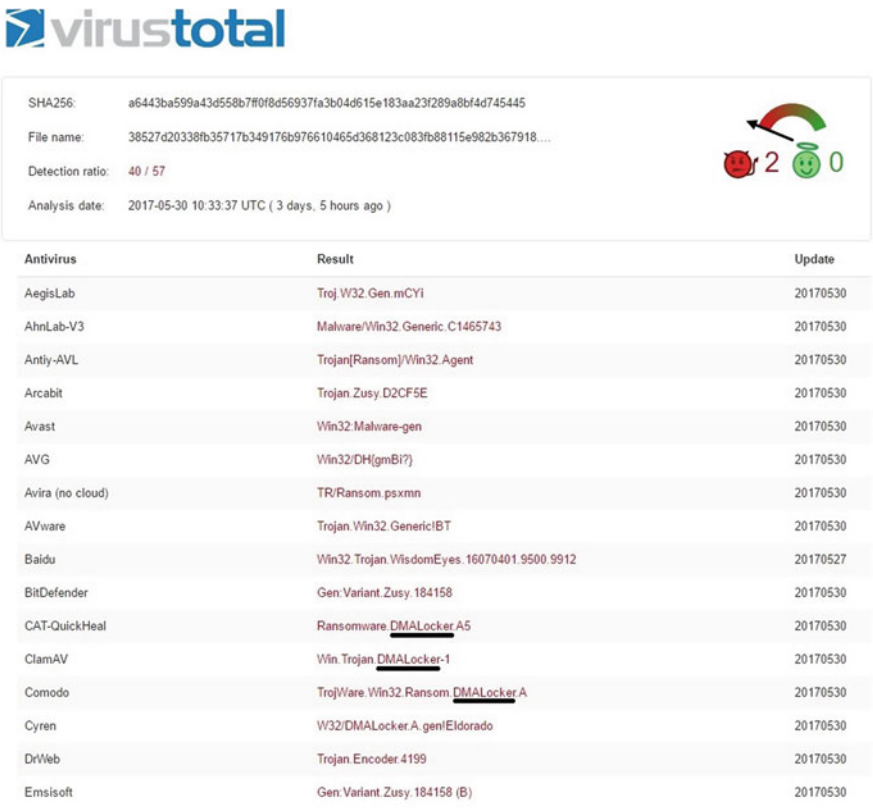
Fig. 16 The file is encrypted using the AES key

2 DMA Locker Ransomware

Name: dma.exe
md5: FDECD41824E51F79DE6A25CDF62A04B5
Type: encrypting ransomware

In Fig. 17 a report by VirusTotal, which shows that the malware is known to most vendors, is presented.

According to Fig. 18, the ransomware isn't packed; if this is obfuscated, it is then necessary to reveal it. As shown in Fig. 19, the malware moves the original file to C:\ProgramData and renames the file svchosd.exe (the author of ransomware is trying to hide the malicious purposes, in order to look like the Service Host Process svchost.exe). Once the file is copied, the malware



virustotal

SHA256: a6443ba599a43d558b7ff0f8d56937fa3b04d615e183aa23f289a8bf4d745445
File name: 38527d20338fb35717b349176b976610465d368123c083fb88115e982b367918....
Detection ratio: 40 / 57
Analysis date: 2017-05-30 10:33:37 UTC (3 days, 5 hours ago)

2 0

Antivirus	Result	Update
AegisLab	Troj.W32.Gen.mCYi	20170530
AhnLab-V3	Malware/Win32.Generic.C1465743	20170530
Antiy-AVL	Trojan[Ransom]/Win32.Agent	20170530
Arcabit	Trojan.Zusy.D2CF5E	20170530
Avast	Win32:Malware-gen	20170530
AVG	Win32/DH{gmBI?}	20170530
Avira (no cloud)	TR/Ransom.psxmn	20170530
AVware	Trojan.Win32.Generic!BT	20170530
Baidu	Win32.Trojan.WisdomEyes.16070401.9500.9912	20170527
BitDefender	Gen:Variant.Zusy.184158	20170530
CAT-QuickHeal	Ransomware.DMALocker.A5	20170530
ClamAV	Win.Trojan.DMALocker-1	20170530
Comodo	TrojWare.Win32.Ransom.DMALocker.A	20170530
Cyren	W32/DMALocker.A.gen!Eldorado	20170530
DrWeb	Trojan.Encoder.4199	20170530
Emsisoft	Gen:Variant.Zusy.184158 (B)	20170530

Fig. 17 VirusTotal report DMA Locker

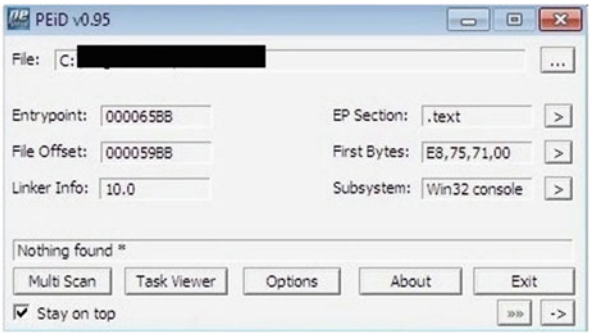


Fig. 18 PEiD report DMA Locker

Fig. 19 The malware moves the original file to another location

starts svchosd.exe process (which obviously is a copy of the original process) and then exits. As shown in Fig. 20, the function **CreateProcessW** is used. The original process creates two keys in registry for persistence:**HKLM\Software\Microsoft\Windows\CurrentVersion\Run\Windows Firewall** which has the value C:\ProgramData\svchosd.exe and **HKLM\Software\Microsoft\Windows\CurrentVersion\ Run\Windows Update**, which has the value notepad C:\ProgramData\cryptinfo.txt (at every reboot the ransom note is shown). The DMA Locker deletes backups and shadow copies, using the native Windows utility VSSAdmin, as shown in Fig. 21. A start.text file is created to show that the encryption has begun (and there is no need to restart it again). Logical disks and network shares are attacked, and checks against the Floppy and CD using **QueryDosDeviceA**(Floppy and CD are skipped) are made, as shown in Fig. 22. The sample uses a hardcoded public RSA key, stored in a form of BLOB, as shown in Fig. 23. Some directories are excluded from the encryption process; this entire list is in Fig. 24. A list of skipped extensions is presented in Fig. 25. A unique AES256 key is generated for every file using the API **CryptGenRandom**, as shown in Fig. 26. The AES key is used to encrypt 16-byte-long data with AES ECB mode, as shown in Fig. 27. Once used, the AES key is encrypted using the hardcoded RSA key (Fig. 28). The structure of the encrypted file is the prefix which is added, the encrypted AES key, and the encrypted original content (Fig. 29). Once the encryption process is complete, a message alert is presented (Fig. 30). The malware may be fooled in order to avoid the encryption through the creation of the files start.txt and cryptinfo.txt in ProgramData directory. If these two files are present, the encryption cannot start and only the ransom message is displayed. However, if the algorithms, which are used in the encryption process, are consistent, the decryption without the RSA private key which is kept secret will not be possible.

Fig. 20 The malware starts a copy of the original process

Fig. 21 DMA Locker deletes backups and shadow copies

Fig. 22 Floppy and CD are skipped

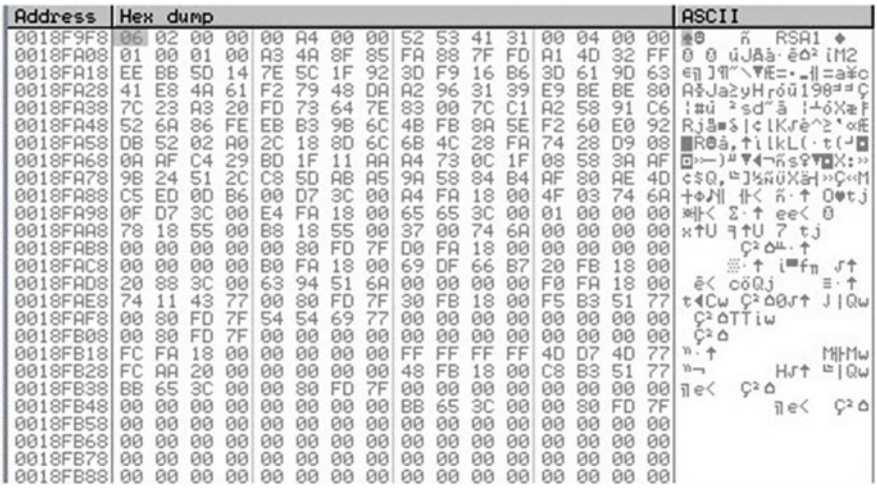


Fig. 23 Hardcoded RSA key

```
003C2880 push    ebp
003C2881 mov     ebp, esp
003C2883 sub     esp, 2Ch
003C2886 push    esi
003C2887 mov     [ebp+var_2C], offset aWindows ; "\\Windows\\"
003C288E mov     [ebp+var_28], offset aWindows_0 ; "\\WINDOWS\\"
003C2895 mov     [ebp+var_24], offset aProgramFiles ; "\\Program Files\\"
003C289C mov     [ebp+var_20], offset aProgramFilesX8 ; "\\Program Files (x86)\\"
003C28A3 mov     [ebp+var_1C], offset aGames ; "Games"
003C28AA mov     [ebp+var_18], offset aTemp ; "\\Temp"
003C28B1 mov     [ebp+var_14], offset aSamplePictures ; "\\Sample Pictures"
003C28B8 mov     [ebp+var_10], offset aSampleMusic ; "\\Sample Music"
003C28BF mov     [ebp+var_C], offset aCache ; "\\cache"
003C28C6 mov     [ebp+var_8], offset aCache_0 ; "\\Cache"
003C28CD xor     esi, esi
003C28CF nop
```

Fig. 24 The directories which are excluded from the encryption

```
003C2907 mov     [ebp+var_30], offset a_exe ; ".exe"
003C290E mov     [ebp+var_2C], offset a_msi ; ".msi"
003C2915 mov     [ebp+var_28], offset a_dll ; ".dll"
003C291C mov     [ebp+var_24], offset a_pif ; ".pif"
003C2923 mov     [ebp+var_20], offset a_scr ; ".scr"
003C292A mov     [ebp+var_1C], offset a_sys ; ".sys"
003C2931 mov     [ebp+var_18], offset a_msp ; ".msp"
003C2938 mov     [ebp+var_14], offset a_com ; ".com"
003C293F mov     [ebp+var_10], offset a_lnk ; ".lnk"
003C2946 mov     [ebp+var_C], offset a_hta ; ".hta"
003C294D mov     [ebp+var_8], offset a_cpl ; ".cpl"
003C2954 mov     [ebp+var_4], offset a_msc ; ".msc"
```

Fig. 25 Skipped extensions

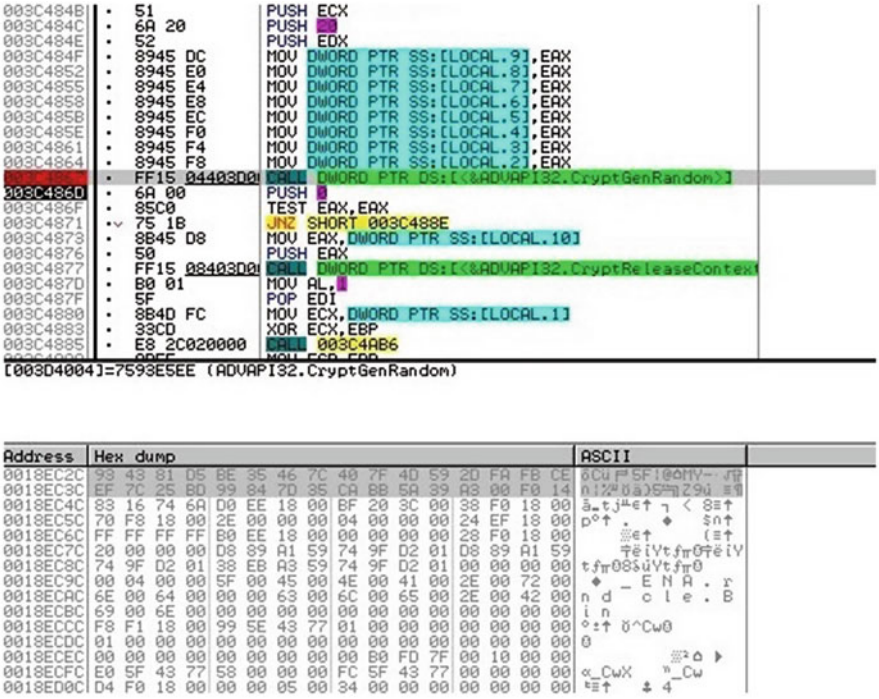


Fig. 26 A unique AES key is generated

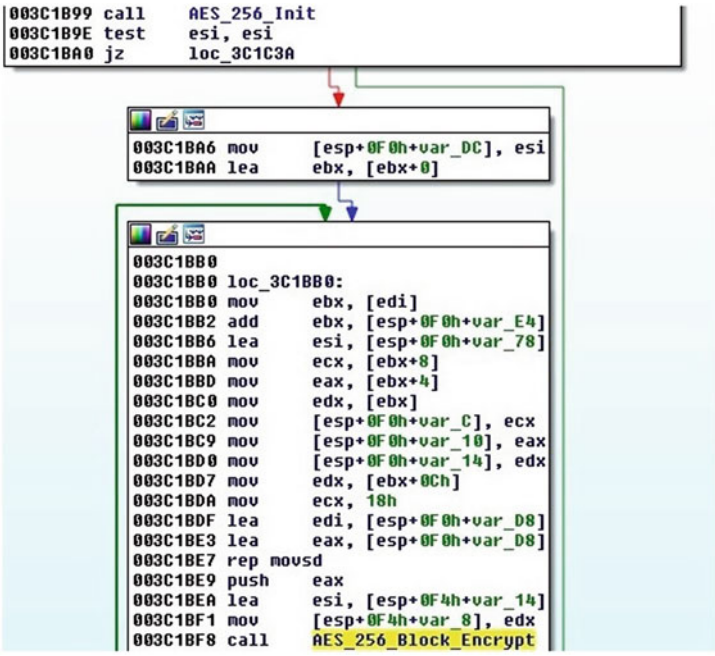


Fig. 27 The data is split in chunks of 16 bytes and encrypted

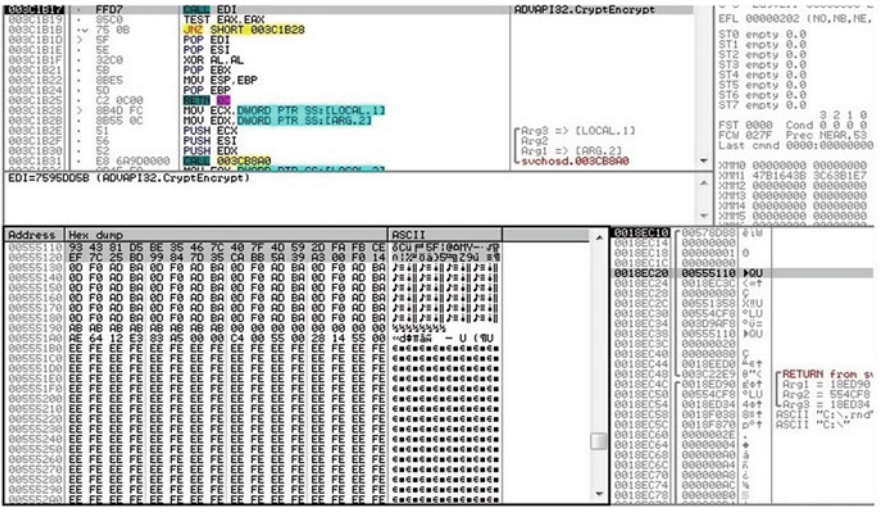


Fig. 28 The AES key is encrypted using the hardcoded RSA key

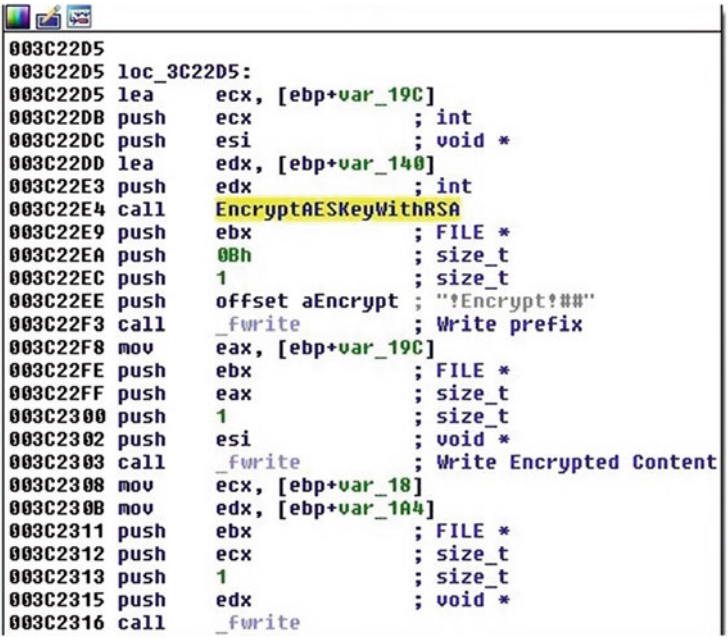


Fig. 29 A prefix is added to each file

Fig. 30 DMA Locker Message

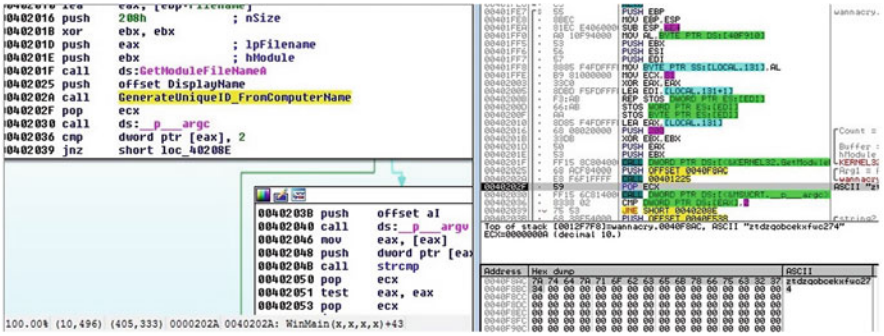


Fig. 31 A unique identifier is generated for every victim

Fig. 32 Ransom notes

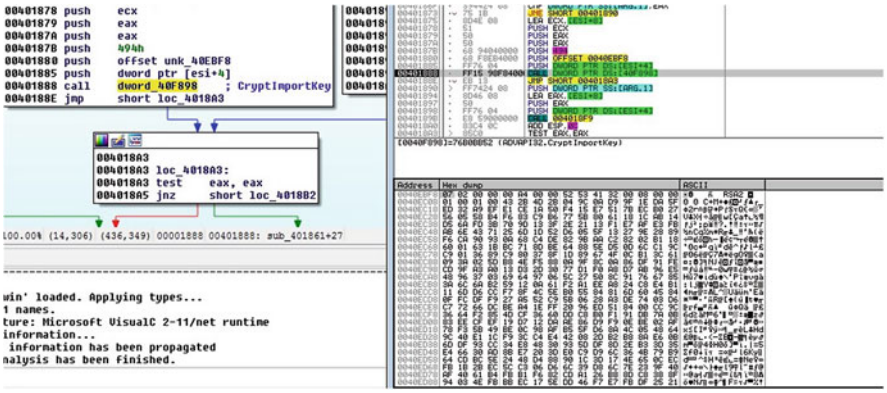


Fig. 33 Private RSA key is being imported

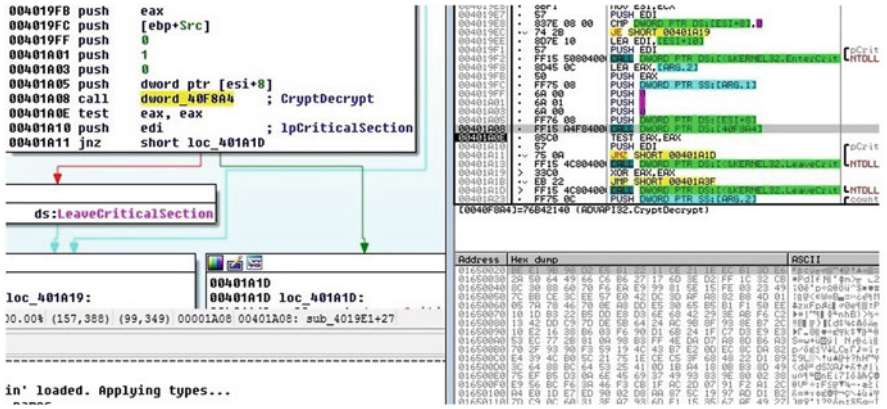


Fig. 34 The encrypted key is decrypted using private RSA key

Fig. 35 Public RSA key is being imported

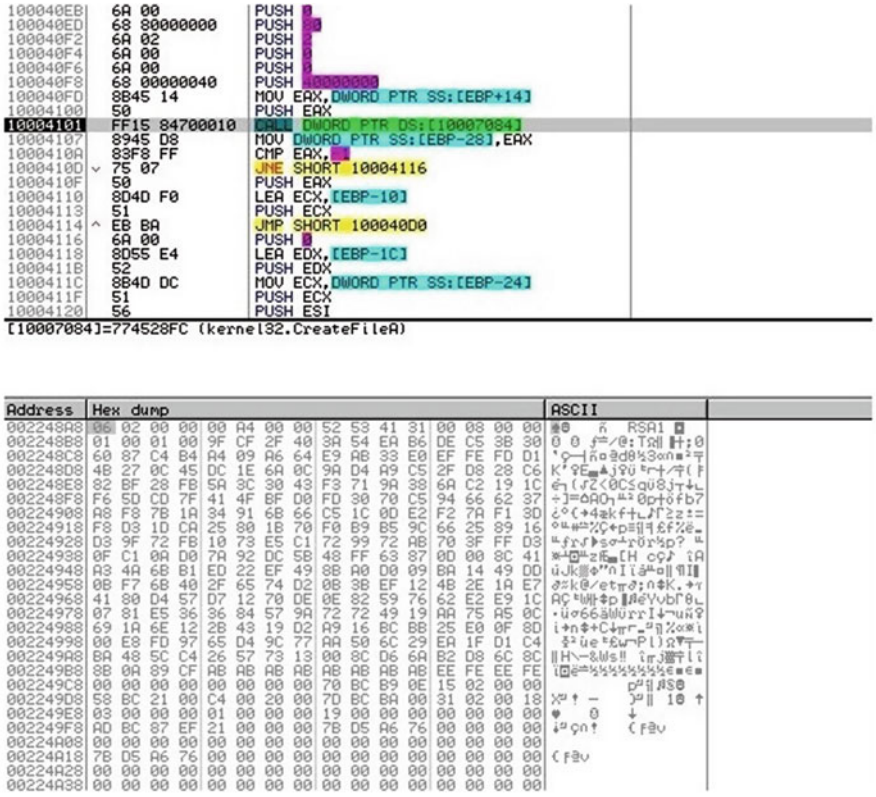


Fig. 36 The public key is exported and saved to 00000000.pky

Fig. 37 The private key is encrypted using hardcoded RSA key

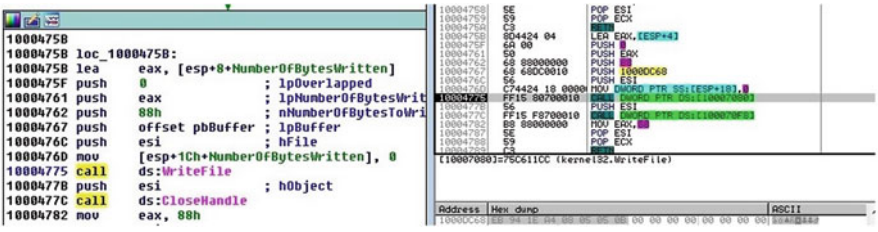


Fig. 38 Firstly, the 8 generated bytes and 128 zero bytes are written to the file

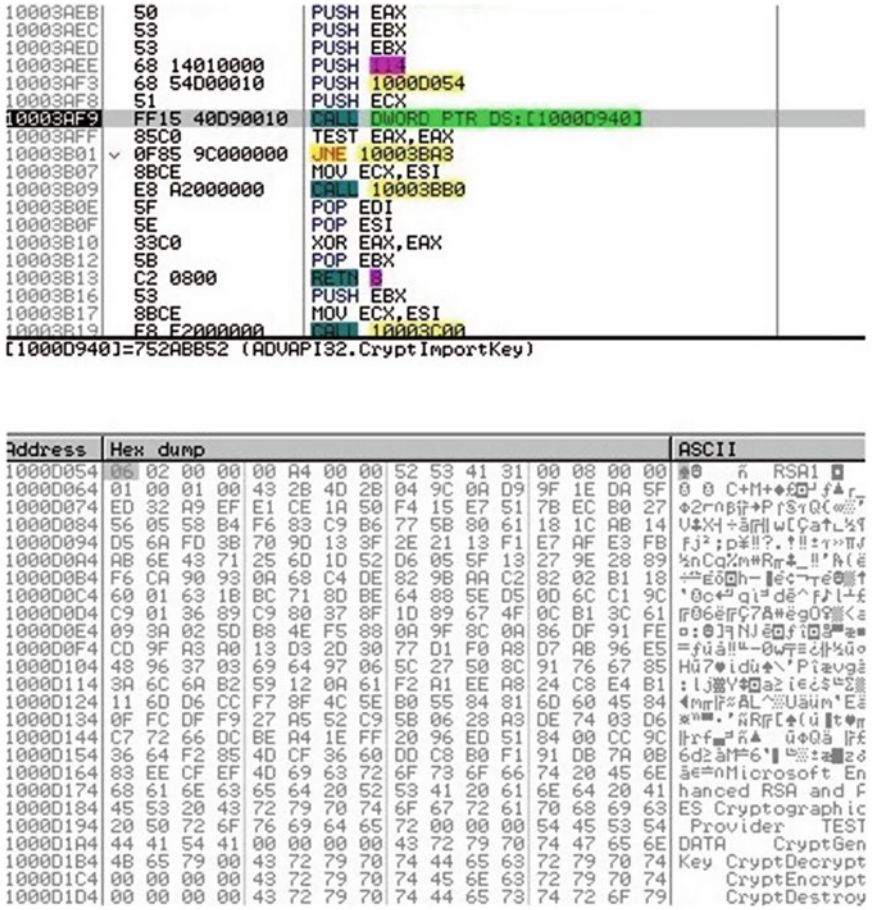


Fig. 39 Another RSA key is being imported

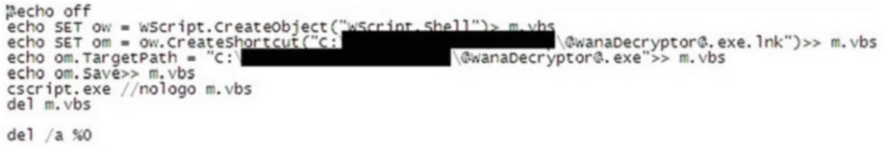


Fig. 40 The malware creates a LNK which points to @WanaDecryptor@.exe

```
.der.pfx.key.crt.csr.p12.pem.odt.ott.sxw.stw.uot.3ds.max.3dm.ods.ots.sxc
.stc.dif.slk.wb2.odp.otp.sxd.std.uop.odg.otg.sxm.mml.lay.lay6.asc.sqlite3
.sqlitedb.sql.accdb.mdb.db.dbf.odb.frm.myd.myi.ibd.mdf.ldf.sln.suo.cs
.c.cpp.pas.h.asm.js.cmd.bat.ps1.vbs.vb.pl.dip.dch.sch.brd.jsp.php.asp.rb
.java.jar.class.sh.mp3.wav.swf.fla.wmv.mpg.vob.mpeg.asf.avi.mov.mp4.3gp.mkv
.3g2.flv.wma.mid.m3u.m4u.djvu.svg.ai.psd.nef.tiff.tif.cgm.raw.gif.png.bmp
.vcd.iso.backup.zip.rar.7z.gz.tgz.tar.bak.tbk.bz2.PAQ.ARC.aes.gpg.vmx.vmdk
.vdi.sldm.sldx.sti.sxi.602.hwp.edb.potm.potx.ppam.ppsx.ppsm.pps.pot.pptm.xltm
.xltx.xlc.xlm.xlt.xlw.xlsb.xlsm.dotx.dotm.dot.docm.docb.jpg.jpeg.snt.onetoc2
.dwg.pdf.wk1.wks.123.rtf.csv.txt.vsdx.vsd.eml.msg.ost.pst.pptx.ppt.xlsx.xls.docx
.doc
```

Fig. 41 Targeted extensions by malware

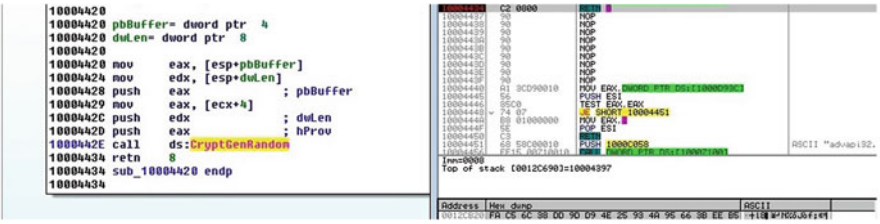


Fig. 42 A new AES key is generated for every file

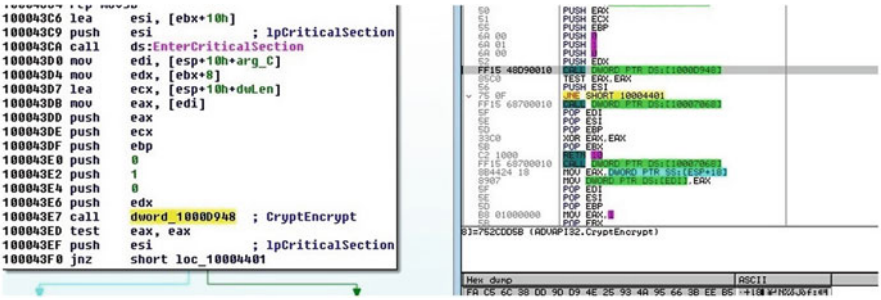


Fig. 43 The AES key is encrypted using RSA key

```
100058D3 push    0               ; lpExitCode
100058D5 push    0               ; dwMilliseconds
100058D7 push    offset aTaskkill_exeFI ; "taskkill.exe /f /im Microsoft.Exchange.".
100058DC call    sub_10001080
100058E1 push    0               ; lpExitCode
100058E3 push    0               ; dwMilliseconds
100058E5 push    offset aTaskkill_exe_0 ; "taskkill.exe /f /im MSExchange*"
100058EA call    sub_10001080
100058EF push    0               ; lpExitCode
100058F1 push    0               ; dwMilliseconds
100058F3 push    offset aTaskkill_exe_1 ; "taskkill.exe /f /im sqlserver.exe"
100058F8 call    sub_10001080
100058FD push    0               ; lpExitCode
100058FF push    0               ; dwMilliseconds
10005901 push    offset aTaskkill_exe_2 ; "taskkill.exe /f /im sqlwriter.exe"
10005906 call    sub_10001080
1000590B push    0               ; lpExitCode
1000590D push    0               ; dwMilliseconds
1000590F push    offset aTaskkill_exe_3 ; "taskkill.exe /f /im mysqld.exe"
10005914 call    sub_10001080
10005919 add     esp, 3Ch
```

Fig. 44 Executed commands after the encryption is over

3 WannaCry Ransomware

Name: diskpart.exe
md5: 84c82835a5d21bbcf75a61706d8ab549
Type: encrypting ransomware

The malware generates a unique identifier based on the computer name, as shown in Fig. 31. A check is made to see if the malware was started with /i argument.

Run with /i Argument
The malware copies the binary to C:\ProgramData\$\langle GeneratedID\rangle$\tasksche.exe if the directory exists; otherwise it is copied to C:\Intel\$\langle GeneratedID\rangle$\tasksche.exe and updates the current directory to the new directory. The binary tries to open the service named $\langle GeneratedID\rangle$. If it doesn't exist, the malware creates one with DisplayName $\langle GeneratedID\rangle$, the BinaryPath of cmd \c $\langle PathOftasksche.exe\rangle$, and starts the service. It attempts to open the mutex Global\MsWinZonesCacheCounterMutexA0; if it isn't created within 60 s, the malware starts itself with no arguments.

Run Without /i Argument
The binary updates the current directory to the path of the module and creates a new registry key HKLM\Software\WanaCrypt0r\wd which is set to the CD. The

malware then loads the XIA resource and extracts multiple files to the current directory; the complete list is shown below:

Filename	MD5 hash
b.wnry	c17170262312f3be7027bc2ca825bf0c
c.wnry	ae08f79a0d800b82fcbe1b43cdbdbefc
r.wnry	3e0020fc529b1c2a061016dd2469ba96
s.wnry	ad4c9de7c8c40813f200ba1c2fa33083
t.wnry	5dcaac857e695a65f5c3ef1441a73a8f
u.wnry	7bf2b57f2a205768755c07f238fb32cc
taskdl.exe	4fef5e34143e646dbf9907c4374276f5
taskse.exe	8495400f199ac77853c53b5a3f278f3e

The msg directory is created with different ransom notes in multiple languages (Fig. 32). The ransomware opens c.wnry (configuration data) and loads it into memory. The malware chooses between three bitcoin addresses, 13AM4VW2dhxYgXeQepoHkH SQuy6NgaE b94, 12t9YDPgwueZ9NyMgw519-p7AA8isjr6SMw, and 115p7UMMngoj1 pMvkpHijcRdfJNXj 6LrLn, writes it to offset 0xB2 in the config data, and writes the updates back to c.wnry. The binary sets a hidden attribute to the current directory using CreateProcessA API with **attrib +h** and executes the command **icacls ./grant Everyone:F /T /C /Q** in order to grant all users permissions to the current directory.

The malware uses **CryptImportKey** to import the hardcoded private RSA key (Fig. 33). The file t.wnry is then opened and the first 8 bytes are compared with the magic value "WANACRY!"; the next 4 bytes need to be 0x100; then the next 256 bytes are written in memory. The encrypted key decrypts to the AES key BEE19B98D2E5B12 211CE211EECB13DE6, as shown in Fig. 34. The AES key is used to decrypt the encrypted data, which was read from t.wnry and saves the result as a DLL. The TaskStart export function of the DLL is called, and it deals with the encryption of the files. It creates a mutex which is called **MsWinZonesCacheCounterMutexA** and reads the configuration file c.wnry. A new mutex is then created by the ransomware, **Global\MsWinZonesCacheCounterMutexA0**.

The binary will try to open a file 00000000.dky file, which at this point doesn't exist, and it will then try to load a 00000000.pky file. If this one doesn't exist, the ransomware will then import a public RSA key, as shown in Fig. 35. A new pair of RSA2048 keys is generated and the public key is saved to 00000000.pky, as shown in Fig. 36. The malware uses the hardcoded RSA key to encrypt the generated private key and saves the result to 00000000.eky (Fig. 37). A thread that writes 136 bytes to 00000000.key is created every 25 s (if it exists, otherwise it is created). Initially, as shown Fig. 38, 8 generated bytes and 128 zero bytes are written to the file, and after that it is written to a buffer, which contains the current time of the system. A thread that launched taskdl.exe, which is used to delete encrypted files, is created (which has that specific extension). Another thread is created that scans for new drives every 3 s; if it finds a new drive and this isn't a CDROM drive, it

encrypts the drive. The sample imports another RSA key, as shown in Fig. 39. The process @WanaDecryptor@.exe with the "fi" argument is created, and this one can communicate with the server in order to obtain an updated bitcoin address. The file u.wrny is copied and saved as @WanaDecryptor@.exe; a script file is created and executed with the content shown below. The ransomware reads the content of r.wnry, updates the content with a ransom amount and bitcoin address, and writes the content to @Please_Read_Me@.txt (Fig. 40). The process starts scanning a directory, creates a hidden file with the prefix "∼SD," and then deletes it. The files which have the .exe, .dll, and .WNCRY extensions as well as the files which were created by the malware are not encrypted. The list of attacked extensions is presented in Fig. 41. Each file is encrypted using AES-128 algorithm in CBC mode with NULL IV. For every file a unique AES key is generated, as is shown in Fig. 42. The structure of an encrypted file is WANACRY!, length of RSA encrypted data, RSA encrypted AES key, file type, original file size, and AES encrypted content. The AES key is encrypted using the embedded RSA key or generated RSA key depending on a number which is generated (if it is a multiple of 100, the AES key is encrypted using the embedded RSA key; otherwise it is encrypted using the generated RSA public key), as shown in Fig. 43. The ransomware executes the following commands after the encryption is finished (Fig. 44). The process is trying to encrypt the logical drives that aren't of DRIVE_CD
ROM type; it executes the commands **@WanaDecryptor@.exe co** and **cmd.exe /c start /b @WanaDecryptor@.exe vs** and copies the b.wnry to every folder on the desktop (it is saved as @WanaDecryptor@.bmp). The encryption algorithms are consistent and it is not possible to restore the files without paying the ransom; however there are some decryptors that work for Windows XP, Windows 7, Windows Vista, and Windows Servers 2003 and 2008.

Acknowledgment The authors would like to thank University Politehnica of Bucharest for the financial support.

Applying Model-Based Situational Awareness and Augmented Reality to Next-Generation Physical Security Systems

Elaine M. Raybourn and Ray Trechter

Abstract Mixed, augmented, and virtual reality holds promise for many security-related applications including physical security systems. When combined with models of a site, an augmented reality (AR) approach can be designed to enhance knowledge and understanding of the status of the facility. The present chapter describes how improved modeling and simulation will increase situational awareness by blurring the lines among the use of tools for analysis, rehearsal, and training—especially when coupled with immersive interaction experiences offered by augmented reality. We demonstrate how the notion of a digital twin can blur these lines. We conclude with challenges that must be overcome when applying digital twins, advanced modeling, and augmented reality to the design and development of next-generation physical security systems.

1 Introduction

Augmented reality (AR), mixed reality (MR), and virtual reality (VR) hold promise for many security-related applications including installation security. When combined with a virtual representation of a site created through modeling, these approaches can be designed to enhance the knowledge and understanding of the status of the facility. A user can view and interact with a 3D model of an entire facility updated with probabilistic assessments based on all current data including predictions for likely potential threats. The commander could know exactly where security personnel are at all times, and the system could guide the operator's actions based on current and historical data [1].

As cyber-physical security systems become model-based and leverage augmented, virtual, or mixed reality, the gaps between training, planning/analysis, and situational awareness simulations disappear. Through a model-driven contextual

E. M. Raybourn (✉) · R. Trechter
Sandia National Laboratories, Albuquerque, NM, USA
e-mail: emraybo@sandia.gov; rtrech@sandia.gov

Ç. K. Koç (ed.), *Cyber-Physical Systems Security*,
https://doi.org/10.1007/978-3-319-98935-8_16

interface, trainees have the ability to experience a virtual representation of a real-world facility and participate in realistic training. Security force leadership can similarly use this model to improve tactics or to plan upgrades. Users of these systems will be able to virtually experience a combat sequence or, in the case of actual watch standers, participate in virtual no-notice drills.

Current efforts by the authors and others include the development of flexible, powerful tools for analyzing security in operational spaces, particularly facilities, and their surrounding terrain. These physics-based, 3D, terrain-aware simulations analyze a system's performance often including the interplay between its components (e.g., sensors, energy, cybersecurity, and personnel). The use of autonomous systems, especially Unmanned Aerial Systems (UAS), as dynamic sensors and other applications is underway. AR is being used to explore and visualize new security concepts. Simulations also often incorporate a mixture of these live and simulated assets.

A science and technology (S&T) goal for next-generation physical security facilities is to increase situational awareness with the use of new technologies such as the integration of artificial intelligence, machine learning, and software analytics with a virtual representation or model of the site [2]. Professor Michael Grieves [3] coined the term "digital twin" in 2003 to refer to "a virtual, digital equivalent to a physical product." This term gained traction in the past decade and has been expanded to manufacturing enterprises, operations, and facilities. When combined with data from sensors, the devices, personnel, and other sources create a living, digital simulation model or digital twin of a site [4]. A facility's digital twin updates and changes as their physical counterparts change, providing understanding of each unique asset, in this case a facility, over time. In addition to real-time data feeds, a digital twin can be informed by historical data from a variety of sources. The twin is not just a generic model of a facility; it is for all intents and purposes a representation of a specific site that improves with data over time.

Ultimately, the creation of a secure site's digital twin will provide new and more versatile tools for evaluating security systems that blur the lines between activities such as real-time situational awareness, command and control, design, analysis, training, and various modes of exercises—be they tabletops or force-on-force rehearsals. Immersive technologies underpinning a digital twin approach will accelerate the adoption of intelligent, adaptive training ecosystems for game-based and transmedia learning [5, 6]. The twin and its data can support a virtual environment, or world, for VR training applications with multiple participants. That same digital twin can easily provide coordination of virtual assets, along with virtual features and cues for AR training applications. Physical security system designers can take advantage of a twin collecting data and learning over time to check proposed design changes, and a vulnerability analyst can use that same data to identify possible threats and a site's readiness to handle them.

Just as important as creating high-fidelity simulations with these techniques are the real-time data channels that feed real sensory data to the virtual representation and vice versa. It is here where autonomous systems and humans with AR technology work with the virtual system to improve overall situational awareness.

AR presents a compelling opportunity to improve security personnel's situational awareness by displaying elements of the virtual world's model, including entities that are not in the responder's line of sight, and predictive analytics based on a simulation's ability to run faster than real time. Examples where the predictive capabilities of simulations might be quite helpful are providing security forces paths that avoid enemy fire or observation, and predicting the future positions of hostile forces based on previous observations.

However, a sobering realization is that far too many critical infrastructure systems and facilities are vulnerable to cyber and physical attack. The physical security installation community has identified several emerging scenarios which serve to update critical infrastructure defensive security countermeasures, but nevertheless there always remain a number of considerations [7]. For example, modern physical security systems and facilities that rely on subsystems communicating via Internet Protocol have given rise to cyber-physical attacks. Cyber-physical system attacks can cripple a nation's critical infrastructure, energy grids, transportation, etc.

Model-based situational awareness is required for improvements in analysis, rehearsal, and training. Subsequent sections of the present chapter describe how achieving this S&T goal is addressed with tools for modeling, simulation, and current practices underway. Simulations are enhanced especially when coupled with immersive interaction experiences offered by AR, MR, and VR [8].[1] We discuss the notion of digital twin in the context of physical security system installations. We then apply this concept to a hypothetical use case loosely based on actual events, in which we set the stage for the ways improved modeling and simulation may facilitate improved situational awareness. We conclude by identifying challenges and proposing recommendations for next-generation physical security systems.

2 Model-Based Situational Awareness for Physical Security

A facility's physical security system is truly a system of systems when one considers all the elements needed to secure a location. To model a facility at the necessary level of fidelity, a site's barriers, buildings, sensors, vehicles, people, and other significant real-world objects must be presented in a model. When done well, simulations may detect vulnerabilities in tactical operations by analyzing the environment based on geography, sensing, and timing. Users can then conduct specified analyses, such as the effectiveness of observation posts in detecting targets and exploring multiple phenomenology including physical, cyber, and human behavioral effects. These analyses may allow a user to target specific areas of concern, minimizing overall system costs [9].

Many tools are used to improve the physical security of facilities today. For example, the Joint Conflict and Tactical Simulation (JCATS) is a well-known

[1]The combination of AR, MR, and VR with real environments is also known as XR (extended reality)

human-in-the-loop simulator that allows response force effectiveness to be evaluated through live, force-on-force exercises using teams of attackers and defenders. However, these exercises can be costly and time-consuming. That said, a simulated force-on-force exercise with JCATS is a great alternative to a live exercise, as it captures the critical human dimension introduced by system operators and provides the opportunity for participants to share knowledge and train together. Site personnel can improve site security by combining simulated exercise results with robust data analysis, the results of other models and simulations, and consultation with subject matter experts.

Facilities also use video game technology, such as serious games and game-based training, to facilitate cognitive training and experiential learning in situated contexts and immersive scenarios [5, 6, 8]. The goals of these cognitive trainers for physical security system personnel are often enhanced retention of knowledge, skill development, and practice of key training objectives. When combined with analytics resulting from tools such as JCATS and game-based trainers, virtual environments become force multipliers (Fig. 1).

Current modeling and simulation capabilities support the analytics and technology underpinning the next-generation digital twin and, as such, directly determine whether an advantage will be provided to the security forces. One such tool, *Dante*, develops physics-based, 3D, terrain-aware simulations that analyze a security system's performance [1]. Simulations include physical objects (e.g., buildings, equipment, vehicles, and weapons), people and their behaviors, communications,

Fig. 1 Virtual facility display with Dante [1]

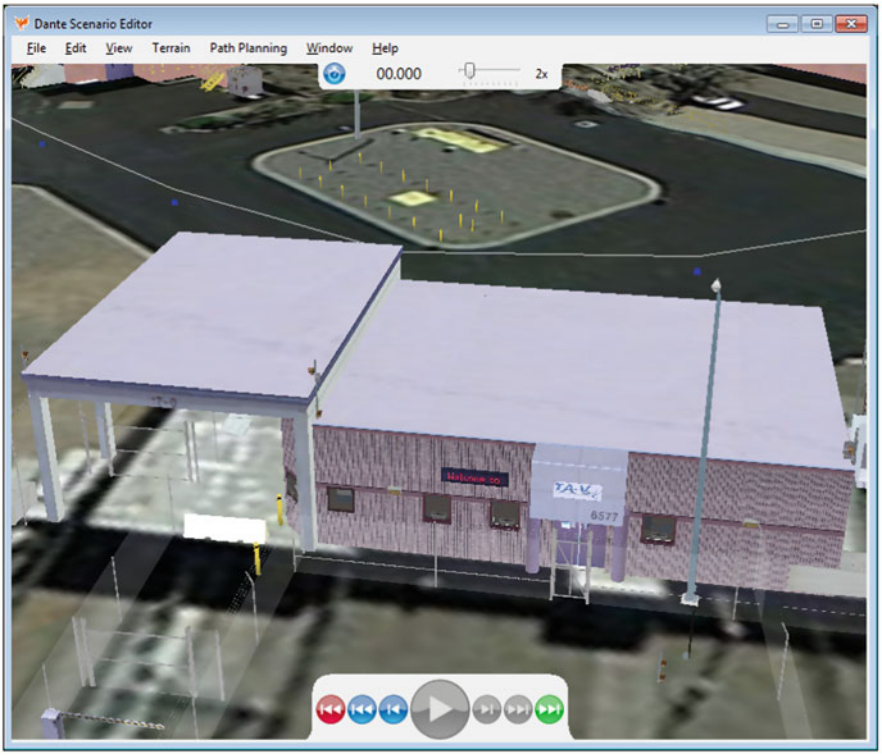


Fig. 2 Dante terrain with features (e.g., buildings, fences) [1]

cyber systems, and the interplay between each of these components. Intelligent characters are driven by non-deterministic simulated behaviors. The characters are influenced by their environment *and* their perceptions. As such, they may respond differently given the same situation. An important consideration for this discussion is that the simulations created with Dante may incorporate a mixture of live and simulated assets. Linkages between the virtual and physical worlds are built into this simulation framework.

Creating an accurate 3D terrain model of a facility for use in a Dante simulation is of key importance (Fig. 2). Terrain models serve as the synthetic environment for a site's virtual world and as a vital backdrop for simulation, exploration, and visualization of security concepts and operations. These terrain models have elevation data, imagery, road data, barriers, buildings, and often building interiors. Geographic Information Systems (GIS) and 3D modeling tools are used along with tools that seek discontinuities, paths, and features to improve confidence in terrain accuracy. Fortunately, terrain data in security simulations tends to consist of static features for a site (e.g., installation features such as buildings and roads do not tend to move often.) Accurate terrain supports path generation algorithms based upon multiple influences such as terrain features, sensor fields, data from imagery, and

energy signatures. For example, these algorithms can plan intelligent movement around a camera's view and take threats into account when finding an optimal route. Accurate terrain is especially important for AR applications as the location of physical world features needs to be mapped into the coordinates of the facility model so that virtual feature and cues generated by a simulation show up in an operator's field of view correctly (e.g., a suggested route on an operator's head-mounted display shows a path around a building rather than through it).

AR applications show promise for enhancing model-based situational awareness, especially when used with physics-based, 3D, terrain-aware simulations that can analyze a security system's performance. AR applications need not only enhance model-based situational awareness with visual representations or feedback. Representations may also be auditory, haptic, or olfactory. AR may also be dynamic, adaptive, and persistent. AR can be used with physical security system models to (1) serve as cues for the existence of sensors that are not visible (blind spots), (2) provide locations of key assets, (3) recommend defensive positions, (4) locate/mark opposing forces for training and experimentation, and (5) generate avatars reflecting "patterns of life," especially those representing vulnerable populations such as children, elderly, etc. While there remain technology maturity challenges with respect to the use of outdoor AR for geolocating personnel (motion, weather, blockage, night, sensor drift, etc.), these applications can support installation design, test and evaluation, and simulation of training and rehearsal.

In summary, we briefly discussed some of the tools for modeling and simulation that are characteristic of current practices underway today as an introduction to the notions that follow. In the next section, we discuss the application of a *digital twin* to the context of physical security system installations.

3 A Secure Facility Meets Its Digital Twin

Accurate facility modeling and terrain development are needed for security simulations in general and to implement a digital twin. Data feeds must be added to reflect the current situation on the ground along with adding predictive capabilities to this adaptive model. The site's digital twin needs sufficient data updates by various means (e.g., sensor input, updates from responders) to stay synchronized with physical counterparts, and support response force decisions with real-time status. Secure facilities usually have a variety of sensors including cameras, fence line sensors, and radars that can update the digital twin in real time, and this information can be made immediately available to the entire response force.

The response force itself is a source of real-time information. Staff in the central alarm/control station and responders typically use radio communications to direct forces and communicate an unfolding situation with each other. Having a digital twin focused on command and control systems, and the right equipment such as a smartphone, tablet, or a head-mounted display, allows communications to be relayed visually and persistently through an AR channel. A display is used in lieu

of, or in conjunction with, radios. The alarm station can mark entities such as enemy forces, identify areas of relative safety, or direct responders to engagement locations. Responders, in turn, can assess and mark threats and other items of tactical significance through their smart devices and feed these back to the twin that stitches together personnel and sensor reports into an overall picture for an engagement as it unfolds. This works the same with training exercises in and around the facility as data is captured from observer notes and After Action Review (AAR) reports along with data from automated engagement systems using Multiple Integrated Laser Engagement Systems (MILES). These data sources can be used to tune a digital twin's artificial intelligence (AI) over time, and, given the virtual representations' near-perfect understanding of the site terrain and pathways to target areas, the system becomes a formidable opponent that can be used to train response forces or act as an aide during actual operations.

In addition to real-time data of an operation, there are innumerable facility data sources that can be used to align a twin with its physical counterpart and improve its usefulness for decision support. A facility's badge system is a great source of information of normal and possibly anomalous access to secure areas [10]. The electronic security system that connects various sensors and cameras can be gleaned for false alarm rates by time of day, season, and environmental conditions such as sun, rain, and wind to better discriminate among real threats and noise in the environment. Recoding the weather may be important given its impact on sensors. Along with these data sources, video captures from numerous cameras can help to establish an installation pattern of life that can be data mined for possible threats.

As it turns out, a site's digital twin may be used in part to address another bedeviling problem faced by the installation community: cyberattacks upon elements of the physical security systems itself [11]. Typically, the operational networks with cameras and other sensors using IP for connectivity are air gapped and have careful configuration management. However, as is the case with other control systems, vulnerabilities may occur through improper configuration, insider attack, and cover communications links. Attacks on cyber infrastructure are hard to diagnose and may go unnoticed because it is the network itself that is used to detect and respond to cyber exploits. If the infrastructure is compromised to some degree and clever attackers cover their tracks, a digital twin can have access to the actual state of the device in the control system and not just the state reported on the network. According to Colin Parris, Vice President of Software Research for GE Global Research, "We're using physics to detect what's going on and we know what the normal state is for the machine" [11]. Using an industrial control system example, Parris further explained that "if a cyber attacker were to spoof a sensor it may be obvious if one sensor says it's 20 degrees and another says it's 200" [11].

Extending this example to the world of physical security, installations have access to overlapping cameras or other sensors covering a particular area. By directly accessing these devices through a separate or redundant path, the signals (e.g., pixels in the case of a camera) from each can be used to update their twin image. These virtual sensor representations can then be compared and triangulated to point out a sensor that is at odds with the status reported over the network

and thus detect some types of exploits [12]. For example, consensus may then be reached among different sensor types (e.g., do a fence line sensor and camera have a signal that represents an intruder in the same location?). What is perhaps even more powerful is the notion that digital twin images of sensors may be able to virtually reconstruct what *should be* seen by another device, allowing a compromised sensor or malfunctioning device to be bypassed and allowing security staff to continue their activities without a gap in their situational awareness.

In the next section, we describe a fictional scenario and fictional tools used by a team of attackers and hackers. This example attack with both a physical and cyber element is based loosely on the 2015 and 2016 attacks against the Ukraine power grid. S&T efforts are needed in artificial intelligence, machine learning, and software analytics if we are to create living, digital simulation models that update and change as their physical counterparts change.

4 Scene: Sunday, 1600. Somewhere in the Countryside . . .

A group of insurgents, traveling by van, roll through the countryside on a sleepy Sunday afternoon. The attack team's assignment is straightforward: they are to breach the facility perimeter, proceed to the location of the control systems, and use their explosives charges to damage the power plant (including backup generators) beyond repair, necessitating weeks for restoration. A cyber exploit will be used in conjunction with the physical attack to compromise the site's Alarm Communication and Display (AC&D) system and hide sensor alarms from the defenders. The attack team is armed modestly with rifles, breaching tools, and some explosive cutting charges. What the assault team lacks in materiel provisioning is made up for by intelligence along with an insider who is part of their team.

The facility, which is experiencing yet another routine day among many, is prepared. Its perimeter has a clear zone made from two parallel physical fences with sensors installed in between the fences. Access into this perimeter is controlled at vehicle and personnel access points. The facility bristles with cameras, thermal imagers, and other devices that feed into a central alarm station. Central alarm station operators monitor video stream and sensor alerts, assess potential anomalies and intrusions, and maintain contact with both forces and the watch commander via radio. Security forces patrol both outside the perimeter and within the facility, backed up by a Quick Reaction Force (QRF) on duty at security forces' headquarters a few minutes away. These security elements—cameras, sensors, and the alarm station—are connected by Alarm Communications and Display (AC&D) software, isolated on its own network. Modern equipment uses the IP protocol, which makes configuration and extension of the AC&D relatively easy. The AC&D is the trusted source for status of the cameras and sensors for the site's physical security. Unbeknownst to the facility's defenders, one of their own has been working patiently over time with a hostile information operations team to compromise the

AC&D at the targeted facility. Their chosen method uses a cyber exploit similar to that of the well-known attack on the Ukrainian power grid.

A fascinating article on the subject of the Ukraine power grid attacks was published in *Wired* on June 20, 2017. Recall that on December 23, 2015, at exactly midnight, a cyberattack to the power grid resulted in 225,000 Ukrainians losing electric power. The same thing happened almost exactly to the day a year later [13]. While power was lost only for a few hours on each occasion, it was enough to be noticed by the global community, especially since it had been noted by Ukrainian officials that "there had been 6500 cyber attacks on 36 Ukrainian targets in just the previous 2 months," including a cyberattack that took down two servers at the same time at StarLightMedia, the largest broadcast conglomerate in the Ukraine [13]. During the forensic analysis of the SilverLightMedia cyber attack, it was discovered that "the hackers used BlackEnergy for access and reconnaissance, then KillDisk for destruction" [13]. By this time, BlackEnergy and KillDisk had infected the networks of at least three Ukrainian power companies and were waiting to be deployed by the hackers at the appropriate moment [14, 15]. All this was preparatory work for the main thrust of the attack; a copy of control software used by the power company had been surreptitiously obtained by the hackers and was run remotely to issue commands that shut down power generation to a large part of the country. According to Greenberg [13], attacks of this kind are becoming more common as hackers find way to obtain copies of system control software and make their own "enhancements."

In our notional scenario, the goal for the cyber element of the attack is to hide the many alarms, video, and other information provided by the AC&D, thereby blinding the central alarm stations and providing the attackers with a tactical advantage. Preparation for this type of attack begins with the acquisition of a legitimate copy of the AC&D software [7]. The software is then examined for configuration options, supporting XML definitions and source code when available. Changes to the Human-Machine Interface (HMI) to not show sensor alarms leave the alarm station operators and watch commander in the dark about events as they unfold and thus prevent effective response. A special command line key sequence can be added that allows the AC&D software to function normally during system checkout and mask off alarms prior to an attack. All that is needed is for an insider colluding with the attack team to load the exploited AC&D software during routinely scheduled maintenance and upgrades.

To execute the plan discussed above, the attack team stops by the road a half-mile away from the mission-critical facility, out of sight and beyond the site's sensor field coverage. Five insurgents exit the van and begin their approach to the south of the facility, while those remaining in the van head north. The attack team on foot is a diversion. They plan to breach and attack the side of the facility opposite of the building housing the critical asset, so as to draw the security personnel toward the south. These attackers take a stealthy approach to the assigned breach point on the fence. They wait for an opportune time to cut the fence and then move aggressively to a diversion target building in the facility. The team remaining in the van positioned themselves and began the main assault from the north at the sound

of gunfire and communications from the diversion team leader that their breach had been successful. The main attack team hopes to win a race against the distracted security personnel to the target building, place their charges in a fashion to cut both normal and backup power, and engage security forces as they arrive. All seems normal from the display in the alarm station and will continue to appear that way even as the facility enters the fight of its life.

4.1 A Digital Twin Saves the Day

With the standard setup of cameras and sensors feeding a central AC&D, the defense of such a facility would be in question. Lacking the situational awareness to discern the number and direction of the attacks in a fight that lasts but a few minutes, there is a reasonable chance that the diversion will succeed, drawing security personnel away just long enough for the main assault on the critical asset building. However, this facility has a digital twin that includes not only the physical security system and its AR displays but also other aspects of the site such as emergency services, building automation systems, and utility usage. The twin has many uses such as predictive maintenance and energy management, but in this case, it serves as secondary status or a watchdog for the electronic security system (ESS), which has virtual representations for all components, even the AC&D system itself. The digital twin is implemented in a continuous simulation with a separate path to sensors sometimes avoiding the network adapter and accessing the device's signals directly. This approach allows the output from different sensor types covering the same terrain to be compared through algorithms that look at the location, size, speed, and even the surmised intent of the entity; it also allows these virtual security system elements to watch each sensor, achieve consensus in what may be in the field of view, and detect when a device is not functioning properly due to hardware malfunction or perhaps even a cyber exploit such as the one planned in our fictitious scenario.

As the diversion attack starts, the alarm station is rightfully caught off guard, but not for long. As the attackers cross the sensing fields, the AC&D system continues to report nothing interesting; however, sensor events are relayed directly from the digital twin to secure phones carried by the security forces and to the AR-enabled, truck-mounted displays. With a heads-up on suspicious activity on the south side of the facility, security forces discover the breaching team as they cross the fence. Meanwhile, the momentarily bewildered alarm station crew also using the status updates provided by the site's twin begins to piece together the situation. The consensus mechanisms built into the twin's virtual agents quickly identify the AC&D system as "odd man out" by not showing an alerted state. Security forces are already executing the defense plan on the south side of the perimeter through radio communications and twin updates. By the time the main attack has initiated, the site security forces have established their response rhythm, and, thanks in part to the twin, this attack is no surprise.

The site's digital twin has more than sensor data at its disposal. The terrain, buildings, fences, and other features have been recorded with a laser scanner. This information includes exact detail of the facility. Every berm, depression, and natural cover from vegetation on all pathways to and from the facility has been enumerated, along with their ease of traversal. Further, the digital twin has participated in and been informed by simulated and live training exercises allowing its artificial intelligence to learn over time. The twin had already proven to be a formidable opponent when training security forces—all the while training its own algorithms. The security forces have used AR and digital twin artificial intelligence to train with "what-if" scenarios involving virtual assets. In this scenario the twin's knowledge was used in real time to suggest maneuver routes and courses of action to the security forces. Those suggested moves were optimized to bring the breach quickly to an end while at the same time ensuring minimal damage. The site's digital twin provided the security personnel with knowledge overmatch.

5 Toward Digital Twin

The preceding section foot stomps how intelligent digital twin technology will ultimately provide improved model-based situational awareness for a variety of cyber-physical security systems to include facilities, military installations, mobile security command posts, and next-generation physical security incorporating AR. The digital twin, as a learning system, learns from itself—using sensor data that convey various aspects of its operating condition. Sensor data can come from (1) human experts, such as engineers with deep and relevant industry domain knowledge, (2) from other similar machines or fleets of machines, and (3) from the larger system and environment of which it may be a part. A digital twin integrates historical data from past machine usage into its digital model.

As a way of introducing how improved models via digital twins and AR representations can improve situational awareness, we embellished one of the most intriguing public hacks against a private company and described how (an albeit futuristic) digital twin could have played a role in a provocative view of the future. While the addition of digital twin technology could greatly enhance cyber-physical security systems, it may also present significant challenges for physical security systems and personnel.

For example, a survey of different-sized companies found that many organizations are not prepared for modern technical challenges. Digital twins present unique modern, socio-technical challenges. The authors of the survey concluded that to face modern technical challenges, security systems must be "supported by educated, informed, well-equipped personnel that grow their skill sets over time" ([16], p. 30). Additionally, according to Gregory-Brown and Wylie [16], safeguarding remains an important cyber-physical security issue:

> . . . reliance on control systems continues to expand across not only industrial settings, but also the operation and maintenance of our cities, our buildings and all kinds of modern smart applications. Recognition that even dedicated, special-purpose ICS components, such as intelligent embedded devices and programmable devices that are used for command and control, can carry vulnerabilities exploitable by malefactors is increasing among ICS security practitioners and the broader security community, as is concern about ransomware, which has started to invade the corners of almost any digital system.

Stamp and others [17] echo the call for training and education of security personnel, as S&T moves toward automation and the inclusion of intelligent technologies for physical security systems and facilities. They underscore the potential for inadequately trained personnel to cause security deficiencies, especially if they interact with automation.

The Ukraine power grid hack should serve as an example of how important it is to mitigate cyber-physical security system vulnerabilities across the board. Several more recent attacks use similar tools as those on the Ukraine power grid [18]. Clem and others [7] recommend utilizing LVC (live, virtual, constructive) model-based situational awareness and simulation to test cyber exploitation of control systems and physical security systems.

In addition to next-generation approaches to using testbeds or LVC to validate intrusion detection systems or approaches against simulated attacks, Yang et al. [19] proposed a multilayer approach to security that utilizes intelligent, electronic devices that initiate alerts toward self-recovery without human intervention, a part of the system's resilience. Another challenge will be keeping the security, reliability, and integrity of machine learning algorithms intact [20] as organizations and security personnel alike begin to trust and depend more heavily on the predictions and recommendations made by intelligent digital twin technology.

Finally, the National Academies found that the electric grid of the United States is vulnerable to a number of attacks, among them cyber [21]. In the report, the US Departments of Energy and Homeland Security are urged to work together to address the vulnerabilities. As far as the impact—it can be far reaching—according to Morgan, a professor of engineering at Carnegie Mellon University and chair of the committee, "long-duration outages that leave millions without power could result in economic damages estimated in the billions of dollars, posing serious threats to health and public safety, and also potentially compromising national security" [21]. Strengthening our cyber-physical security systems for US critical infrastructure with cyber countermeasures remains an evolving challenge for the federal government, private enterprise, and the public that must be addressed collaboratively.

6 Conclusion

In the highly VUCA environment that constitutes cyber-physical security systems and security operations, training is obsolete as soon as it is deployed. A survey of service strategy documents conducted by the first author highlights the shared

belief of the need for training and education modernization and some congruence on how to achieve it. Modernization will require much more realistic scenarios utilizing robust models, emulated software and hardware environments, and XR simulations, with adaptive, persistent, and blended live, virtual, constructive, and gaming environments [5, 6, 8, 22]. The use of AR, model-based situational awareness, and digital twins will greatly enhance the future of training personnel with immersive simulation. According to Machi [22], AR and XR could be combined with artificially intelligent avatars who serve as instructors to train personnel in a number of maneuvers. As cyber-physical security systems become model-based and leverage extended (XR) reality, the gaps between training, planning/analysis, and situational awareness simulations disappear. In the present chapter, we discussed the notion of digital twin in the context of physical security system facilities. We applied this notion to a hypothetical scenario loosely based on actual events. We concluded with a discussion on the challenges that will be encountered as S&T moves toward digital twin technology and offered general recommendations for next-generation physical security systems.

Acknowledgments Sandia National Laboratories is a multimission laboratory managed and operated by the National Technology and Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International, Inc., for the US Department of Energy's National Nuclear Security Administration under contract DE-NA-0003525.

References

1. R. Trechter, *Physical security simulation and analysis tools*. SAND 2014-3718P (Sandia National Laboratories, Albuquerque, NM, 2014). http://umbra.sandia.gov/pdfs/resources/danteopshed.pdf. Accessed 17 Apr 2018
2. D. Callow, *SAND2016–12214 physical security system of the future: vision and roadmap* (Sandia National Labs, 2016)
3. M. W. Grieves, *Digital twin: Manufacturing excellence through virtual factory replication*. Digital Twin whitepaper. Michael W. Grieves, LLC (2014). http://www.apriso.com/library/Whitepaper_Dr_Grieves_DigitalTwin_ManufacturingExcellence.php. Accessed 17 Apr
4. General Electric Research, *Predix technology brief: Digital Twin* (2017). https://www.predix.com/sites/default/files/predix-digital-twin-technology.pdf. Accessed 17 Apr 2018
5. E.M. Raybourn, A new paradigm for serious games: Transmedia learning for more effective training & education. J. Comput. Sci. **5**(3), 471–481 (2014)
6. E.M. Raybourn, Applying simulation experience design methods to creating serious game-based adaptive training systems. Interact. Comput. **19**, 207–214 (2007)
7. J. Clem, W. Atkins, V. Urias, *Investigation of cyber-enabled physical attack scenarios*. SAND2015-4202C (Sandia National Laboratories, Albuquerque, NM, 2015). https://www.osti.gov/scitech/servlets/purl/1255768. Accessed 6 Mar 2018
8. E.M. Raybourn, in *A metaphor for immersive environments: Learning experience design challenges and opportunities. Proceedings of MODSIM*. (NTSA, Arlington, VA, 2016)
9. M.L. Garcia, *The Design and Evaluation of Physical Protection Systems*, 2nd edn. (Butterworth-Heinemann, Newton, MA, 2007)

10. K. Klinger, D. Small, E. Gottlieb, J. Whetzel, H. Gillis, J. Wharton, *Final report for advanced high security command and control interface LDRD (AHSC2I)*. SAND2013–8249 (Sandia National Laboratories, Albuquerque, NM, 2013)
11. L. Dignan, *GE aims to replicate digital twin success with security-focused digital ghost*. Between the lines, ZDNet.com (2017). https://www.zdnet.com/article/ge-aims-to-replicate-digital-twin-success-with-security-focused-digital-ghost/. Accessed 17 Apr 2018
12. J. Russel, J. Andersen., C. Sterns, *Video motion detector fused radar: the first volumetric ultra-low NAR sensor for exterior environments*. SAND2016–0083 (Sandia National Laboratories, Albuquerque, NM, 2016)
13. A. Greenberg, *How an entire nation became Russia's test lab for cyberwar*. Wired (2017, June). https://www.wired.com/story/russian-hackers-attack-ukraine/. Accessed 6 Mar 2018
14. ICS-CERT, *Cyber attack against Ukrainian critical infrastructure* (2016). https://ics-cert.us-cert.gov/alerts/IR-ALERT-H-16-056-01. Accessed 6 Mar 2018
15. R. Lipovski, A. Cherepanov, *BlackEnergy trojan strikes again: attacks Ukrainian electric power industry*. WeLiveSecurity (2016, Jan). https://www.welivesecurity.com/2016/01/04/blackenergy-trojan-strikes-again-attacks-ukrainian-electric-power-industry/. Accessed 6 Mar 2018
16. B. Gregory-Brown, D. Wylie, *Securing industrial control systems – 2017: A SANS survey* (SANS Institute InfoSec Reading Room, 2017) https://www.sans.org/reading-room/whitepapers/analyst/securing-industrial-control-systems-2017-37860. Accessed 6 Mar 2018
17. J. Stamp, J. Dillinger, W. Young, J. Depoy, *Common vulnerabilities in critical infrastructure control systems*. SAND2003-1772C (Sandia National Laboratories, Albuquerque, NM, 2003)
18. K. J. Higgins, *Latest Ukraine blackout tied to 2015 cyberattackers*. Dark Reading (2017, Jan). https://www.darkreading.com/threat-intelligence/latest-ukraine-blackout-tied-to-2015-cyberattackers/d/d-id/1327863. Accessed 6 Mar 2018
19. Y. Yang, K. McLaughlin, S. Sezer, T. Littler, E.G. Im, B. Pranggono, H.F. Wang, Multiattribute SCADA-specific intrusion detection system for power networks. IEEE Trans. Power Deliv. **29**(3), 1092–1102 (2014)
20. E. M. Raybourn, N. Fabian, W. Davis, R. C. Parks, J. McClain, D. Trumbo, D. Regan., P. Durlach, Data privacy and security considerations for personal assistants for learning (PAL). *Proceedings of the 20th International Conference on Intelligent User Interfaces Companion* (2015), pp. 69–72
21. R. Walton, *National academies report finds grid vulnerable to cyber physical attacks*. UtilityDive (2017, July 24). http://www.utilitydive.com/news/national-academies-report-finds-grid-vulnerable-to-cyber-physical-attacks/447707/. Accessed 6 Mar 2018
22. V. Machi, *The future of training and simulation: preparing warfighters for tomorrow's battlefields*. National Defense Magazine (2017, Nov). http://www.nationaldefensemagazine.org/articles/2017/11/22/the-future-of-training-and-simulation-preparing-warfighters-for-tomorrows-battlefields. Accessed 29 Apr 2018