Ubiquitous Verification of Ubiquitous Systems*

Reinhard Wilhelm and Matteo Maffei

Universität des Saarlandes, Saarbrücken, Germany
{wilhelm,maffei}@cs.uni-saarland.de

Abstract. Ubiquitous embedded computing systems expected to reliably perform one or more relevant tasks need design and verification methods currently not available. New envisioned applications and trends in system design increase this need. Several of these trends, e.g. function integration, concurrency, energy awareness, networking and their consequences for verification are considered in this article. It is described that, already in the past, verification was made possible only due to rules restricting the design and it is argued that even more so in the future the constructive influence on the design of hardware and software will be a necessary condition to keep the verification task tractable.
Keywords: embedded systems, verification, security, networking, timing analysis, WCET, hard real-time, timing predictability

1 Introduction

Ubiquitous embedded systems often combine an impressive number of requirements: They should be functionally correct as most other computing systems. They often need to react in real time. They should save energy if they are mobile, and they need to be secure if tampering is possible through an interface or a network. This article considers the verification challenge for such a combination of requirements. This challenge is quite formidable! Several of the individual properties are already very hard to verify. This is witnessed by undecidability and by complexity results for some of the tasks. An escape has always been to resort to simplified settings or to heuristic methods. However, these simplified problems were still hard enough. A modular approach to the challenge of verifying systems with such a combination of required properties seems to be the only solution. However, the interdependence between the different properties does not easily allow this. This is already clear from the example of timing validation where the interdependence of different architectural components introduces timing anomalies [35, 43] and forces any sound timing analysis to analyze a huge architectural state space.

* The research reported herein was supported by the European Network of Excellence *ArtistDesign*, the Deutsche Forschungsgemeinschaft in SFB/TR 14 *AVACS*, the European Community's Seventh Framework Programme FP7/2007-2013 under grant agreement number 216008 (*Predator*), the initiative for excellence and the Emmy Noether program of the German federal government, and by Miur'07 Project SOFT (*Security Oriented Formal Techniques*)'.

This article presents an analysis of past developments and of current trends with their implications for verification. It attempts to convey the message that *the trends in computer and software architecture and in system development increase the need for and the complexity of system verification*, and that *constructive influence on the design of hardware and software has to be exercised in order to keep the verification task tractable.*

The structure of this article is as follows. The development of the timing-analysis problem and the methods to solve it are presented in Section 2. It is shown how different notions of *state* evolved and how the state spaces to be analyzed exploded. Also, examples of how constructive influence was exercised to keep the problem tractable are given. In Sections 2.2 and 2.3, it is sketched how the introduction of concurrency and the need to save energy create new verification challenges. In Sections 2.4 and 3, two research directions are discussed in more detail. The first one is how to overcome difficulties in timing analysis, namely by making architectures timing predictable. The second concerns the security problems created by connecting embedded systems by networks. The state of the art in automatic verification is presented and open problems are given.

Several analogies between the two system properties, i.e., timing behavior and security, and the resulting verification problems are stressed:

- The security domain has seen correctness proofs of security protocols. However, they did not necessarily hold for their implementations since the proofs abstracted from essential system properties. Surprisingly, even proofs about the implementation on the source level may not be sufficient as witnessed in [44]. So, several levels have to considered for a total proof of security properties.
  Similarly, timing validation has been done on the specification level [16]. However, those prrofs don't carry over to the implementation level since they typically use unit-time abstraction, i.e., all transitions in the architecture take 1 unit of time. Neither can central parts of timing analysis be done on the source level since the source doesn't refer to the architecture.
- Compositionality of the resource behavior is the dream behind the AUTOSAR and IMA architecture movements in the automotive and avionics domains. So far, it has not been achieved.
  The security domain knows some compositionality results. However, they only hold under appropriate conditions.

2 Timing Analysis

Timing analysis of hard real-time systems is a good area to demonstrate how trends in application and system design have increased the pressure for sound verification methods and at roughly the same time the complexity of the verification task.

Figure 1 shows the development over time of the hardware and software architectures and the timing-analysis methods used in time-critical embedded

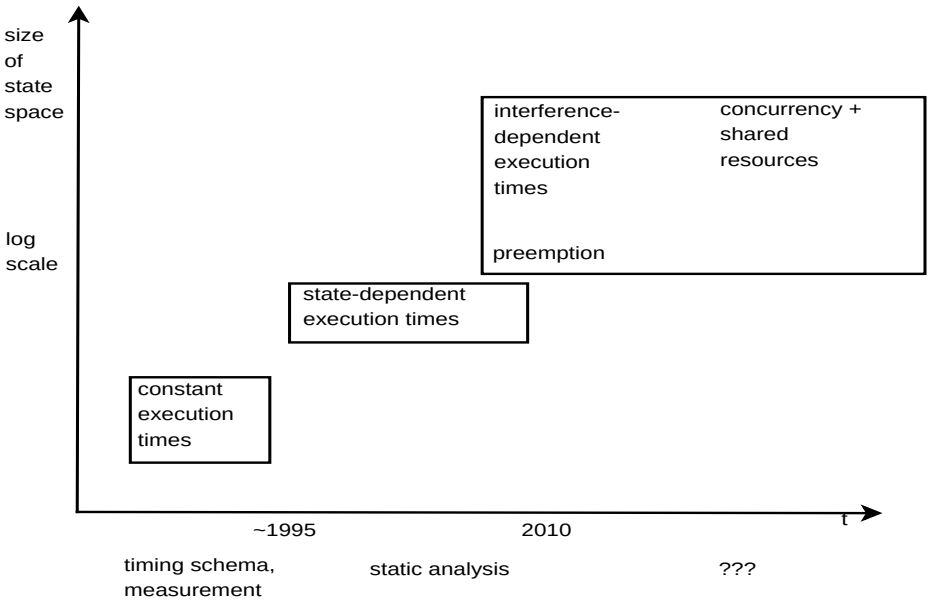

Fig. 1. Timing analysis — increasing the complexity of the architectures increases the complexity of the verification problems and makes established methods obsolete.

systems. Each transition extended the state space to be explored, often adding one more dimension and leading to an exponential growth of the state space. We will see in the course of the article that the development trends led to different notions of *state*.

Until the 90s, processor architectures used in time-critical embedded systems had instruction-execution times that were mostly independent of the execution state. The small onboard cache if any was often switched off. The processor handbook would give the (constant) number of cycles each instruction would take to execute. The most popular methods for timing analysis in these times were measurement und the so-called *timing schema* approach [46, 39], sometimes in combination. Exhaustive measurement could be used if the input domain was sufficiently small. The state space to search was determined by the input domain. The transitions through the space of execution states of the architecture were abstracted away by the constant execution times. The timing-schema approach used induction over the structure of programs to compute upper timing bounds of programs from bounds of components.

2.1 State-dependent execution times

EADS Airbus used the Motorola ColdFire processor in its A340 planes, a processor with a unified data/instruction cache of 8K size with a pseudo-round-robin replacement policy. This cache turned out to have very bad predictability prop-

erties [28]. Static-analysis based methods were still able to determine suffciently precise bounds on the execution times [21]. The execution time of instructions depended strongly on the (architectural) state in which the instructions were executed. The state space to explore became the cartesian product of the data domain and the architectural state space. This was clearly too large to be exhaustively covered by measurement-based methods. Timing schemas could not easily be used any more since combining worst-case timings would produce too pessimistic results. Resorting to higher order schemas by adding a state parameter and combining timings with matching states avoided this pitfall [34].

It helped that task sets were expected to be scheduled non-preemptively. So, bounds on the execution time were determined for uninterrupted task execution. No interference from outside of the task needed to be considered. This was sufficient for many safety-critical avionics systems as they were typically synthesized from SCADE models [48]. We consider this as an example, where problem-aware developers of safety-critical systems avoided creating a problem that was not solvable by methods existing at the time.

At least in research, preemptive execution was considered [32, 33]. However, it was clear that, for complexity reasons, the necessary determination of context-switch costs could not be performed at every program point. Delayed preemption was mostly assumed to keep the effort for the determination of program-point-specific context-switch costs tolerable. This, we consider as a second example for how constructive influence was taken to keep complexity within bounds.

Methods based on this work were improved, so that they are now finally mature enough to be interfaced to schedulability analyses [3].

Preemptive scheduling introduced a timing-analysis problem where the interference between tasks influenced the tasks' execution times. Another type of interference creates more severe problems with the advent of multi-processor/multi-core architectures and multi-threading software.

2.2 Concurrency

Future embedded system will be executed on manycore/multicore systems and will, thus, enjoy all performance, energy, and reliability advantages of these platforms. Several functions will be integrated on such platforms as witnessed in the AUTomotive Open System ARchitecture (AUTOSAR)[1] and the Integrated Modular Avioics (IMA) [49] examples.

However, this step also increases the verification pressure and increases the state space in another dimension by admitting potentially many different interleavings of the concurrent tasks' executions. Proving the functional correctness of concurrent systems is still a challenge. The IMA standard imposes *temporal* and *spatial partitioning* of the functions integrated on one execution platform. This eases the verification of functional correctness by avoiding the interference of tasks on a global state represented in the platforms memory hierarchy. This is another example of how a system-design discipline eases the verification task.

[1] see www.autosar.org

However, it does not prevent the interference on shared resources regarding their architectural state, i.e. their occupancy. It therefore does not support *compositionality* of the resource behavior. The performance of a function implemented on the platform may change when another function's implementation is replaced. This undermines the envisioned *incremental qualification. Resource isolation* of concurrently running tasks is one means to achieve this as will be later argued.

2.3 Power Awareness

Ubiquitous embedded systems whenever they are parts of mobile applications need to be designed in a power-aware fashion. Dynamic voltage scaling is often applied to reduce energy consumption [29]. Turning down the processor frequency has an impact on the performance and, thus, interacts with timing analysis. [45] describes an approach to timing analysis in combination with dynamic frequency scaling. However, the described method only works with simple processor architectures.

2.4 Predictability

One way to alleviate the timing-analysis task is to increase the predictability of the underlying architecture. This should be done on the single-core level [50] and, even more importantly, on the multi-core level [15]. The *predictability* notion has been around for a while [47], however, without a formal foundation.

Design for predictability of architectures is a very active area of current research. However, there is no agreement on the notion of predictability. The strictest notion requires execution-state independent timing behaviour of instructions. This direction is represented by the newly appeared XCore platform of XMOS[2] and by the PRET project [20]. Both do not use caches in their design as caches introduce a large variability of execution times. It is so far unclear how much performance has to be sacrificed.

The MERASA project concentrates on making architectural components more predictable. [37] describes architectural support for predictable symmetric multithreading. [38] presents a multi-core design enforcing a bounded delay on the access to a shared bus.

Our conception of the predictability of architectures admits architectural components introduced to increase average-case performance as long as the variability of their behaviour can be precisely and efficiently bounded by static analysis for a given software.

[42, 41] have given precise notions for predictability of different cache replacement strategies (LRU, PLRU, FIFO, MRU). This work was the first to formally define cache predictability as *speed of recovery from uncertainty* and to rigorously compare different replacement policies. Similarly, the *sensitivity* to the

[2] https://www.xmos.com/products

initial state of cache performance for different replacement policies has been investigated. Cache performance under non-LRU policies are extremely sensitive to the initial cache state. Therefore, measurement-based approaches can yield results which are significantly lower than the actual WCET. The results led to the conclusion that LRU is superior to any other considered replacement policy and this under different criteria: performance, predictability, and therefore expected precision of cache analysis, and sensitivity. In addition, all approaches to determine the cache-related preemption delay only work for LRU.

[50] examines the relation between performance and analysis effort for the design of pipelines and buses and derived design guidelines.

[15] identifies *design rules for predictable multi-processors*. The first principles are to avoid interference on shared resources in the architecture and to allow the application designer to map applications to the target architecture without introducing new interferences that were not present in the application. This is because *interference by sharing* resources such as buses and caches is the main obstacle towards timing analysis for multi-core architectures. Thus, removal of sharing is the key to predictability. For costly and infrequently accessed resources such as I/O devices, interference costs can be bounded by imposing deterministic access protocols. This doesn't introduce much overestimation due to the generally low utilisation. Additional sharing (e.g., to meet cost constraints) is allowed if safe and sufficiently small delays for the access to shared resources can be guaranteed.

Compositionality Compositionality of system properties is extremely important as it allows the component-wise verification of a system with a guarantee of the property for the whole system. Expectations towards compositionality of the predicted resource behavior, however, are misleading. Full compositionality can not be expected as the resources in embedded systems are bounded. All that can be expected is compositionality of the resource behavior given an unlimited supply. Hence, two proof obligations arise. Firstly, that one component's resource behaviour does not change that of another component assuming unbounded resources, and secondly, that the overall resource requirements are satisfiable. Incremental qualification faces the same limitation.

3 Networking and Security

3.1 Security issues in networked embedded systems

One of the distinctive features of modern embedded systems is the support for networking. This is motivated by the increasing demand of personalized services and collaborative platforms. Just to mention some examples, vehicular ad-hoc networks allow drivers and passengers to communicate with each other as well as with the roadside infrastructure in order to send and receive warnings about traffic jam, incidents, queues, and so on; some household thermostats offer Internet connectivity to let the owner switch on the heating a certain time before the arrival; some hospitals use wireless networks for patient care equipment.

Although networking paves the way for the development of services that were not imaginable a few years ago, it also poses serious security issues. For instance, even the notoriously quoted refrigerator who, connected to the internet to order food and drinks, poses such a risk. Who would be happy to find all of ALDI's beer supply infront of one's door when a hacker enjoyed playing a nice joke? More seriously, security and trust mechanisms have to be introduced in vehicular ad-hoc networks in order to protect the privacy of drivers and to prevent malicious drivers or corrupted devices from broadcasting false warnings [40]; similarly, access control rules have to be enforced in order to ensure that only the house owner can control the thermostat therein [30]; perhaps more surprisingly, it has recently been shown that several attacks can be mounted on implantable cardioverter defibrillators, compromising patient safety and patient privacy, up to inducing electrical shocks into the patient's heart [27]! These attacks have been discovered by applying reverse engineering and eavesdropping techniques to a previously unknown radio communication protocol.

From this perspective, the recent trend to introduce wired and wireless networks on planes and to rely on software solutions to ensure the intended safety and security guarantees further witnesses the dramatic need of automated verification tools for the security of networked embedded systems. A FAA (Federal Aviation Administration) document dating back to 2008 points out weaknesses in the communication network within the Boeing's new 787 Dreamliner. The proposed architecture of the 787," the FAA stated, "allows new kinds of passenger connectivity to previously isolated data networks connected to systems that perform functions required for the safe operation of the airplane...The proposed data network design and integration may result in security vulnerabilities from intentional or unintentional corruption of data and systems critical to the safety and maintenance of the airplane" [2].

3.2 Automated verification of cryptographic protocols

The security model that should be taken into account in the analysis of networked embedded systems comprises internal attackers (i.e., malicious users and corrupted devices) as well as external attackers. The security desiderata are application dependent and may include the secrecy and integrity of data, access control policies, trust policies, and user anonymity.

Cryptographic protocols constitute core building blocks for designing systems that stay secure even in the presence of malicious entities. The design of security protocols has long been known to be a challenging task, which is even more challenging in the context of embedded systems since the limited amount of available resources often rules out the possibility to employ expensive, powerful cryptographic operations (e.g., zero-knowledge proofs and secure multiparty computations). Even in heavily simplified models where the complexity of cryptographic operations is abstracted away and cryptographic messages are modelled as symbolic terms, security properties of cryptographic protocols are in general undecidable. Additionally, security analyses of such protocols are awkward to make for humans, due to the complexity induced by multiple interleaved

protocol runs and the unpredictability of the attacker behavior. Formal methods, and in particular static analysis techniques such as type systems [1, 24, 14, 5, 7, 9], abstract interpretation [23, 12, 13, 6], and theorem proving [11] proved to constitute salient tools for reliably analyzing security protocols. Nowadays, the analysis of sophisticated properties, such as anonymity, privacy, and access control policies, is within the scope of automated verification tools and the running time for such analyses ranges from a few seconds to a couple of hours, depending on the complexity of the protocol and of the cryptographic primitives.

3.3 Towards a security analysis of embedded systems

Despite these promising results, however, the verification of security properties of networked embedded systems is still an open issue. The main reason is that the aforementioned automated analysis techniques focus on the logic of the protocol (i.e., the way cryptographic messages are exchanged) and tend to abstract away from its implementation. Consequently, a gap often exists between the verified protocol models and the actually deployed implementations. Hence, even if the abstract model is proved to be safe, security flaws may still affect its implementation [10]. Only recently, some works have tackled the analysis of the source code of protocol implementations, with a specific focus on functional languages [10, 9] and C code [25, 17]. Still the semantics of the programming languages is idealized (e.g., by encoding in the lambda calculus) and the verified models abstract a number of potentially troublesome details. Recent papers have touched the security analysis of bytecode, hardware language, and timing behavior, but they mainly focused on information flow properties and non-concurrent code [31, 22, 8]. The verification of distributed implementations of security protocols is still an open issue and it has been recognized as the grand challenge of the next twenty years at the 21st Computer Security Foundation Symposium (CSF'08).

The security dimension of computing systems is not independent of the other dimensions listed so far. The secrecy of data, in particular, is a "local" property crucially depending on implementation and hardware details. The observation of the timing behavior and of the energy consumption of program parts, for instance, can offer covert channels to leak private data [51]. Higher-level security properties (e.g., authentication, user anonymity, and distributed access control) are however typically "global": they build on top of the secrecy of some data, such as keys, passwords, and credentials, but for the rest they solely depend on the messages exchanged on the network. An interesting research direction that is worth to be explored, consequently, is the automated verification of secrecy on detailed hardware models and the investigation of compositionality results ensuring that global security properties that are verified on abstract models carry over to the actual implementation, as long as this locally preserves the expected secrecy properties and complies with the intended communication protocol.

3.4 Design guidelines to simplify the analysis

A careful architecture design may strengthen the security of the system and help to reduce the complexity of the analysis. Here we discuss two important aspects, namely, *hardware solutions* and *compositionality principles*.

Hardware solutions, in conjunction with software-based security mechanisms, enhance the performance and ensure the correctness of basic cryptographic operations, thus improving the security of the system and reducing the complexity of the analysis. By way of example, the secrecy of sensible data, which as discussed above constitutes the building block of higher level security properties, can be enforced by using dedicated cryptographic chips, such as the trusted platform modules (TPMs). These cryptographic chips facilitate the secure generation of cryptographic keys, allow for securely storing these keys, and offer support for the efficient implementation of a number of cryptographic operations, including advanced schemes such as zero-knowledge proofs. Nowadays, TPMs are included in most high level laptops and their applications include secure disk encryption, password protection, and digital right management.

Security protocols in general do not enjoy compositionality properties: If we consider two protocols that are secure when executed in physically separated systems, we are not guaranteed that their concurrent execution in the same environment achieves the same security properties. Nevertheless, there exist design principles that can be followed to obtain compositionality guarantees. These principles include the usage of distinct cryptographic keys or disjoint encryption schemes [26], the tagging of cryptographic messages [36, 18], and certain patterns enforced by compositional analysis techniques [14, 19, 4]. Protocol compositionality is a crucial aspect in the analysis of cryptographic protocols as it allows for the independent verification of each single component, thus significantly reducing the state space, yet obtaining in the end security guarantees for the system as a whole.

4 Conclusions

We have discussed several required properties of ubiquitous embedded systems and the resulting verification problem, highlighting open issues and suggesting directions of future research. Security and timing requirements were discussed in more depth. It was described how design methods and compositionality properties have helped to keep the verification problem tractable and argued that this will also hold in the future.

References

1. M. Abadi. Secrecy by typing in security protocols. *Journal of the ACM*, 46(5):749–786, 1999.
2. F. A. Administration. Federal Register: January 2, 2008 (Volume 73, Number 1).

3. S. Altmeyer, C. Maiza, and J. Reineke. Resilience analysis: tightening the CRPD bound for set-associative caches. In J. Lee and B. R. Childers, editors, *LCTES*, pages 153–162. ACM, 2010.
4. S. Andova, C. Cremers, K. Gjøsteen, S. Mauw, S. F. Mjølsnes, and S. Radomirović. A framework for compositional verification of security protocols. *Information and Computation*, 206(2-4):425–459, 2008.
5. M. Backes, A. Cortesi, R. Focardi, and M. Maffei. A calculus of challenges and responses. In *Proc. 5rd ACM Workshop on Formal Methods in Security Engineering (FMSE)*, pages 101–116. ACM Press, 2007.
6. M. Backes, A. Cortesi, and M. Maffei. Causality-based abstraction of multiplicity in cryptographic protocols. In *Proc. 20th IEEE Symposium on Computer Security Foundations (CSF)*, pages 355–369. IEEE Computer Society Press, 2007.
7. M. Backes, C. Hriţcu, and M. Maffei. Type-checking zero-knowledge. In *15th Proc. ACM Conference on Computer and Communications Security*, pages 357–370. ACM Press, 2008.
8. G. Barthe, D. Pichardie, and T. Rezk. A certified lightweight non-interference java bytecode verifier. In *Proc. 16th European Symposium on Programming (ESOP)*, Lecture Notes in Computer Science, pages 125–140. Springer-Verlag, 2007.
9. J. Bengtson, K. Bhargavan, C. Fournet, A. D. Gordon, and S. Maffeis. Refinement types for secure implementations. In *CSF '08: Proceedings of the 2009 21st IEEE Computer Security Foundations Symposium*, pages 17–32. IEEE Computer Society, 2008.
10. K. Bhargavan, C. Fournet, A. D. Gordon, and S. Tse. Verified interoperable implementations of security protocols. In *Proc. 19th IEEE Computer Security Foundations Workshop (CSFW)*, pages 139–152. IEEE Computer Society Press, 2006.
11. B. Blanchet. An efficient cryptographic protocol verifier based on Prolog rules. In *Proc. 14th IEEE Computer Security Foundations Workshop (CSFW)*, pages 82–96. IEEE Computer Society Press, 2001.
12. C. Bodei, M. Buchholtz, P. Degano, F. Nielson, and H. R. Nielson. Static validation of security protocols. *Journal of Computer Security*, 13(3):347–390, 2005.
13. Y. Boichut and T. Genet. Feasible trace reconstruction for rewriting approximations. In *Term Rewriting and Applications, 17th International Conference (RTA 2006)*, pages 123–135, 2006.
14. M. Bugliesi, R. Focardi, and M. Maffei. Dynamic types for authentication. *Journal of Computer Security*, 15(6):563–617, 2007.
15. C. Burguiere, D. Grund, J. Reineke, R. Wilhelm, C. Cullmann, C. Ferdinand, G. Gebhard, and B. Triquet. Predictability considerations in the design of multi-core embedded systems. In *Embedded Real Time Software and Systems (ERTSS)*, 2010.
16. S. V. A. Campos, S. Vale, A. Campos, M. Gerais, B. Horizonte, and E. Clarke. Analysis and verification of real-time systems using quantitative symbolic algorithms. *Journal of Software Tools for Technology Transfer*, 2:260–269, 1999.
17. S. Chaki and A. Datta. Aspier: An automated framework for verifying security protocol implementations. In *CSF '09: Proceedings of the 2009 22nd IEEE Computer Security Foundations Symposium*, pages 172–185. IEEE Computer Society, 2009.
18. V. Cortier and S. Delaune. Safely composing security protocols. *Formal Methods in System and Design*, 34(1):1–36, 2009.
19. A. Datta, A. Derek, J. Mitchell, and A. Roy. Protocol composition logic (pcl). *Electronic Notes on Theoretical Computer Science*, 172:311–358, 2007.

20. S. A. Edwards and E. A. Lee. The case for the precision timed (pret) machine. In *DAC*, pages 264–265. IEEE, 2007.
21. C. Ferdinand, R. Heckmann, M. Langenbach, F. Martin, M. Schmidt, H. Theiling, S. Thesing, and R. Wilhelm. WCET Determination for a Real-Life Processor. In T. Henzinger and C. Kirsch, editors, *Embedded Software (EMSOFT 2001)*, volume 2211 of *Lecture Notes in Computer Science*, pages 469 – 485. Springer, 2001.
22. S. Genaim and F. Spoto. Information flow analysis for java bytecode. In *Verification, Model Checking, and Abstract Interpretation*, pages 346–362. Springer-Verlag, 2005.
23. T. Genet and V. Tong. Reachability analysis of term rewriting systems with timbuk. In *Proc. Artificial Intelligence on Logic for Programming (LPAR '01)*, pages 695–706. Springer-Verlag, 2001.
24. A. D. Gordon and A. Jeffrey. Types and effects for asymmetric cryptographic protocols. *Journal of Computer Security*, 12(3):435–484, 2004.
25. J. Goubault-Larrecq and F. Parrennes. Cryptographic protocol analysis on real C code. In *Proc. 6th International Conference on Verification, Model Checking and Abstract Interpretation (VMCAI'05)*, volume 3385 of *Lecture Notes in Computer Science*, pages 363–379. Springer-Verlag, 2005.
26. J. D. Guttman and F. J. Thayer. Protocol independence through disjoint encryption. In *Proc. 13th IEEE Computer Security Foundations Workshop (CSFW)*, pages 24–34. IEEE Computer Society Press, 2000.
27. D. Halperin, T. S. Heydt-Benjamin, B. Ransford, S. S. Clark, B. Defend, W. Morgan, K. Fu, T. Kohno, and W. H. Maisel. Pacemakers and implantable cardiac defibrillators: Software radio attacks and zero-power defenses. In *SP '08: Proceedings of the 2008 IEEE Symposium on Security and Privacy*, pages 129–142, Washington, DC, USA, 2008. IEEE Computer Society.
28. R. Heckmann, M. Langenbach, S. Thesing, and R. Wilhelm. The Influence of Processor Architecture on the Design and the Results of WCET Tools. *IEEE Proceedings on Real-Time Systems*, 91(7):1038–1054, 2003.
29. W. Kim, J. Kim, and S. Min. A dynamic voltage scaling algorithm for dynamic-priority hard real-time systems using slack time analysis. In *DATE '02: Proceedings of the conference on Design, automation and test in Europe*, page 788, Washington, DC, USA, 2002. IEEE Computer Society.
30. P. Koopman. Embedded system security. *Computer*, 37(7):95–97, 2004.
31. B. Köpf and D. A. Basin. An information-theoretic model for adaptive side-channel attacks. In *Proc. 14th ACM Conference on Computer and Communications Security*, pages 286–296, 2007.
32. C. Lee, J. Han, Y. Seo, S. Min, R. Ha, S. Hong, C. Park, M. Lee, and C. Kim. Analysis of Cache-Related Preemption Delay in Fixed-Priority Preemptive Scheduling. In *Proceedings of the IEEE Real-Time Systems Symposium*, December 1996.
33. S. Lee, C.-G. Lee, L. M., S. L. Min, and C. S. Kim. Limited Preemptible Scheduling to Embrace Cache Memory in Real-Time Systems. In *Proceedings of the ACM SIGPLAN LCTES'98 Workshop on Languages, Compilers and Tools for Embedded Systems*, pages 51–64, June 1998.
34. S.-S. Lim, Y. H. Bae, G. T. Jang, B.-D. Rhee, S. L. Min, C. Y. Park, H. Shin, K. Park, S.-M. Moon, and C. S. Kim. An accurate worst case timing analysis for RISC processors. *IEEE Transactions on Software Engineering*, 21(7):593–604, July 1995.
35. T. Lundquist and P. Stenström. Timing Anomalies in Dynamically Scheduled Microprocessors. In *20th IEEE Real-Time Systems Symposium*, 1999.

36. M. Maffei. Tags for multi-protocol authentication. In *Proc. 2nd International Workshop on Security Issues in Coordination Models, Languages, and Systems (SECCO '04)*, Electronic Notes on Theoretical Computer Science, pages 55–63. Elsevier Science Publishers Ltd., 2004.
37. J. Mische, I. Guliashvili, S. Uhrig, and T. Ungerer. How to enhance a superscalar processor to provide hard real-time capable in-order smt. In C. Müller-Schloer, W. Karl, and S. Yehia, editors, *ARCS*, volume 5974 of *Lecture Notes in Computer Science*, pages 2–14. Springer, 2010.
38. M. Paolieri, E. Quiones, F. J. Cazorla, G. Bernat, , and M. Valero. Hardware support for wcet analysis of hrt multicore systems. In *The 36th International Symposium on Computer Architecture (ISCA 2009)*, 2009.
39. P. Puschner and C. Koza. Calculating the maximum execution time of real-time programs. *Real-Time Systems*, 1:159–176, 1989.
40. M. Raya and J.-P. Hubaux. The security of vehicular ad hoc networks. In *SASN '05: Proceedings of the 3rd ACM workshop on Security of ad hoc and sensor networks*, pages 11–21. ACM, 2005.
41. J. Reineke. *Caches in WCET Analysis*. PhD thesis, Universität des Saarlandes, Saarbrücken, 2008.
42. J. Reineke, D. Grund, C. Berg, and R. Wilhelm. Timing Predictability of Cache Replacement Policies. *Real-Time Systems*, 37(2):99–122, 2007.
43. J. Reineke, B. Wachter, S. Thesing, R. Wilhelm, I. Polian, J. Eisinger, and B. Becker. A definition and classification of timing anomalies. In *Proceedings of 6th International Workshop on Worst-Case Execution Time (WCET) Analysis*, July 2006.
44. T. W. Reps and G. Balakrishnan. Improved memory-access analysis for x86 executables. In L. J. Hendren, editor, *CC*, volume 4959 of *Lecture Notes in Computer Science*, pages 16–35. Springer, 2008.
45. K. Seth, A. Anantaraman, F. Mueller, and E. Rotenberg. Fast: Frequency-aware static timing analysis. *ACM Trans. Embed. Comput. Syst.*, 5(1):200–224, 2006.
46. A. C. Shaw. Reasoning About Time in Higher-Level Language Software. *IEEE Transactions on Software Engineering*, 15(7):875–889, 1989.
47. J. A. Stankovic and K. Ramamritham. Editorial: What is predictability for real-time systems? *Real-Time Systems*, 2(4):247–254, 1990.
48. S. Thesing, J. Souyris, R. Heckmann, F. Randimbivololona, M. Langenbach, R. Wilhelm, and C. Ferdinand. An Abstract Interpretation-Based Timing Validation of Hard Real-Time Avionics Software Systems. In *Proceedings of the Performance and Dependability Symposium, San Francisco, CA*, June 2003.
49. C. B. Watkins and R. Walter. Transitioning from federated avionics architectures to integrated modular avionics. In *26th Digital Avionics Systems Conference (DASC)*, 2007.
50. R. Wilhelm, D. Grund, J. Reineke, M. Pister, M. Schlickling, and C. Ferdinand. Memory hierarchies, pipelines, and buses for future time-critical embedded architectures. *IEEE TCAD*, 28(7):966–978, July 2009.
51. J. C. Wray. An analysis of covert timing channels. *Security and Privacy, IEEE Symposium on*, 0:2, 1991.