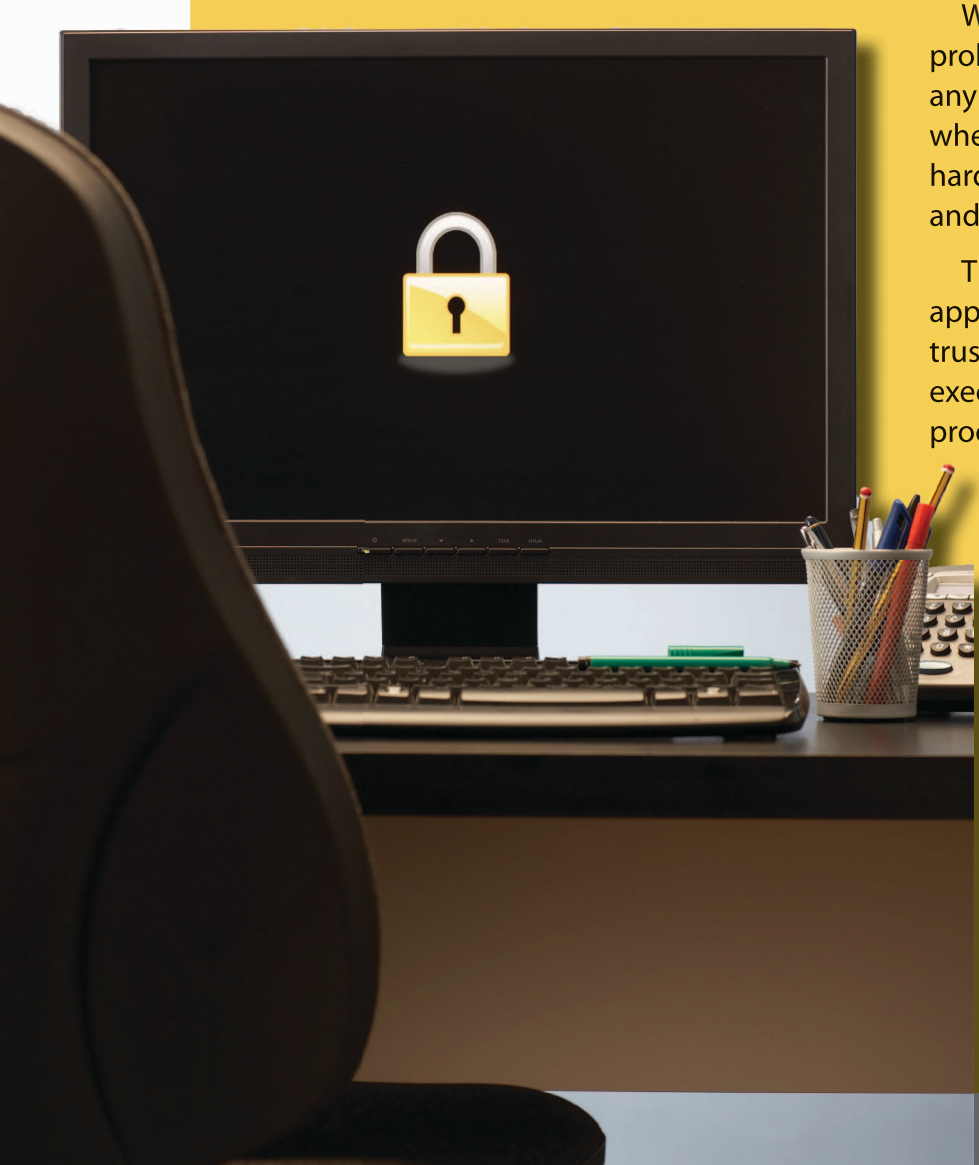
Proof-carrying data: Secure computation on untrusted platforms

Alessandro Chiesa and Eran Tromer

When running software applications and services, we rely on the underlying execution platform: the hardware and the lower levels of the software stack. The execution platform is susceptible to a wide range of threats, ranging from accidental bugs, faults, and leaks to maliciously induced Trojan horses. The problem is aggravated by growing system complexity and by increasingly pertinent outsourcing and supply chain consideration. Traditional mechanisms, which painstakingly validate all system components, are expensive and limited in applicability.



What if the platform assurance problem is just too hard? Do we have any hope of securely running software when we cannot trust the underlying hardware, hypervisor, kernel, libraries, and compilers?

This article will discuss a potential approach for doing just so: conducting trustworthy computation on untrusted execution platforms. The approach, proof-carrying data (PCD), circumnavigates the threat of faults and leakage by reasoning solely about properties of a computation's output data, regardless of the process that produced it. In PCD, the system designer prescribes the desired properties of the computation's outputs. These properties are then enforced using cryptographic proofs attached to all data flowing through the system and verified at the system perimeter as well as internal nodes.

1. Introduction

Integrity of data, information flow control, and fault isolation are three examples of security properties of which attainment, in the general case and under minimal assumptions, is a major open problem. Even when particular solutions for specific cases are known, they tend to rely on platform trust assumptions (for example, the kernel is trusted, the central processing unit is trusted), and even then they cannot cross trust boundaries between mutually untrusting parties. For example, in cloud computing, clients are typically interested in both integrity [1] and confidentiality [2] when they delegate their own computations to the untrusted workers.

Minimal trust assumptions and very strong certification guarantees are sometimes almost a basic requirement. For example, within the information technology supply chain, faults can be devastating to security [3] and hard to detect; moreover, hardware and software components are often produced in faraway lands from parts of uncertain origin where it is hard to carry out quality assurance in case trust is not available [4]. This all implies risks to the users and organizations [5, 6, 7, 8].

2. Goals

In order to address the aforementioned problems, we propose the following goal:

> **Goal.** A compiler that, given a protocol for a distributed computation and a security property (in the form of a predicate to be verified at every node of the computation), yields an augmented protocol that enforces the security property.

We wish this compiler to *respect the original distributed computation* (that is, the compiler should preserve the computation's communication graph, dynamics, and efficiency). This implies, for example, that *scalability* is preserved: If the original computation can be jointly conducted by numerous parties, then the compiler produces a secure distributed computation that has the same property.

3. Our approach

We propose a generic solution approach, proof-carrying data (PCD), to solve the aforementioned problems by defining appropriate checks to be performed on each party's computation and then letting parties attach proofs of correctness to each message. Every piece of data flowing through a distributed computation is augmented by a short proof string that certifies the data as compliant with some desired property. These proofs can be propagated and aggregated "on the fly," as the computation proceeds. These proofs may be between components of a single platform or between components of mutually untrusting platforms, thereby extending trust to any distributed computation.

But what "properties" do we consider? Certainly we want to consider the property that every node carried out its own computation without making any mistakes. More generally, we consider properties that can be expressed as a requirement that every step in the computation satisfies some *compliance predicate* C computable in polynomial time; we call this notion *C-compliance*. Thus, each party receives inputs that are augmented with proof strings, computes some outputs, and augments each of the outputs with a new proof string that will convince the next party (or the verifier of the ultimate output) that the output is consistent with a C-compliant computation. See figure 1 for a high-level diagram of this idea.

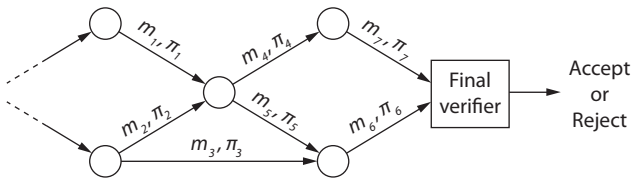


FIGURE 1. A distributed computation in which each party sends a message m_i that is augmented with a short proof π_i. The final verifier inspects the computation's outputs in order to decide whether they are "compliant" or not.

For example, C could simply require that each party's computation was carried out without errors. Or, C could require that not only each party's computation was carried out without errors, but also that the program run by each party carried a signature valid under the system administrator's public key; in such a case, the *local program* supplied by each party would be the combination of the program and the signature. Or, C could alternatively require that each party's computation involved a binary produced by

a compiler prescribed by the system administrator, which is known to perform certain tests on the code to be compiled (for example, type safety, static analysis, dynamic enforcement). Note that a party's local program could be a combination of code, human inputs, and randomness.

To formalize the above, we define and construct a PCD scheme: A cryptographic primitive that fully encapsulates the proof system machinery and provides a simple but very general "interface" to be used in applications.[a]

Our construction does require a minimal trusted setup: Every party should have black-box access to a simple *signed-input-and-randomness* functionality, which signs every input it receives along with some freshly-generated random bits. This is similar to standard functionality of cryptographic signing tokens and can also be implemented using Trusted Platform Module chips or a trusted party.

3.1. Our results

We introduce the generic approach of PCD for securing distributed computations and describing the cryptographic primitive of PCD schemes to capture this approach:

> **Theorem (informal).** PCD schemes can be constructed under standard cryptographic assumptions, given signed-input-and-randomness tokens.

3.2. The construction and its practicality

We do not rely on the traditional notion of a proof; instead, we rely on *computationally sound proofs.* These are proofs that always exist for true theorems and can be found efficiently given the appropriate witness. For false theorems, however, we only have the guarantee that no *efficient* procedure will be able to write a proof that makes us accept with more than negligible probability. Nonetheless, computationally sound proofs are just as good as traditional ones, for we are not interested in being protected against infeasible attack procedures, nor do we mind accepting a false theorem with, say, 2^{-100} probability.

The advantage of settling for computationally sound proofs is that they can be much shorter than the computation to which they attest and can be verified much more quickly than repeating the entire computation. To this end, we use probabilistically checkable proofs (PCPs) [11, 12], which originate in the field of computational complexity and its cryptographic extensions [9, 13, 14].

While our initial results establish theoretical foundations for PCD and show their possibility in principle, the aforementioned PCPs are computationally heavy and are notorious for being efficient only in the asymptotic sense, and they are not yet of practical relevance. Motivated by the potential impact of a practical PCD scheme, we have thus taken on the challenge of constructing a practical PCP system, in an ongoing collaboration with Professor Eli Ben-Sasson and a team of programmers at the Technion.

4. Related approaches

Cryptographic tools. Secure multiparty computation [15, 16, 17] considers the problem of secure function evaluation; our setting is not *one* function evaluation, but ensuring a single invariant (that is, *C*-compliance) through *many* interactions and computations between parties.

Platforms, languages, and static analysis. Integrity can be achieved by running on suitable fault-tolerant systems. Confidentiality can be achieved by platforms with suitable information flow control mechanisms following [18, 19] (for example, at the operating-system level [20, 21]). Various invariants can be achieved by statically analyzing programs and by programming language mechanisms such as type systems following [22, 23]. The inherent limitation of these approaches is that the output of such computation can be trusted only if one trusts the whole platform that executed it; this renders them ineffective in the setting of mutually untrusting distributed parties.

Run-time approaches. In proof-carrying code (PCC) [24], the code producer augments the code with formal, efficiently checkable proofs of the desired properties (typically, using the aforementioned language or static analysis techniques); PCC and PCD are

a. PCD schemes generalize the "computationally-sound proofs" of Micali [9], which consider only the "one-hop" case of a single prover and a single verifier and also generalize the "incrementally verifiable computation" of Valiant [10], which considers the case of an a-priori fixed sequence of computations.

complementary techniques, in the sense that PCD can enforce properties expressed via PCC. Dynamic analysis monitors the properties of a program's execution at run-time (for example, [25, 26, 27]). Our approach can be interpreted as extending dynamic analysis to the distributed setting, by allowing parties to (implicitly) monitor the program execution of all prior parties without actually being present during the executions. The Fabric system [28] is similar to PCD in motivation, but takes a very different approach: Fabric aims to make maximal use of distributed-system *given* trust constraints, while PCD *creates* new trust relations.

5. The road onward

We envision PCD as a framework for achieving security properties in a nonconventional way that circumvents many difficulties with current approaches. In PCD, faults and leakage are acknowledged as an expected occurrence, and rendered inconsequential by reasoning about properties of data that are independent of the preceding *computation.* The system designer prescribes the desired properties of the computation's output; proofs of these properties are attached to the data flowing through the system and are mutually verified by the system's components.

We have already shown explicit constructions of PCD, under standard cryptographic assumptions, in the model where parties have black-box access to a simple hardware token. The theoretical problem of weakening this requirement, or formally proving that it is (in some sense) necessary, remains open. In recent work, we show how to resolve this problem in the case of a single party's computation [29].

As for practical realizations, since there is evidence that the use of PCPs for achieving short proofs is inherent [30], we are tackling head-on the challenge of making PCPs practical. We are also studying devising ways to express the security properties, to be enforced by PCD, using practical programming languages such as C++.

In light of these, as real-world practicality of PCD becomes closer and closer, the task of *compliance engineering* becomes an exciting direction. While PCD provides a protocol compiler to ensure any compliance predicate in a distributed computation, figuring out what are useful compliance predicates in this or that setting is a problem in its own right.

We already envision problem domains where we believe enforcing compliance predicates will come a long way toward securing distributed systems in a strong sense:

- **Multilevel security.** PCD may be used for information flow control. For example, consider enforcing multilevel security [31, Chap. 8.6] in a room full of data-processing machines. We want to publish outputs labeled "nonsecret," but are concerned that they may have been tainted by "secret" information (for example, due to bugs, via software side channel attacks [32] or, perhaps, via literal eavesdropping [33, 34, 35]). PCD then allows you to reduce the problem of controlling information flow to the problem of controlling the perimeter of the information room by ensuring that every network packet leaving the room is inspected by the PCD verifier to establish it carries a valid proof.
- **IT supply chain and hardware Trojans.** Using PCD, one can achieve fault isolation and accountability at the level of system components (for example, chips or software modules) by having each component augment every output with a proof that its computation, *including all history it relied on,* was correct. Any fault in the computation, malicious or otherwise, will then be identified by the first nonfaulty subsequent component. Note that even the PCD verifiers themselves do not have to be trusted except for the very last one.
- **Distributed type safety.** Language-based type-safety mechanisms have tremendous expressive power, but are targeted at the case where the underlying execution platform can be trusted to enforce type rules. Thus, they typically cannot be applied across distributed systems consisting of multiple mutually untrusting execution platforms. This barrier can be surmounted by using PCD to augment typed values passing between systems with proofs for the correctness of the type.

Efforts to understand how to think about compliance in concrete problem domains are likely to uncover common problems and corresponding design patterns [36], thus improving our overall ability to correctly phrase desired security properties as compliance predicates.

We thus pose the following challenge: Given a genie that grants every wish expressed as a compliance predicate on distributed computations, what compliance predicates would you wish for in order to achieve the security properties your system needs? 

Acknowledgments

This research was partially supported by the Check Point Institute for Information Security, the Israeli Centers of Research Excellence program (center No. 4/11), the European Community's Seventh Framework Programme grant 240258, the National Science Foundation (NSF) grant NSF-CNS-0808907, and the Air Force Research Laboratory (AFRL) grant FA8750-08-1-0088. Views and conclusions contained here are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either express or implied, of AFRL, NSF, the US government or any of its agencies.

About the authors

Alessandro Chiesa is a second-year doctoral student in the Theory of Computation group in the Computer Science and Artificial Intelligence Laboratory (CSAIL) at Massachusetts Institute of Technology (MIT). He is interested in cryptography, complexity theory, quantum computation, mechanism design, algorithms, and security. He can be reached at MIT CSAIL, alexch@csail.mit.edu.

Eran Tromer is a faculty member at the School of Computer Science at Tel Aviv University. His research focus is information security, cryptography, and algorithms. He is particularly interested in what happens when cryptographic systems meet the real world, where computation is faulty and leaky. He can be reached at Tel Aviv University, tromer@cs.tau.ac.il.

References

[1] Ferdowsi A. S3 data corruption? *Amazon Web Services* (discussion forum). 2008 Jun 22. Available at: https://forums.aws.amazon.com/thread.jspa?threadID=22709&start=0&tstart=0

[2] Ristenpart T, Tromer E, Shacham H, Savage S. Hey, you, get off of my cloud! Exploring information leakage in third-party compute clouds. In: *Proceedings of the 16th ACM Conference on Computer and Communications Security;* Nov 2009; Chicago, IL. p. 199–212. Available at: http://cseweb.ucsd.edu/~hovav/dist/cloudsec.pdf

[3] Biham E, Shamir A. Differential fault analysis of secret key cryptosystems. In: Kaliski BS Jr., editor. *Advances in Cryptology—CRYPTO '97* (Proceedings of the 17th Annual International Cryptology Conference; Aug 1997; Santa Barbara, CA). LNCS, 1294. London (UK): Springer-Verlag; 1997. p. 513–525. DOI: 10.1007/BFb0052259

[4] Collins DR. Trust, a proposed plan for trusted integrated circuits. Paper presented at a conference; Mar 2006; p. 276–277. Available at: http://oai.dtic.mil/oai/oai?verb=getRecord&metadataPrefix=html&identifier=ADA456459

[5] Agrawal D, Baktir S, Karakoyunlu D, Rohatgi P, Sunar B. Trojan detection using IC fingerprinting. In: *Proceedings of the 2007 IEEE Symposium on Security and Privacy;* May 2007; Oakland, CA. p. 296–310. DOI: 10.1109/SP.2007.36

[6] Biham E, Carmeli Y, Shamir A. Bug attacks. In: Wagner D, editor. *Advances in Cryptology—CRYPTO 2008* (Proceedings of the 28th Annual International Cryptology Conference; Aug 2008; Santa Barbara, CA). LNCS, 5157. Berlin (Germany): Springer-Verlag; 2008. p. 221–240. DOI: 10.1007/978-3-540-85174-5_13

[7] King ST, Tucek J, Cozzie A, Grier C, Jiang W, Zhou Y. Designing and implementing malicious hardware. In: *Proceedings of the First USENIX Workshop on Large-Scale Exploits and Emergent Threats;* Apr 2008; San Francisco, CA. p. 1–8. Available at: http://www.usenix.org/events/leet08/tech/full_papers/king/king.pdf

[8] Roy JA, Koushanfar F, Markov IL. Circuit CAD tools as a security threat. In: *Proceedings of the First IEEE International Workshop on Hardware-Oriented Security and Trust;* Jun 2008; Anaheim, CA. p. 65–66. DOI: 10.1109/HST.2008.4559052

[9] Micali S. Computationally sound proofs. *SIAM Journal on Computing.* 2000;30(4):1253–1298. DOI: 10.1137/S0097539795284959

[10] Valiant P. Incrementally verifiable computation or

proofs of knowledge imply time/space efficiency. In: Canetti R, editor. *Theory of Cryptography* (Proceedings of the Fifth Theory of Cryptography Conference; Mar 2008; New York, NY). LNCS, 4948. Berlin (Germany): Springer-Verlag; 2008. p. 1–18. DOI: 10.1007/978-3-540-78524-8_1

[11] Babai L, Fortnow L, Levin LA, Szegedy M. Checking computations in polylogarithmic time. In: *Proceedings of the 23rd Annual ACM Symposium on Theory of Computing;* May 1991; New Orleans, LA. p. 21–32. DOI: 10.1145/103418.103428

[12] Ben-Sasson E, Sudan M. Simple PCPs with poly-log rate and query complexity. In: *Proceedings of the 37th Annual ACM Symposium on Theory of Computing;* May 2005; Baltimore, MD. p. 266–275. DOI: 10.1145/1060590.1060631

[13] Kilian J. A note on efficient zero-knowledge proofs and arguments. In: *Proceedings of the 24th Annual ACM Symposium on Theory of Computing;* May 1992; Victoria, BC, Canada. p. 723–732. DOI: 10.1145/129712.129782

[14] Barak B, Goldreich O. Universal arguments and their applications. In: *Proceedings of the 17th IEEE Annual Conference on Computational Complexity;* May 2002; Montreal, Quebec , Canada. p. 194–203. DOI: 10.1109/CCC.2002.1004355

[15] Goldreich O, Micali S, Wigderson A. How to play ANY mental game. In: *Proceedings of the 19th Annual ACM Symposium on Theory of Computing;* May 1987; New York, NY. p. 218–229. DOI: 10.1145/28395.28420

[16] Ben-Or M, Goldwasser S, Wigderson A. Completeness theorems for non-cryptographic fault-tolerant distributed computation. In: *Proceedings of the 20th Annual ACM Symposium on Theory of Computing;* May 1988; Chicago, IL. p. 1–10. DOI: 10.1145/62212.62213

[17] Chaum D, Crépeau C, Damgård I. Multiparty unconditionally secure protocols. In: *Proceedings of the 20th Annual ACM Symposium on Theory of Computing;* May 1988; Chicago, IL. p. 11–19. DOI: 10.1145/62212.62214

[18] Denning DE, Denning PJ. Certification of programs for secure information flow. *Communications of the ACM.* 1977;20(7):504–513. DOI: 10.1145/359636.359712

[19] Myers AC, Liskov B. A decentralized model for information flow control. In: *Proceedings of the 16th ACM SIGOPS Symposium on Operating Systems Principles;* Oct 1997; Saint-Malo, France. p. 129–142. DOI: 10.1145/268998.266669

[20] Krohn M, Yip A, Brodsky M, Cliffer N, Kaashoek MF, Kohler E, Morris R. Information flow control for standard OS abstractions. In: *Proceedings of the 21st ACM SIGOPS Symposium on Operating Systems Principles;* Oct 2007; Stevenson, WA. p. 321–334. DOI: 10.1145/1294261.1294293

[21] Zeldovich N, Boyd-Wickizer S, Kohler E, Mazières D. Making information flow explicit in HiStar. In: *Proceedings of the Seventh USENIX Symposium on Operating Systems Design and Implementation;* Nov 2006; Seattle, WA. p. 19–19. Available at: http://www.usenix.org/event/osdi06/tech/full_papers/zeldovich/zeldovich.pdf

[22] Andrews GR, Reitman RP. An axiomatic approach to information flow in programs. *ACM Transactions on Programming Languages and Systems.* 1980;2(1):56–76. DOI: 10.1145/357084.357088

[23] Denning DE. A lattice model of secure information flow. *Communications of the ACM.* 1976;19(5):236–243. DOI: 10.1145/360051.360056

[24] Necula GC. Proof-carrying code. In: *Proceedings of the 24th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages;* Jan 1997; Paris, France. p. 106–119. DOI: 10.1145/263699.263712

[25] Nethercote N, Seward J. Valgrind: A framework for heavyweight dynamic binary instrumentation. In: *Proceedings of the 2007 ACM SIGPLAN Conference on Programming Language Design and Implementation;* Jun 2007; San Diego, CA. p. 89–100. DOI: 10.1145/1250734.1250746

[26] Suh GE, Lee JW, Zhang D, Devadas S. Secure program execution via dynamic information flow tracking. In: *Proceedings of the 11th International Conference on Architectural Support for Programming Languages and Operating Systems;* Oct 2004; Boston, MA. p. 85–96. DOI: 10.1145/1024393.1024404

[27] Kiriansky V, Bruening D, Amarasinghe SP. Secure execution via program shepherding. In: *Proceedings of the 11th USENIX Security Symposium;* Aug 2002; San Francisco, CA. p. 191–206. Available at: http://www.usenix.org/publications/library/proceedings/sec02/full_papers/kiriansky/kiriansky_html/index.html

[28] Liu J, George MD, Vikram K, Qi X, Waye L, Myers AC. Fabric: A platform for secure distributed computation and storage. In: *Proceedings of the 22nd ACM SIGOPS Symposium on Operating Systems Principles;* Oct 2009; Big Sky, MT. p. 321–334. DOI: 10.1145/1629575.1629606

[29] Bitansky N, Canetti R, Chiesa A, Tromer E. From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again. Cryptology ePrint Archive. 2011;Report 2011/443. Available at: http://eprint.iacr.org/2011/443

[30] Rothblum GN, Vadhan S. Are PCPs inherent in efficient arguments? In: *Proceedings of the 24th IEEE Annual Conference on Computational Complexity;* Jul 2009; Paris, France. p. 81–92. DOI: 10.1109/CCC.2009.40

[31] Anderson RJ. *Security Engineering: A Guide to Building Dependable Distributed Systems.* 2nd ed. Indianapolis (IN): Wiley Publishing; 2008. ISBN: 978-0-470-06852-6

[32] Brumley D, Boneh D. Remote timing attacks are practical. *Computer Networks: The International Journal of Computer and Telecommunications Networking.* 2005;48(5):701–716.

[33] LeMay M, Tan J. Acoustic surveillance of physically unmodified PCs. In: *Proceedings of the 2006 International Conference on Security and Management;* Jun 2006; Las Vegas, NV. p. 328–334. Available at: http://ww1.ucmss.com/books/LFS/CSREA2006/SAM4311.pdf

[34] Asonov D, Agrawal R. Keyboard acoustic emanations. In: *Proceedings of the 2004 IEEE Symposium on Security and Privacy;* May 2004; Oakland, CA. p. 3–11. DOI: 10.1109/SECPRI.2004.1301311

[35] Tromer E, Shamir A. Acoustic cryptanalysis: On nosy people and noisy machines. Presentation at: *Eurocrypt 2004 Rump Session;* May 2004; Interlaken, Switzerland. Available at: http://people.csail.mit.edu/tromer/acoustic

[36] Gamma E, Helm R, Johnson R, Vlissides J. *Design Patterns: Elements of Reusable Object-Oriented Software.* Boston (MA): Addison-Wesley Longman Publishing Co., Inc.; 1995. ISBN: 9780201633610