CANDIDACY EXAM

CAPITALIZING ON SECURITY, PERFORMANCE, AND ENERGY TRADEOFFS IN FULL DRIVE ENCRYPTION SCHEMES FOR FUN AND PROFIT

BY

BERNARD DICKENS III

MARCH 2019

TABLE OF CONTENTS

LIST OF TABLES

ABSTRACT

Security is an important concern among several in modern computer systems. These concerns often exist in contention over a set of finite resources, e.g. DRAM bandwidth/capacity, CPU time, drive space. These create a space of possible tradeoffs that we can optimize against. However, exploiting the slack these spaces expose in the context of full-drive encryption (FDE) and file systems is not straightforward.

Full-drive encryption is especially important for mobile devices because they contain large quantities of sensitive data yet are easily lost or stolen. As this research demonstrates, the standard approach to FDE—the AES block cipher in XTS mode—is 3-5x slower than unencrypted storage. Authenticated encryption based on stream ciphers is already used as a faster alternative to AES in other contexts, such as HTTPS, but the conventional wisdom is that stream ciphers are unsuitable for FDE. Used naively in drive encryption, stream ciphers are vulnerable to attacks, and mitigating these attacks with on-drive metadata is generally believed to ruin performance.

With this research, we first demonstrate that recent developments in mobile hardware invalidate this assumption, making it possible to use fast stream ciphers for FDE. Next, we evaluate a new StrongBox implementation capable of using ciphers beyond ChaCha20 and AES-CTR to capitalize on security, performance, and energy tradeoffs. Finally, we will present a theoretical framework formally analyzing the security of the StrongBox FDE construction.

INTRODUCTION

In this section we make our thesis statement, describe the problem StrongBox solves, and outline the general StrongBox approach.

1.1 Thesis Statement

With this research into filesystem, device driver, and hardware Flash Translation Layer (FTL) based Full Disk Encryption (FDE) schemes, we primarily consider: (1) the established wisdom in the crypto community that stream ciphers are unsuitable for FDE and (2) exploring the tradeoff space made between total energy use, filesystem performance, and reasonable security guarantees when comparing specific cipher configurations. In the first case, we develop and implement a secure approach to FDE based on stream ciphers, the proliferation of secure hardware, and the characteristics of Log-Structured Filesystems (LFS). In the second case, we implement a dozen stream ciphers—each exposing several knobs—and demonstrate navigating our tradeoff space via runtime cipher switching while maintaining reasonable security guarantees. We then present a formal analysis of our system's security guarantees.

1.2 Problem Description

Full-drive encryption (FDE)[1] is an essential technique for protecting the privacy of data at rest. For mobile devices, maintaining data privacy is especially important as these devices contain sensitive personal and financial data yet are easily lost or stolen. The current standard for securing data at rest is to use the AES cipher in XTS mode [20, 26]. Unfortunately, employing AES-XTS increases read/write latency by 3–5× compared to unencrypted storage.

[1] The common term is full-*disk* encryption, but this work targets SSDs, so we use *drive*.

It is well known that authenticated encryption using *stream* ciphers—such as ChaCha20 [2]—is faster than using AES. Indeed, Google made the case for stream ciphers over AES, switching HTTPS connections on Chrome for Android to use a stream cipher for better performance [28]. Stream ciphers are not used for FDE, however, for two reasons: (1) confidentiality and (2) performance. First, when applied naively to stored data, stream ciphers are trivially vulnerable to attacks—including *many-time pad and rollback attacks*—that reveal the plaintext by overwriting a secure storage location with the same key. Second, it has been assumed that adding the meta-data required to resist these attacks would ruin the stream cipher's performance advantage. Thus, the conventional wisdom is that FDE necessarily incurs the overhead of AES-XTS or a similar primitive.

We argue that two technological shifts in mobile device hardware overturn this conventional wisdom, enabling confidential, high-performance storage with stream ciphers. First, these devices commonly employ solid-state storage with Flash Translation Layers (FTL), which operate similarly to Log-structured File Systems (LFS) [13, 14, 23]. Second, mobile devices now support trusted hardware, such as Trusted Execution Environments (TEE) [16, 25] and secure storage areas [5]. FTLs and LFSes are used to limit sector/cell overwrites, hence extending the life of the drive. Most writes simply appended to a log, reducing the occurrence of overwrites and the chance for attacks. The presence of secure hardware means that drive encryption modules have access to persistent, monotonically increasing counters that can be used to prevent rollback attacks when overwrites do occur.

Given these trends, we propose StrongBox, a new method for securing data at rest. StrongBox is a drop-in replacement for AES-XTS-backed FDE such as dm-crypt [17]; i.e. it requires no interface changes. The primary challenge is that even with a FTL or LFS running above an SSD, filesystem blocks will occasionally be overwritten; e.g. by segment cleaning or *garbage collection*. StrongBox overcomes this challenge by using a fast stream cipher for confidentiality and performance with integrity preserving Message Authentication

Codes [18] or “MAC tags” and a secure, persistent hardware counter to ensure integrity and prevent attacks. *StrongBox’s main contribution is a system design enabling the first confidential, high-performance drive encryption based on a stream cipher.*

BACKGROUND AND RELATED WORK

In this section we describe the three primary components of the StrongBox research project, and define a timeline of completion.

Some of the most popular cryptosystems offering a confidentiality guarantee for data at rest employ a symmetric encryption scheme known as a Tweakable Enciphering Scheme (TES) [3, 22]. There have been numerous TES-based constructions securing data at rest [3, 8, 30], including the well known XEX-based XTS operating mode of AES [26] explored earlier in this work. Almost all TES constructions and the storage management systems that implement them use one or more block ciphers as their primary primitive [3, 24].

Our StrongBox implementation borrows from the design of these systems. One in particular is *dm-crypt*, a Linux framework employing a *LinuxDeviceMapper* to provide a virtual block interface for physical block devices. Dm-crypt provides an implementation of the AES-XTS algorithm among others and is used widely in the Linux ecosystem [1, 17]. The algorithms provided by dm-crypt all employ block ciphers [17]. Instead of a block cipher, however, StrongBox uses a stream cipher to provide the same confidentiality guarantee and consistent or better I/O performance. Further unlike dm-crypt and other similar virtualization frameworks, StrongBox's ciphering operations do not require sector level tweaks, depending on the implementation. With StrongBox, several physical blocks consisting of one or more sectors are considered as discrete logical units, i.e. nuggets and flakes.

Substituting a block cipher for a stream cipher forms the core of several contributions to the state-of-the-art [3, 24]. Chakraborty et al. proposed STES—a stream cipher based low cost scheme for securing stored data [3]. STES is a novel TES which can be implemented compactly with low overall power consumption. It combines a stream cipher and a universal hash function via XOR and is targeting low cost FPGAs to provide confidentiality of data on USBs and SD cards. Our StrongBox, on the other hand, is not a TES and does not directly implement a TES. StrongBox combines a stream cipher with nonce "tweak" and nugget data via XOR and is targeting any configuration employing a well-behaved Log-structured Filesystem (LFS) at some level to provide confidentiality of data.

Offering a transparent cryptographic layer at the block device level has been proposed numerous times [9]. Production implementations include storage management systems like dm-crypt. Specifically, Hein et al. proposed the Secure Block Device (SBD) [9]—an ARM TrustZone secure world transparent filesystem encryption layer optimized for ANDIX OS and implemented and evaluated using the Linux Network Block Device (NBD) driver. StrongBox is also implemented and evaluated using the NBD, but is not limited to one specific operating system. Further unlike StrongBox, SBD is not explicitly designed for use outside of the ARM TrustZone secure world. Contrarily, StrongBox was designed to be used on any system that provides a subset of functionality provided by a Trusted Platform Module (TPM) and/or Trusted Execution Environment (TEE). Specifically, StrongBox requires the availability of a dedicated hardware protected secure monotonic counter to prevent rollback attacks and ensure the freshness of StrongBox. The primary design goal of StrongBox is to achieve provide higher performance than the industry standard AES-XTS algorithm utilizing a stream cipher.

StrongBox's design is only possible because of the availability of hardware support for security, which has been a major thrust of research efforts [6, 10, 15, 27, 29, 31], and is now available in almost all commercial mobile processors [7, 12, 21, 25]. Our implementation makes use of the replay protected memory block on eMMC devices [5, 21], but it could be reimplemented using any hardware that supports persistent, monotonic counters.

The combination of trusted hardware and monotonic counters enables new security mechanisms. For example, van Dijk et al. use this combination allow clients to securly store data on an untrusted server [4]. Like StrongBox, their approach relies on trusted hardware (TPM specifically [7]), logs, and monotonic counters. The van Dijk et al. approach, however, uses existing secure storage and is not concerned with storage speed. StrongBox uses these same mechanisms along with novel metadata layout and system design to solve a different problem: providing higher performance than AES-XTS based approaches.

Achieving on-drive data integrity protection through the use of checksums has been used by filesystems and many other storage management systems. Examples include ZFS [19] and others [9]. For our implementation of StrongBox, we used the Merkle Tree library offered by SBD to manage our in-memory checksum verification. A proper implementation of StrongBox need not use the SDB SHA-256 Merkle Tree library. It was chosen for convenience.

Khati et al. visited the FDE problem from both a theoretical and practical standpoint with their work [11], attempting to bridge the gap between the theory of cryptography and applied cryptography, as well the formalization of many FDE notions.

STRONGBOX I, II, III

In this section we describe the three primary components of the StrongBox research project, and define a timeline of completion.

3.1 Project Timeline

These are dates we expect certain research milestones to be accomplished:

1. **4/2019** Complete use case implementation and case studies.

2. **5/2019** Literature review on FDE theory and formal organization.

3. **5-6/2019** StrongBox II cipher switching draft paper is completed.

4. **6/2019** Formal theoretical framework to model StrongBox security guarantees.

5. **12/2019** Analysis of security model and guarantees versus AES-XTS and other schemes.

6. **2/2020** StrongBox III (theoretical security framework) draft paper is completed.

3.2 StrongBox I: Confidentiality, Integrity, and Performance using Stream Ciphers

StrongBox I is the original published work demonstrating how recent developments in hardware coupled with core insights about filesystem behavior invalidate the assumption that practically mitigating attacks against high performance stream ciphers for FDE is infeasible.

With this research, it was shown that recent developments in mobile hardware invalidate the assumption that stream ciphers are unsuitable for FDE, making it possible to take advantage of fast stream ciphers. Modern mobile devices employ solid-state storage with Flash Translation Layers (FTL), which operate similarly to Log-structured File Systems

(LFS). They also include trusted hardware such as Trusted Execution Environments (TEEs) and secure storage areas. StrongBox I, implemented with the ChaCha20 stream cipher, leveraged these two trends to outperform dm-crypt, the de-facto Linux FDE endpoint.

3.2.1 Completed Work

- **Published.** *StrongBox: Confidentiality, Integrity, and Performance using Stream Ciphers for Full-Drive Encryption* was published in the Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'18).

- **Implemented.** A prototype of the StrongBox approach was implemented in C. The full 5000+ LoC source for StrongBox I is available open source (see Appendix A).

- **Patented.** The StrongBox approach has been patented by the University of Chicago, pending prosecution and non-provisional filing.

- **Inspired.** Using stream ciphers for Full-Drive Encryption motivated research in a similar vein: using stream ciphers with highly available systems to secure arbitrary resources on the internet (HASCHK).

3.3 StrongBox II: Energy, Security, and Performance Tradeoffs with Cipher Switching

With StrongBox II, we evaluate a new StrongBox implementation capable of using ciphers beyond ChaCha20 and AES-CTR; specifically, the following profile 1 stream ciphers were added to StrongBox: Sosemanuk, HC-128, Rabbit, Salsa20, Salsa12, and Salsa8, ChaCha12, ChaCha8, Freestyle, and ARM Neon SIMD accelerated versions of ChaCha.

The eSTREAM portfolio ciphers fall into two profiles. Profile 1 contains stream ciphers more suitable for software applications with high throughput requirements. Profile 2 stream ciphers are particularly suitable for hardware applications with restricted resources such as limited storage, gate count, or power consumption. This research does not consider profile 2 stream ciphers offered by the eSTREAM portfolio.

3.3.1 Completed Work

- **Cipher abstraction and eSTREAM implementations.** With StrongBox II, we abstracted the cipher interface into an independent subsystem such that StrongBox II functions with any stream or block cipher, including the original ChaCha20. We also included an implementation of the Freestyle randomized output stream cipher for consideration with FDE, similarly implemented in C. The full source code for StrongBox II is available alongside the original (see Appendix A).

- **Cipher switching framework and tradeoff exploration.** We performed experiments to quantify the performance and security properties of the various ciphers we implemented. From that, we engineered a "cipher switching" abstraction allowing StrongBox to switch (offline) between any cipher in its library. Further, by studying the contention between StrongBox's observed *energy* use, the *security* of the StrongBox construction given a certain cipher configuration, and the *performance* of the construction under a given workload, we modeled a navigable tradeoff space where manipulating magnitude in one dimension yields a non-linear change in another.

- **Cipher switching strategies.** We implemented five "switching strategies" based on our experimental observations. These strategies allow StrongBox to navigate the aforementioned tradeoff space both offline *and online.*

3.3.2 Current Work

- **4-5/2019**
  - Complete experimental evaluation of (online) cipher switching strategies.
  - Finish use case framework implementation and determine worthy use case implementations based on the aforementioned evaluation.
  - Complete study and experimental evaluation of worthy use cases.
- **5-6/2019**
  - Collate research into paper: *StrongBox II: A Study of Practical Tradeoffs Between Energy, Performance, and Security.*

3.3.3 Considerations

At this point in the project, we have established a useful set of pareto curves, the cipher switching strategy implementations are practical, and our use cases are built on top of those switching strategies. Our biggest concern is accurate and meaningful energy readings. Integrating the energy use telemetry (i.e. Energymon) directly into the StrongBox core instead of at the test fixture level along with other tweaks and checks should address this. However, even without a robust set of metrics on energy use, the performance-security curve alone motivates the switching strategies and, in turn, the use cases.

3.4 StrongBox III: Theoretical Framework for Analyzing StrongBox FDE

With the final component of this research, at minimum we want: (1) define a set of formal FDE security notions in context, (2) define the cryptographic goals and guarantees of

StrongBox, (3) introduce a theoretical framework and models to evaluate StrongBox, (4) potentially explore AES-XTS security guarantees versus StrongBox guarantees.

3.4.1 Current Work

- **5-6/2019**
  - Complete review of literature on relevant theoretical frameworks that examine security within the constraints of FDE, e.g. efficient constructions that achieve security in context and subject to practical constraints (Rogaway, Fruhwirth, et al).
  - Outline well-defined cryptographic goals StrongBox should achieve; outline guarantees any FDE scheme is expected to provide.
- **6-12/2019**
  - Introduce theoretical framework to formalize important security notions and evaluate StrongBox FDE, perhaps under IND-CPA, IND-CCA, "time-to-bruteforce" where relevant, or related definitions.
  - Explore AES-XTS security guarantees versus StrongBox ciphers in an appropriate mode, perhaps including key management concerns (e.g. circular encryption) and StrongBox's inherent cryptographic agility.
- **1-2/2020**
  - Collate research into paper: *A Formal Analysis of StrongBox FDE.*

CODE AVAILABILITY

This appendix provides links for the StrongBox I and StrongBox II source code. The two libraries/APIs that our StrongBox implementations rely on are additionally provided. Note that resource locations, URLs, frameworks, interfaces, etc. will likely change overtime, while this text remains static.

You can find instructions on how to build, test, and modify the StrongBox source in the provided README documentation linked below.

Table A.1: Provides URLs for the products yielded and/or used by this research.

Project Source	**Language**	**URL**
StrongBox I Source	C	https://github.com/research/buselfs-public
StrongBox II Source	C	https://github.com/research/buselfs2-public
SBD Merkle Tree Implementation	C	https://github.com/IAIK/secure-block-device
Block Device in User Space (BUSE)	C/Shell	https://github.com/acozzette/BUSE

BIBLIOGRAPHY

[1] *Android Open Source Project: Full-Disk Encryption.* URL: `https://source.android.com/security/encryption/full-disk` (visited on 04/26/2017).

[2] D. J. Bernstein. *ChaCha, a variant of Salsa20.* Tech. rep. University of Illinois at Chicago, 2008.

[3] D. Chakraborty, C. Mancillas-López, and P. Sarkar. "STES: A Stream Cipher Based Low Cost Scheme for Securing Stored Data". In: *IEEE Transactions on Computers* 64.9 (2015), pp. 2691–2707. ISSN: 0018-9340. DOI: `10.1109/TC.2014.2366739`.

[4] M. van Dijk, J. Rhodes, L. F. G. Sarmenta, and S. Devadas. "Offline Untrusted Storage with Immediate Detection of Forking and Replay Attacks". In: *Proceedings of the 2007 ACM Workshop on Scalable Trusted Computing.* STC '07. New York, NY, USA: ACM, 2007, pp. 41–48. ISBN: 978-1-59593-888-6. DOI: `10.1145/1314354.1314364`. URL: `http://doi.acm.org/10.1145/1314354.1314364`.

[5] *EMBEDDED MULTI-MEDIA CARD (e•MMC), ELECTRICAL STANDARD (5.1).* 2015. URL: `https://www.jedec.org/standards-documents/results/jesd84-b51` (visited on 04/26/2017).

[6] A. Ferraiuolo, R. Xu, D. Zhang, A. C. Myers, and G. E. Suh. "Verification of a Practical Hardware Security Architecture Through Static Information Flow Analysis". In: *Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS 2017, Xi'an, China, April 8-12, 2017.* 2017, pp. 555–568. DOI: `10.1145/3037697.3037739`. URL: `http://doi.acm.org/10.1145/3037697.3037739`.

[7] T. C. Group. *TCG: Trusted platform module summary.* 2008.

[8] S. Halevi and P. Rogaway. “A Tweakable Enciphering Mode”. In: *Advances in Cryptology - CRYPTO 2003: 23rd Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 2003. Proceedings.* Ed. by D. Boneh. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 482–499. ISBN: 978-3-540-45146-4. DOI: 10.1007/978-3-540-45146-4_28. URL: http://dx.doi.org/10.1007/978-3-540-45146-4_28.

[9] D. Hein, J. Winter, and A. Fitzek. “Secure Block Device – Secure, Flexible, and Efficient Data Storage for ARM TrustZone Systems”. In: *2015 IEEE Trustcom/BigDataSE/ISPA*. Vol. 1. 2015, pp. 222–229. DOI: 10.1109/Trustcom.2015.378.

[10] M. Hicks, C. Sturton, S. T. King, and J. M. Smith. “SPECS: A Lightweight Runtime Mechanism for Protecting Software from Security-Critical Processor Bugs”. In: *Proceedings of the Twentieth International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS ’15, Istanbul, Turkey, March 14-18, 2015.* 2015, pp. 517–529. DOI: 10.1145/2694344.2694366. URL: http://doi.acm.org/10.1145/2694344.2694366.

[11] L. Khati. “Full Disk Encryption: Bridging Theory and Practice”. In: *Topics in Cryptology – CT-RSA 2017.* Cham: Springer International Publishing, 2017, pp. 241–257.

[12] D. Kirovski, M. Drinić, and M. Potkonjak. “Enabling Trusted Software Integrity”. In: *Proceedings of the 10th International Conference on Architectural Support for Programming Languages and Operating Systems.* ASPLOS X. New York, NY, USA: ACM, 2002, pp. 108–120. ISBN: 1-58113-574-2. DOI: 10.1145/605397.605409. URL: http://doi.acm.org/10.1145/605397.605409.

[13] R. Konishi, Y. Amagai, K. Sato, H. Hifumi, S. Kihara, and S. Moriai. “The Linux Implementation of a Log-structured File System”. In: *SIGOPS Oper. Syst. Rev.* 40.3 (July 2006), pp. 102–107. ISSN: 0163-5980. DOI: 10.1145/1151374.1151375. URL: http://doi.acm.org/10.1145/1151374.1151375.

[14] C. Lee, D. Sim, J. Hwang, and S. Cho. "F2FS: A New File System for Flash Storage". In: *13th USENIX Conference on File and Storage Technologies (FAST 15)*. Santa Clara, CA: USENIX Association, 2015, pp. 273–286. ISBN: 978-1-931971-201. URL: https://www.usenix.org/conference/fast15/technical-sessions/presentation/lee.

[15] X. Li, V. Kashyap, J. K. Oberg, M. Tiwari, V. R. Rajarathinam, R. Kastner, T. Sherwood, B. Hardekopf, and F. T. Chong. "Sapper: a language for hardware-level security policy enforcement". In: *Architectural Support for Programming Languages and Operating Systems, ASPLOS '14, Salt Lake City, UT, USA, March 1-5, 2014*. 2014, pp. 97–112. DOI: 10.1145/2541940.2541947. URL: http://doi.acm.org/10.1145/2541940.2541947.

[16] A. Limited. *ARM security technology: Building a secure system using TrustZone technology.* PRD29-GENC-009492C. 2009.

[17] *Linux kernel device-mapper crypto target.* 2013. URL: https://gitlab.com/cryptsetup/cryptsetup (visited on 04/26/2017).

[18] *Message Authentication Code Standard ISO/IEC 9797-1:2011.* 2011. URL: https://www.iso.org/standard/50375.html (visited on 04/26/2017).

[19] *Oracle blog: ZFS End-to-End Data Integrity.* 2005. URL: https://blogs.oracle.com/bonwick/zfs-end-to-end-data-integrity (visited on 04/26/2017).

[20] *Recommendation for Block Cipher Modes of Operation: The XTS-AES Mode for Confidentiality on Storage Devices.* NIST Special Publication 800-38E. 2010. URL: http://nvlpubs.nist.gov/.

[21] A. K. Reddy, P. Paramasivam, and P. B. Vemula. "Mobile secure data protection using eMMC RPMB partition". In: *2015 International Conference on Computing and*

Network Communications (CoCoNet). 2015, pp. 946–950. DOI: 10.1109/CoCoNet.2015.7411305.

[22] P. Rogaway. *Efficient Instantiations of Tweakable Blockciphers and Refinements to Modes OCB and PMAC*. Tech. rep. University of California at Davis, 2004.

[23] M. Rosenblum and J. K. Ousterhout. "The Design and Implementation of a Log-structured File System". In: *ACM Trans. Comput. Syst.* 10.1 (Feb. 1992), pp. 26–52. ISSN: 0734-2071. DOI: 10.1145/146941.146943. URL: http://doi.acm.org/10.1145/146941.146943.

[24] P. Sarkar. *Tweakable Enciphering Schemes From Stream Ciphers With IV*. Tech. rep. Indian Statistical Institute, 2009.

[25] G. P. D. Technology. *TEE client API specification version 1.0*. GPD_SPE_007. 2010.

[26] *The XTS-AES Tweakable Block Cipher*. IEEE Std 1619-2007. 2008.

[27] M. Tiwari, J. Oberg, X. Li, J. Valamehr, T. E. Levin, B. Hardekopf, R. Kastner, F. T. Chong, and T. Sherwood. "Crafting a usable microkernel, processor, and I/O system with strict and provable information flow security". In: *38th International Symposium on Computer Architecture (ISCA 2011), June 4-8, 2011, San Jose, CA, USA*. 2011, pp. 189–200. DOI: 10.1145/2000064.2000087. URL: http://doi.acm.org/10.1145/2000064.2000087.

[28] *TLS Symmetric Crypto*. 2014. URL: https://www.imperialviolet.org/2014/02/27/tlssymmetriccrypto.html (visited on 04/26/2017).

[29] J. Valamehr, M. Chase, S. Kamara, A. Putnam, D. Shumow, V. Vaikuntanathan, and T. Sherwood. "Inspection resistant memory: Architectural support for security from physical examination". In: *39th International Symposium on Computer Architecture (ISCA 2012), June 9-13, 2012, Portland, OR, USA*. 2012, pp. 130–141. DOI: 10.1109/ISCA.2012.6237012. URL: https://doi.org/10.1109/ISCA.2012.6237012.

[30] P. Wang, D. Feng, and W. Wu. "HCTR: A Variable-Input-Length Enciphering Mode". In: *Information Security and Cryptology: First SKLOIS Conference, CISC 2005, Beijing, China, December 15-17, 2005. Proceedings.* Ed. by D. Feng, D. Lin, and M. Yung. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 175–188. ISBN: 978-3-540-32424-9. DOI: 10.1007/11599548_15. URL: http://dx.doi.org/10.1007/11599548_15.

[31] R. Zhang, N. Stanley, C. Griggs, A. Chi, and C. Sturton. "Identifying Security Critical Properties for the Dynamic Verification of a Processor". In: *Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS 2017, Xi'an, China, April 8-12, 2017.* 2017, pp. 541–554. DOI: 10.1145/3037697.3037734. URL: http://doi.acm.org/10.1145/3037697.3037734.