

A Formally Verified Secure Caching Mechanism on TrustZone-enabled Microcontrollers

Salvatore Bramante

IMT School for Advanced Studies Lucca
Lucca, Italy
salvatore.bramante@imtlucca.it

Matteo Busi

Ca' Foscari University of Venice
Venezia, Italy
matteo.busi@unive.it

Alessandro Cilardo

University of Naples Federico II
Napoli, Italy
acilardo@unina.it

Riccardo Focardi

Ca' Foscari University of Venice
Venezia, Italy
focardi@unive.it

Flaminia Luccio

Ca' Foscari University of Venice
Venezia, Italy
luccio@unive.it

Stefano Mercogliano

University of Naples Federico II
Napoli, Italy
stefano.mercogliano@unina.it

Abstract—Trusted Execution Environments (TEEs) on resource-constrained microcontrollers are an emerging area of interest, yet they present unique security challenges, particularly in managing encrypted code execution through limited secure memory. This paper presents a formal verification approach for *Umbra*, a TEE framework for ARM TrustZone-M, currently under development, that implements secure caching mechanisms to execute encrypted enclaves from flash memory. We employ model checking techniques to formally analyze critical security properties, including data isolation between secure and non-secure worlds, integrity of the Enclave Flash Block Cache (EFBC), and resilience against identified threats such as Direct Memory Access (DMA) handover attacks and timing-based side channels. Our threat model considers privileged attackers in the non-secure world and compromised host operating systems, analyzing vulnerabilities in DMA reconfiguration windows and context switch dependencies. Through formal modeling, we identify replay and timing side-channel attacks; by introducing countermeasures, these guarantees are restored in the model.

Index Terms—ARM TrustZone-M, Trusted Execution Environment, Formal Verification, Model Checking

I. INTRODUCTION

Microcontrollers play an increasingly critical role in domains such as robotics, automotive systems, aerospace, and military applications [1]–[5]. These platforms now execute increasingly sophisticated workloads, often supported by complex operating systems. At the same time, their critical role in ensuring system security has grown significantly. Traditional hardware mechanisms such as Memory Protection Units (MPUs) are insufficient to address today’s security demands. A prominent example is ARM’s TrustZone-M [6], which enables the deployment of TEEs on microcontrollers.

Research on embedded platforms is more limited than in server- and desktop-grade systems. Differently from prior work that analyzes existing commercial technology, in this paper we verify *Umbra* [7] [8], a novel open-source framework for trusted execution on TrustZone-M-enabled microcontrollers, currently under development.

Umbra provides comprehensive security features, including enclaves, strong isolation between secure and non-secure

worlds, and protection of integrity and confidentiality against both software- and hardware-level threats.

The verification of *Umbra* is particularly challenging due to the complexity of its enclave lifecycle, which depends on a delicate interplay between heterogeneous memory resources, and secure decryption and integrity checks. To address these challenges, we formally model *Umbra*’s secure caching architecture using a process algebra and analyze it with the mCRL2 toolchain [9], [10]. Our verification effort focuses on integrity of the EFBC, and resilience against DMA handover and timing side channels. Through model checking, we identify issues corresponding to real attacks, which we mitigate by incorporating appropriate countermeasures into the specification. We then demonstrate that *Umbra*’s architectural design, enhanced with these countermeasures, enforces the intended security guarantees throughout system evolution.

II. BACKGROUND

Umbra is a Rust-based kernel designed to enable the construction of software TEEs independently of any existing software stack, while maintaining both strong security guarantees and high performance. *Umbra* provides a compatibility layer that allows a host OS, such as an RTOS, to execute third-party secure applications developed independently, and enforces only the minimal set of security checks while delegating bulkier management tasks to the host OS. Application enclaves are encrypted in the non-secure flash and organized by *Umbra* into logical flash partitions called Enclave Flash Blocks (EFBs). Figure 1 shows *Umbra* secure caching architecture.

The host kernel interacts with enclaves via a set of APIs, referred to as `teecalls`, exposed through a shared Non-Secure Callable space. *Umbra* redefines the allocated memory region as secure, which includes a dedicated executable region called the EFBC that holds one or more EFBs, decrypted and ready to execute. In addition to the main SRAM, the secure SRAM, physically distinct and reserved exclusively for *Umbra*, hosts the implementation of `teecalls` as well as the Enclave Swap Space (ESS).

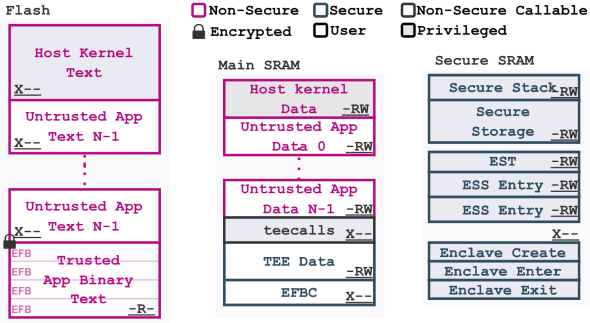


Fig. 1. Umbra secure caching architecture: encrypted EFBs reside in non-secure flash; EFBC in secure main SRAM; and ESS in secure secondary SRAM.

This secure memory plays a central role in both **Umbra**'s performance and its security model. Each time the host OS deschedules an enclave, **Umbra** configures the cryptographic accelerators, which have direct DMA access to secure memory, to decrypt enclave blocks from flash and “cache them” into secure SRAM. From there, the decrypted blocks are speculatively transferred into the EFBC.

III. THREAT MODEL

The Secure World (SW) of TrustZone-M, where **Umbra** runs the enclave code, is trusted in its core design. The Non Secure World (NSW), where the host operating system and untrusted applications run, is instead under the control of the attacker and we assume that attackers can gain elevated privileges on it. Attackers can also control the host OS and manipulate shared resources like DMA channels or hardware timers. Our analysis focuses on whether interference from the NSW can weaken **Umbra**'s guarantees. A privileged NSW adversary can configure DMA requests from the NSW as they wish, read flash contents (including old encrypted blocks), and cause `memfault` exceptions by manipulating enclave execution requests. We also consider a compromised host OS that can access the shared hardware timer used by the TEE to schedule enclave context switches, create contention or delays in validation, and trigger `memfault` during specific validation sequences.

We highlight two significant threats. The first is a DMA-based replay attack: during DMA reconfiguration, a privileged NSW attacker exploits a brief handover window to inject a stale block, which may become executable, violating freshness guarantees. The second threat involves a timer-driven side channel: a compromised host OS saturates the shared timer to make validation delays noticeable, then measures the resulting delay and infers which EFB the enclave requested based on timing.

IV. FORMAL SPECIFICATION

The formal specification of **Umbra**'s secure caching mechanism is written in mCRL2 [8]–[10]. The model includes the Enclave Executor, EFBC, ESS, Validation Engine, and DMA Controller, seen as communicating processes with actions for validation, execution, and exception handling. We analyze

TABLE I
VERIFICATION OF THE MODEL WITH THE SECURITY MONITOR.

Description	Duration (s)
Timing attack with SecurityMonitor	21.974
Replay attack with SecurityMonitor	30.533

critical security properties, including data isolation between secure and non-secure worlds, integrity of EFBC, and resilience against DMA handover and timing-based side channels, using model checking with the mCRL2 toolchain.

V. RESULTS

We made several simplifying assumptions in the model to keep the verification problem tractable and avoid state-space explosion. First, we limited the number of enclave flash blocks to a small, fixed set and restricted the sizes of the EFBC and ESS. We simplified the DMA controller by using fewer progress states and modeled cryptographic functions such as encryption and HMAC as atomic operations. These simplifications preserve the core idea of the secure caching mechanism while keeping model checking manageable.

Before verifying attacks, a set of sanity checks is used to ensure that the model correctly represents the expected behavior. These checks confirm that blocks already present in EFBC can always be executed without additional validation, that cache misses are consistently resolved via ESS, and that DMA transfers complete without leaving channels in dead-locked states.

Automatic verification detected two attack traces, corresponding to a timing side-channel attack and a replay attack, showing that the initial design does not maintain the promised security guarantees across all possible system evolutions. In order to prevent these attacks, the specification was updated with a proactive containment mechanism. We added a Security Monitor that keeps track of the global security state, continuously monitors system events, and upon detecting a suspicious action updates the security state to *compromised*. The Security Monitor immediately locks the system, blocking all critical operations such as DMA transfers, block validation, and replay attempts, preventing the attack before it can succeed. We rechecked the model enriched with the Security Monitor and verified that the previously identified attacks were prevented; verification times are reported in Table I.

VI. CONCLUSION

We formally verified the security of **Umbra**, a lightweight TEE framework for ARM TrustZone enabled microcontrollers, using the mCRL2 toolchain. In particular, we encoded **Umbra**'s secure caching architecture using mCRL2's process algebra, and then model checked it against our a real-world threat model (cf. Sec. III). The verification process showed that the original design is vulnerable to replay and timing side-channel attacks, and we confirmed their feasibility on the actual **Umbra** implementation. Looking ahead, we plan to investigate unlinkability and non-interference properties to rigorously assess the integrity and security of EFBs, progressively moving towards a fully verified TEE architecture.

REFERENCES

- [1] J. O. Hamblen and G. M. Van Bekkum, "An embedded systems laboratory to support rapid prototyping of robotics and the internet of things," *IEEE Transactions on Education*, vol. 56, no. 1, pp. 121–128, 2012.
- [2] S. Sonko, C. D. Daudu, F. Osasona, A. M. Monebi, E. A. Etukudoh, and A. Atadoga, "The evolution of embedded systems in automotive industry: A global review," *World Journal of Advanced Research and Reviews*, vol. 21, no. 2, pp. 096–104, 2024.
- [3] L. Lo Bello, R. Mariani, S. Mubeen, and S. Saponara, "Recent advances and trends in on-board embedded and networked automotive systems," *IEEE Transactions on Industrial Informatics*, vol. 15, no. 2, pp. 1038–1051, 2018.
- [4] D. C. Sharp, A. E. Bell, J. J. Gold, K. W. Gibbar, D. W. Gvillo, V. M. Knight, K. P. Murphy, W. C. Roll, R. G. Sampigethaya, V. Santhanam *et al.*, "Challenges and solutions for embedded and networked aerospace software systems," *Proceedings of the IEEE*, vol. 98, no. 4, pp. 621–634, 2010.
- [5] B. Schott and R. Parker, "Defense applications for large numbers of distributed small sensors," in *IEEE MTT-S International Microwave Symposium Digest, 2005*. IEEE, 2005, pp. 369–372.
- [6] ARM, "TrustZone for Cortex-M," 2025, <https://www.arm.com/technologies/trustzone-for-cortex-m>.
- [7] S. Mercogliano and A. Cilaro, "Umbra: An Efficient Framework for Trusted Execution on Modern TrustZone-Enabled Microcontrollers," in *2025 Design, Automation & Test in Europe Conference (DATE)*, 2025, pp. 1–2.
- [8] "Umbra Project GitHub Repository," 2025, <https://github.com/HiSA-Team/umbra>.
- [9] J. F. Groote, A. Mathijssen, M. Reniers, Y. Usenko, and M. van Weerdenburg, "The Formal Specification Language mCRL2," in *Methods for Modelling Software Systems (MMOSS)*, ser. Dagstuhl Seminar Proceedings (DagSemProc), E. Brinksma, D. Harel, A. Mader, P. Stevens, and R. Wieringa, Eds., vol. 6351. Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2007, pp. 1–34. [Online]. Available: <https://drops.dagstuhl.de/entities/document/10.4230/DagSemProc.06351.12>
- [10] "The mCRL2 toolset," 2025, <https://mcrl2.org/>.