RV-TEE for Post-Quantum Digital Signatures: Trustworthy Dilithium Deployment

Adriano d'Alessandro
adrianodalessandro@yahoo.it
Università degli studi di Modena e Reggio Emilia
Modena, Italy

Angelo Ferrando
angelo.ferrando@unimore.it
Università degli studi di Modena e Reggio Emilia
Modena, Italy

Christian Colombo
christian.colombo@um.edu.mt
University of Malta
Msida, Malta

Abstract

Analytical security of cryptographic protocols does not immediately translate to operational security: incorrect implementation and runtime attacks can undermine even a formally sound design. The RV-TEE framework has been applied to address this gap across a growing range of protocols, from classical session-oriented protocols such as SSH and TLS, to post-quantum constructions including group authenticated key exchange based on CRYSTALS-Kyber. With the standardisation of CRYSTALS-Dilithium as ML-DSA under FIPS 204, attention now turns to the trustworthy deployment of post-quantum digital signatures — a class of cryptographic primitive not yet addressed within the RV-TEE line of work. Unlike the session-oriented and key exchange protocols studied previously, Dilithium is a stateless signature scheme: its operations — key generation, signing, and verification — are atomic and self-contained, calling for a different characterisation of what constitutes correct and secure execution. Relevant runtime properties shift away from protocol sequencing and towards computational integrity, parameter validity, and the faithful execution of lattice-based arithmetic.

CCS Concepts

• **Security and privacy** → **Digital signatures**; **Logic and verification**; *Hardware security implementation*; *Trusted computing*; • **Hardware** → Quantum computation.

Keywords

Cryptography, Post-Quantum Cryptography Runtime Verification, Hardware Security Modules

ACM Reference Format:
Adriano d'Alessandro, Angelo Ferrando, and Christian Colombo. 2026. RV-TEE for Post-Quantum Digital Signatures: Trustworthy Dilithium Deployment. In *Proceedings of (Vortex 2026).* https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 Introduction

Modern cryptography is based on the principle of provable security, in which mathematical guarantees are established within carefully defined theoretical models. These models intentionally omit implementation details, in order to reason about the security of a primitive element in isolation. In this sense, a security proof provides strong confidence in the design of the cryptographic "vault" but does not automatically guarantee the correctness of its development and, consequently, the conditions under which it is used.

In practice, cryptographic security is a chain linking the theoretical model, the software implementation, and the runtime environment. A theoretically secure scheme can still fail if its implementation is flawed, and even a carefully designed implementation can behave unexpectedly when run in a particularly complex environment. In this regard, the gap between the careful mathematical abstraction and the actual execution of the algorithm becomes a security issue for users.

Post-quantum cryptography (PQC) has solid theoretical foundations, but its behaviour is still poorly understood. This uncertainty has become an urgent issue given the advent of the first quantum machines, supported also by the recent standardisation by NIST of the **ML-DSA (FIPS 204)** algorithm, a digital signature scheme that will be the cornerstone of quantum-resistant security.

The main challenge is that lattice-based cryptography is significantly more complex to implement correctly than historically known alternatives. It relies on complex mathematical operations, such as polynomial ring arithmetic and rejection sampling. If these operations are not performed consistently with the specifications, recent studies have demonstrated that ML-DSA is vulnerable to a variety of implementation-level threats: single fault injections can bypass randomized signing to recover secret keys [8] or force the verification routine to accept forged signatures [3]. Furthermore, side-channel analysis has proven highly effective, with attackers exploiting power leakages not only from the final signature but even from the rejected sampling trials inherent to the design [9, 18]. These vulnerabilities highlight a fundamental truth: adopting a mathematically secure algorithm is not enough if the execution environment allows internal arithmetic or parameter boundaries to be manipulated.

Bridging this gap is a challenge that requires mechanisms to observe the behaviour of cryptographic primitives during execution and verify that security properties continue to be maintained during execution. Rather than relying solely on design guarantees, the aim is to provide an additional safeguard that the implemented system behaves consistently with the assumptions on which the original security was based.

Runtime Verification (RV) offers a different approach: instead of simply hoping that the code is well written and properly tested,

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
Vortex 2026, Bruxelles, Belgium
© 2026 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 978-x-xxxx-xxxx-x/YYYY/MM
https://doi.org/10.1145/nnnnnnn.nnnnnnn

it "observes" it as it runs. A monitor can check critical internal variables in real time — such as the coefficients needed for ML-DSA signatures — ensuring that they comply with the parameter bounds defined by the standard. As a motivating example, consider a high-stakes document signing scenario such as an executive authorising a legally binding contract or a software provider pushing a critical security update. Even if the underlying signature scheme is mathematically secure against quantum computers, a bad actor could still try to tamper with the signing process as it executes to forge the approval. If the monitor detects something going wrong, it can immediately take action, safeguarding the integrity of the sensitive signing process.

In this work, we adapt the RV-TEE framework to post-quantum signatures. Our primary goal is to see if runtime monitoring can provide a genuine layer of protection without bogging down the system with expensive new hardware or complex operating system modifications.

The remainder of this paper is organized as follows:
Section 2 provides the necessary background, outlining the operational challenges of ML-DSA and introducing the principles of hardware-isolated monitoring (RV-TEE). In Section 3, we present our core architecture and monitor design, describing how cryptographic operations are segmented and how critical security properties are classified. Section 4 situates our work within the existing literature, emphasising the shift from monitoring session-based protocols to securing stateless post-quantum signature schemes. Finally, Section 5 concludes the article by summarising our current progress and discussing future directions, including resilience testing and the gradual relaxation of our core trust assumptions.

2 Background

Even when a cryptographic scheme has a robust formal security proof, its physical implementation can introduce a wide range of errors. Low-level details, hardware quirks, and unexpected execution contexts can create vulnerabilities that completely bypass the theoretical design.

While this classification was originally applied to interactive protocols like TLS or SSH, the transition to post-quantum cryptography (PQC) introduces an entirely new dimension to these threats. The standardisation of ML-DSA shifts the focus of runtime security away from protocol sequentiality and firmly towards computational integrity.

2.1 ML-DSA Overview

ML-DSA (Module-Lattice Digital Signature Algorithm), standardised under **FIPS 204** [13], is a lattice-based digital signature scheme originating from CRYSTALS-Dilithium. It relies on the difficulty of the module learning with errors (MLWE) problem. From a system and architectural perspective—which is critical when deploying within a Trusted Execution Environment (TEE)—ML-DSA presents specific computational challenges.

First, its core operations require heavy polynomial arithmetic over module lattices. To maintain performance on resource-constrained platforms, ML-DSA uses the Number Theoretic Transform (NTT) to perform polynomial multiplications in $O(n \log n)$ time [1]. Additionally, the scheme relies extensively on the Keccak-based SHAKE extendable-output function for public matrix generation and challenge hashing [13]. Together, NTT and symmetric hashing dominate the execution time and represent primary targets for hardware acceleration—and consequently, for potential attackers.

Second, the signing algorithm follows the "Fiat–Shamir with Aborts" paradigm [10]. In this setting, signature generation is inherently probabilistic: intermediate values are sampled and then checked against strict bounds. If these bounds are not satisfied, the candidate signature is discarded, and the process is restarted. This rejection sampling mechanism is essential to prevent statistical leakage of the secret key, but also introduces a potentially unbounded number of iterations. As a result, the control flow of the signing procedure becomes data-dependent and non-deterministic, which poses a significant challenge when attempting to securely monitor execution within a TEE.

Building on the foundations laid by Vella et al. [15], we adopt a multi-layered architectural philosophy. Our runtime verification serves a dual purpose: it continuously checks that the post-quantum arithmetic executes correctly to maintain design-level security, while simultaneously acting as an active monitor that scrutinises any data crossing the TEE's trust boundary.

2.2 Runtime Verification and Control

In security-critical systems, Runtime Verification (RV) moves beyond design-time checks to actively observe the dynamic execution of a system. By instrumenting the code, the relevant events and internal variables are exposed to an external monitor generated from formal specifications. This ensures that the assumptions made during the design phase actually hold when the system is running.

However, our goal extends beyond passive monitoring, where the runtime monitor simply detects deviations from the formal specification [6] giving a verdict, we explore how runtime *controllers*[1] can actively protect post-quantum primitives by manipulating the execution flow. If the system enters an unsafe state, the monitor intercepts the process and prevents further damage, actively adapting the execution to enforce security properties at runtime [7].

This capability is vital for lattice-based algorithms. Because they manage massive internal states and complex arithmetic, detecting and preventing abnormal execution *before* it compromises a signature is far more valuable than simply logging a failure after the fact.

2.3 The RV-TEE Paradigm

The ultimate goal for executing security-critical post-quantum arithmetic is isolation within a Trusted Execution Environment (TEE) [15]. Traditionally, the industry relies on CPU-based TEEs (like Intel SGX or ARM TrustZone) or tamper-evident Hardware Security Modules (HSMs). However, CPU-TEEs have proven vulnerable to complex microarchitectural exploits (e.g., Plundervolt, Foreshadow) and often require heavy operating system modifications [4, 11, 12, 16].

In contrast, the RV-TEE approach offers a more flexible deployment model by combining Runtime Verification with an HSM. By

[1] We use terminology in line with [5] where "controllers" refers to a wide range of runtime reactions which are neither strictly "enforcers" (i.e. they may not completely prevent property violation), nor necessarily "reparators" (i.e. they may react before the actual violation).

offloading complex lattice-based operations to a dedicated security module, we process sensitive cryptographic material within a controlled trust domain. The RV monitor sits at the boundary, inspecting all data flows between the trusted and non-trusted environments.

3 Architecture and Monitor

When securing complex cryptographic algorithms like ML-DSA, deciding where to place the runtime monitor is just as critical as deciding what to monitor. We can generally observe a system at a few different levels: down at the hardware layer, at the firmware layer, or higher up where the application and middleware meet.

Due to the complexity of lattice-based schemes, a natural first approach would be to monitor the firmware execution to ensure both the correctness of the process and the compliance with the required security properties.

Even if observing the firmware execution may result in a perfect step-by-step observation, this strategy may not be feasible in practice. Hardware constraints can significantly limit the ability to instrument or observe execution, and the firmware environment itself often lacks the mechanisms necessary to support comprehensive monitoring. Consequently, we introduce a tiered trust model [2] to ensure that cryptographic analysis remains effective despite hardware constraints. This framework guarantees that the properties we extract are consistent with our monitoring objectives, allowing them to be directly translated into the monitor's specification language

The core of our approach is captured in Figure 1, which illustrates the RV-TEE framework. Because of the hardware limitations we have divided our architecture into two distinct domains using a strict trust boundary: the Rich Execution Environment (REE) on the left, where the runtime monitor is deployed, and the Trusted Execution Environment (TEE) on the right, which serves as a secure enclave for the post-quantum primitives.

To ensure the validity of this framework, we establish two core assumptions:

(1) **Encrypted Communication Channel:** All data exchanged over the USB channel is fully encrypted using keys stored exclusively within the HSM, preventing man-in-the-middle attacks or data leakage.
(2) **Validation of Optimised Primitives:** The monitored post-quantum algorithms have undergone necessary arithmetic optimisations (e.g., NTT and INVNTT loop fusion) to achieve higher performance on embedded hardware.

Interaction with the secure device is facilitated by a middleware that can be considered as a Rich Application (RA) that is able to bridge the communication originating from the Rich Execution Environment (REE) with the Trusted Execution Environment (TEE) and vice versa. This layer enables the exposure of high-level cryptographic operations that can be performed within the TEE.

A major operational challenge when deploying ML-DSA within a resource-constrained HSM is the sheer computational and memory footprint of the algorithm. To overcome these hardware limitations, the execution of the signature scheme cannot be offloaded as a single monolithic black-box request. Instead, the algorithm has to be segmented into smaller and manageable cryptographic steps.

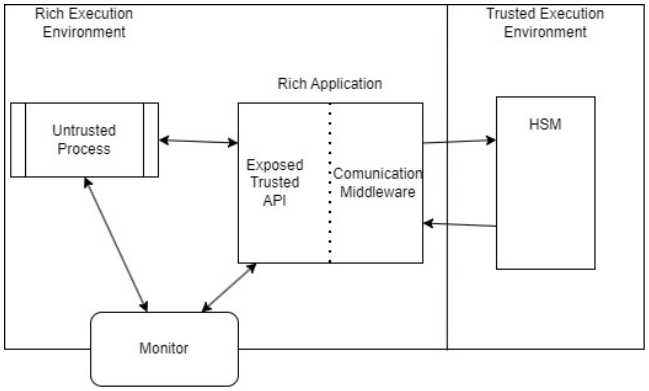


Figure 1: Architectural overview of the proposed RV-TEE framework. The Runtime Verification (RV) monitor is deployed within the Rich Execution Environment (REE).

The C++ middleware exposes just the high level functionality, so that these segmented operations are invisible to the user.

This architectural segmentation not only makes the execution feasible on the HSM but also provides a natural advantage for runtime verification. We instrument the Rich Application (RA) by selecting dedicated hooking points at the boundaries of these specific function calls. As the REE invokes each step of the segmented ML-DSA process, these hooks seamlessly expose the execution flow, feeding information directly to the runtime monitor.

3.1 Security Properties and Controlling Capabilities

Many vulnerabilities in this domain stem not from communication flows but from invalid parameters, improper randomness handling, or the runtime manipulation of complex polynomial operations.

The properties described before can be synthesised in the following two macro categories:

- **Parameter Validity**: Ensuring that the cryptographic inputs and parameters strictly adhere to the bounds defined by the FIPS 204 [13] specification.
- **Computational Integrity**: Verifies that mathematical operations are executed faithfully and without deviations, protecting the opacity of polynomial operations.

The security properties identified for ML-DSA operations represent the specific set of rules that our monitor must follow. However, not all failures are created equal: a formatting error in a domain string is a procedural flaw, whereas a mathematical anomaly in a signature could signal a fault-injection attack aimed at exposing parts of the private key.

To navigate these differences, we categorize each property by its Severity. This metric represents the potential impact on the system's security if a property is violated, ranging from Critical to Low, depending on the extent of the damage and the systemic nature of the consequences.

In Table 1, we explore the most significant properties alongside their impact and severity levels. The security properties evaluated for the runtime monitor within the RV-TEE framework are

specifically designed to address the unique challenges posed by post-quantum digital signatures. They serve as a proactive defense against known vulnerabilities and attacks targeting cryptographic algorithms and primitives:

- **P1 (Key Structure Validation):** Before the Trusted Execution Environment (TEE) begins processing a cryptographic key, the monitor verifies its structural integrity.
  Impact (High Severity): This validation prevents an attacker from supplying malformed or oversized keys, effectively malicious payloads, designed to trigger parsing errors, memory corruption, or **Buffer Overflows** within the trusted core. By enforcing this property, the monitor may block potential **Remote Code Execution (RCE)** attempts at the ingestion stage. Although P1 is a vital defence against implementation-level attacks, it focuses on the "form" of the data; therefore, it does not inherently protect against higher-level logical bypasses, such as the algorithmic flaws seen in **CVE-2022-21449** [14].
- **P2 (Domain Separation):** This property ensures that a digital signature generated for a specific purpose, such as a software update, cannot be maliciously "replayed" or reused in a completely different context. By binding the signature to its intended scope, the monitor prevents an attacker from lifting a valid credential from one domain and applying it to another.
  Impact (Medium Severity): While the underlying cryptographic algorithm remains mathematically secure, a failure here allows for **Context Confusion Attacks**[17]. An attacker might capture a signature from a legitimate financial transaction and resubmit it to bypass application logic in an unauthorised different context. The system may erroneously accept the signature as proof of intent for an action that the signer never actually authorised in that specific domain.
- **P3 (Infinity Norm Bounds):** In lattice-based cryptography, the coefficients of a valid signature must strictly fall within predefined boundaries. If an attacker injects a fault during the signing process to bypass these checks, the resulting anomalous signature can directly leak the private key, or significant portions of it [8]. The monitor acts as a definitive safety measure by inspecting the signature's boundaries and intercepting any malformed outputs before they leave the secure environment.
  Impact (Critical Severity): Even a single signature that violates the infinity norm bound can leak enough information to reconstruct the Secret Key. This allows an attacker to forge any subsequent signature for that user, rendering the cryptographic identity permanently compromised and unsafe for further use.
- **P4 (Verify-after-Sign):** Even highly secure environments can suffer from microscopic hardware issues or induced computational faults, and to counteract this phenomenon, the monitor may force a check: Once a signature is generated, the system uses the corresponding public key to verify it internally. If this verification fails, it immediately signals that the internal arithmetic was somehow corrupted during execution, generating an unsafe signature that should not be used. **Impact (High severity):** Without a validation, if the HSM has been attacked or has glitched for any reason, it may release a flawed signature that may not be verifiable or, in a worst case scenario, can be exploited by an attacker to try and retrieve part of the secret key used to sign.

Together, these properties define the perimeter of secure execution. The monitor does not observe prolonged state transitions but constantly guards the validity bounds and execution integrity of each individual set of operations.

Table 1: Monitored security properties for ML-DSA operations

ID	Property	Severity	Enforceability
P1	ML-DSA key structures validated prior to TEE execution	High	Controllable
P2	Pre-hash domain separation	Medium	Controllable
P3	Signature infinity norm bounds verified before release	Critical	Controllable
P4	Mandatory Verify-after-Sign to detect TEE computation faults	High	Detectable

4 Related Work

The transition to post-quantum cryptography has sparked a massive amount of research. However, while the mathematical foundations of these new algorithms are thoroughly scrutinised, their real-world behaviour under execution remains a critical area of study.

4.1 The RV-TEE Framework

Our research is based directly on the RV-TEE framework originally proposed by Vella et al. [15], which bridges the gap between software monitoring and hardware isolation. By placing the runtime monitor at the boundary of a trusted module, RV-TEE gains the tamper-resistance of an HSM while maintaining the flexibility of formal verification.

However, previous applications of the RV-TEE framework have focused almost exclusively on session-oriented, interactive protocols, such as TLS and SSH. These protocols rely heavily on state machines and message sequencing. Our work addresses a distinct gap in the literature by applying this architecture to a completely different cryptographic paradigm: stateless, lattice-based digital signatures. In this new context, the challenge is no longer about tracking communication states, but rather about ensuring the absolute integrity of complex polynomial arithmetic and strict parameter compliance during isolated, atomic operations.

5 Conclusion and Future Work

In this paper, we adopted the RV-TEE framework to support post-quantum digital signature operations, focusing on the ML-DSA standard. Because this signature scheme is stateless by design, we demonstrated that security monitoring can no longer rely on session tracking; instead, it must evolve to ensure computational integrity, parameter validity, and the faithful execution of lattice-based arithmetic.

We are currently in the process of porting the ML-DSA algorithm itself into a Hardware Security Module (HSM). In parallel, the practical implementation of the monitoring framework is underway.

Crucially, this monitor resides outside the TEE, operating instead within a high-trust middle layer of the Rich Execution Environment (REE). This architectural choice is key for policy enforcement: by intercepting and validating requests at the REE level, the monitor acts as a proactive guardian preventing malformed inputs from ever reaching the HSM, thereby shielding the trusted environment from requests that could otherwise exhaust its limited memory or trigger execution faults.

Once both components are fully deployed, our initial goal will be to evaluate the monitor's performance overhead when overseeing these highly resource-constrained HSM operations. However, beyond simply measuring execution time and memory footprint, we aim to empirically test the framework's resilience against some forms of active threats.

Future work will involve subjecting the C++ middleware (Trusted Application) to simulated Denial-of-Service (DoS) attacks. This will allow us to verify that the monitor can accurately identify and mitigate resource exhaustion attempts before they can impact the HSM. Furthermore, our objective is to systematically relax the core trust assumptions established in this paper, effectively shrinking the trusted boundary and exposing the framework to a more hostile real-world operational environment.

References

[1] Amin Abdulrahman, Felix Oberhansl, Hoang Nguyen Hien Pham, Jade Philipoom, Peter Schwabe, Tobias Stelzer, and Andreas Zankl. 2025. Towards ML-KEM & ML-DSA on OpenTitan. In *2025 IEEE Symposium on Security and Privacy (SP)*. 1–19. doi:10.1109/SP61157.2025.00220

[2] Robert Abela, Christian Colombo, Axel Curmi, Mattea Fenech, Mark Vella, and Angelo Ferrando. 2023. Runtime Verification for Trustworthy Computing. *Electronic Proceedings in Theoretical Computer Science* 391 (Sept. 2023), 49–62. doi:10.4204/eptcs.391.7

[3] Sven Bauer and Fabrizio De Santis. 2025. Forging Dilithium and Falcon Signatures by Single Fault Injection. Cryptology ePrint Archive, Paper 2025/2029. https://eprint.iacr.org/2025/2029

[4] Jo Van Bulck, Marina Minkin, Ofir Weisse, Daniel Genkin, Baris Kasikci, Frank Piessens, Mark Silberstein, Thomas F. Wenisch, Yuval Yarom, and Raoul Strackx. 2018. Foreshadow: Extracting the Keys to the Intel SGX Kingdom with Transient Out-of-Order Execution. In *27th USENIX Security Symposium (USENIX Security 18)*. USENIX Association, Baltimore, MD, 991–1008. https://www.usenix.org/conference/usenixsecurity18/presentation/bulck

[5] Christian Colombo, Gordon J. Pace, and Gerardo Schneider. 2021. Runtime Verification: Passing on the Baton. In *Formal Methods in Outer Space - Essays Dedicated to Klaus Havelund on the Occasion of His 65th Birthday (Lecture Notes in Computer Science)*, Ezio Bartocci, Yliès Falcone, and Martin Leucker (Eds.). Springer, 89–107. doi:10.1007/978-3-030-87348-6_5

[6] Yliès Falcone, Srđan Krstić, Giles Reger, and Dmitriy Traytel. 2021. A taxonomy for classifying runtime verification tools. *International Journal on Software Tools for Technology Transfer* 23 (04 2021). doi:10.1007/s10009-021-00609-z

[7] Yliès Falcone, Laurent Mounier, Jean-Claude Fernandez, and Jean-Luc Richier. 2011. Runtime Enforcement Monitors: composition, synthesis, and enforcement abilities. *Formal Methods in System Design* 38, 3 (June 2011), 223–262. doi:10.1007/s10703-011-0114-4

[8] Elisabeth Krahmer, Peter Pessl, Georg Land, and Tim Güneysu. 2024. Correction Fault Attacks on Randomized CRYSTALS-Dilithium. Cryptology ePrint Archive, Paper 2024/138. https://eprint.iacr.org/2024/138

[9] Zheng Liu, An Wang, Congming Wei, Yaoling Ding, Jingqi Zhang, Annyu Liu, and Liehuang Zhu. 2025. Release the Power of Rejected Signatures: An Efficient Side-Channel Attack on the ML-DSA Cryptosystem. Cryptology ePrint Archive, Paper 2025/582. https://eprint.iacr.org/2025/582

[10] Vadim Lyubashevsky. 2009. Fiat-Shamir with Aborts: Applications to Lattice and Factoring-Based Signatures. In *Proceedings of the 15th International Conference on the Theory and Application of Cryptology and Information Security: Advances in Cryptology* (Tokyo, Japan) *(ASIACRYPT '09)*. Springer-Verlag, Berlin, Heidelberg, 598–616. doi:10.1007/978-3-642-10366-7_35

[11] Mathias Morbitzer, Manuel Huber, Julian Horsch, and Sascha Wessel. 2018. SEVered: Subverting AMD's Virtual Machine Encryption. In *Proceedings of the 11th European Workshop on Systems Security (EuroSys '18)*. ACM, 1–6. doi:10.1145/3193111.3193112

[12] Kit Murdock, David Oswald, Flavio D. Garcia, Jo Van Bulck, Daniel Gruss, and Frank Piessens. 2020. Plundervolt: Software-based Fault Injection Attacks against Intel SGX. In *2020 IEEE Symposium on Security and Privacy (SP)*. 1466–1482. doi:10.1109/SP40000.2020.00057

[13] National Institute of Standards and Technology. 2024. *FIPS 204: Module-Lattice-Based Digital Signature Standard.* Standard. U.S. Department of Commerce. doi:10.6028/NIST.FIPS.204 Finalized August 2024.

[14] NVD. 2022. CVE-2022-21449: Oracle Java SE and GraalVM "Psychic Signatures" Vulnerability. https://nvd.nist.gov/vuln/detail/cve-2022-21449. Accessed: 2026-04-07.

[15] Mark Vella, Christian Colombo, Robert Abela, and Peter Špaček. 2021. RV-TEE: secure cryptographic protocol execution based on runtime verification. *Journal of Computer Virology and Hacking Techniques* 17 (09 2021). doi:10.1007/s11416-021-00391-1

[16] Qifan Wang and David Oswald. 2026. Confidential Computing on Heterogeneous CPU-GPU Systems: Survey and Future Directions. *ACM Comput. Surv.* 58, 9, Article 230 (Feb. 2026), 35 pages. doi:10.1145/3793532

[17] Mingming Zhang, Xiaofeng Zheng, Kaiwen Shen, Ziqiao Kong, Chaoyi Lu, Yu Wang, Haixin Duan, Shuang Hao, Baojun Liu, and Min Yang. 2020. Talking with Familiar Strangers: An Empirical Study on HTTPS Context Confusion Attacks. In *CCS '20: 2020 ACM SIGSAC Conference on Computer and Communications Security, Virtual Event, USA, November 9-13, 2020*, Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna (Eds.). ACM, 1939–1952. doi:10.1145/3372297.3417252

[18] Yuhan Zhao, Wei Cheng, Zehua Qiao, Yuejun Liu, and Yongbin Zhou. 2026. Rejection Matters: Efficient Non-Profiling Side-Channel Attack on ML-DSA via Exploiting Public Templates. Cryptology ePrint Archive, Paper 2026/056. https://eprint.iacr.org/2026/056