1,2Mircea ȚĂLU

A REVIEW OF ADVANCED TECHNIQUES FOR DATA PROTECTION IN WEBASSEMBLY

1. The Technical University of Cluj–Napoca, Faculty of Automation and Computer Science, Cluj–Napoca, ROMANIA
2. SC Accesa It Systems SRL, Cluj–Napoca, ROMANIA

Abstract: In an era where data breaches and cyber threats are increasingly prevalent, securing data has become a critical concern for developers and users alike. WebAssembly (or Wasm) has emerged as a highly efficient and secure framework for running code across multiple platforms. To ensure data protection in Wasm applications, various cryptographic techniques, Trusted Execution Environments (TEEs), and privacy–preserving methods have been integrated, addressing the growing demand for secure cloud–edge computing and IoT systems. This review provides a comprehensive analysis of these advanced techniques, comparing different approaches in terms of efficiency, security features, and compatibility with Wasm. The review explores the synergies between cryptographic techniques and TEEs, highlighting their role in protecting sensitive data while maintaining system performance. This work aims to offer insights into the latest advancements in data protection within Wasm environments, emphasizing the importance of security in modern distributed systems.

Keywords: WebAssembly, Cryptography, Trusted Execution Environments (TEEs), Data Security, Homomorphic Encryption, Post–Quantum Cryptography

1. INTRODUCTION

WebAssembly (Wasm) emerged in 2015 as a transformative approach aimed at enhancing the performance and security of web–based applications. The initiative, spearheaded by Brendan Eich, sought to address the limitations of asm.js by providing a more efficient, low–level, assembly–like language optimized for web environments [1–3]. WebAssembly was developed by a consortium of leading technology companies, including Mozilla, Microsoft, Apple, and Google, to overcome the limitations of traditional web technologies. Wasm enables developers to compile high–level programming languages into a compact, platform–independent format that can run natively on various environments, offering near–native performance, enhanced security, and cross–platform portability. Since its introduction, WebAssembly has seen rapid adoption, becoming an official W3C standard in 2019 and establishing itself as a vital element in modern web and cloud ecosystems [4–7]. With the rapid expansion of cloud–edge computing, IoT, and embedded systems, safeguarding data integrity, confidentiality, and privacy has become a top priority.

At its core, WebAssembly is designed with three principal goals: performance, security, and portability. Its performance stems from the ability to take advantage of modern hardware features and optimize code execution, ensuring that web applications perform efficiently even under heavy computational loads. From a security perspective, Wasm emphasizes memory safety, isolated execution, and sandboxing, creating a robust environment where malicious activities are minimized. Its portability further reinforces the flexibility of Wasm, enabling it to be utilized across various hardware architectures and platforms, including embedded systems, browsers, and cloud–edge computing environments.

WebAssembly offers a binary instruction format that allows code written in languages like C, C++, Rust, and others to be executed in a sandboxed environment. This design ensures both speed and security while supporting a growing array of programming languages, including AssemblyScript, C#, Go, F#, Dart, Go, Kotlin, Swift, D, Pascal, Zig, and Grain [8–10].

The architecture of a WebAssembly binary is modular, comprising a collection of functions, global variables, and linear memory that can be efficiently executed through a stack–based machine model. Within this framework, an embedder – such as a host JavaScript engine –manages the loading and execution of Wasm modules, facilitating interactions with the host system and handling input/output, timers, and error management. Although WebAssembly was intended to provide secure execution, it is still susceptible to a range of security vulnerabilities that can be exploited by attackers. These vulnerabilities originate from various fundamental aspects of the WebAssembly framework, its integration with host environments, and its operational dynamics within web browsers. Key security issues identified include memory safety flaws, side–channel attack vectors, vulnerabilities related to speculative execution, and the complexities involved in its interaction with JavaScript [11–13].

This review discusses advanced techniques for data protection in WebAssembly, summarizing key works in the field, comparing their approaches, and presenting the latest updates.

2. RESEARCH METHODOLOGY

A comprehensive survey was conducted to assess the growing emphasis into the latest advancements in data protection within Wasm environments based on five key steps:

a) Developing review questions;
b) Identifying relevant literature;
c) Assessing the quality of the studies;
d) Summarizing the collected evidence; and
e) Analyzing the research outcomes.

Recognizing the practical significance of this field, the review spans the years 2018 to 2024. It examines numerous journal articles that explore fundamental concepts and practical applications in this domain.

Cryptographic Techniques in WebAssembly

Cryptographic methods are at the heart of data protection in WebAssembly environments. Several works focus on the integration of cryptographic libraries with Wasm to ensure secure storage, sharing, and transmission of data across heterogeneous platforms. One prominent example is the client–side encrypted storage system introduced by Sun et al. (2020), which leverages WebAssembly and the Web Cryptography API for data protection across platforms, enhancing security during data sharing without relying on external encryption solutions. While client–side encryption offers a potential solution, existing methods face three main challenges: inadequate security due to low–entropy PINs, cumbersome data sharing with traditional algorithms, and poor usability due to reliance on specific software or plugins. WebCloud addresses these issues and introduces additional features, such as robust user revocation, fast data processing through offline encryption and outsourced decryption, and compatibility with any device using a web browser. Built on ownCloud for file management, WebCloud incorporates WebAssembly and the Web Cryptography API for cryptographic operations. Comprehensive testing across various browsers, Android, and PC applications demonstrates its cross–platform efficiency. Additionally, WebCloud's design incorporates a practical ciphertext–policy attribute–based key encapsulation mechanism (CP–AB–KEM) scheme, which can be applied in other contexts [14].

Homomorphic encryption is another critical method, allowing computation on encrypted data without revealing the data itself. Attrapadung et al. [15] introduced an efficient two–level homomorphic public–key encryption scheme in prime–order bilinear groups, capable of supporting polynomially many homomorphic additions and one multiplication over encrypted data in WebAssembly. This scheme parallels the Boneh, Goh, and Nissim (BGN) cryptosystem from TCC 2005, which operates in composite–order bilinear groups. Their work improved upon the Freeman scheme from Eurocrypt 2010, the current standard for two–level homomorphic encryption in prime–order groups, enhancing efficiency across nearly all aspects while maintaining identical ciphertext sizes. The scheme is notably straightforward, functioning as a concatenation of two ElGamal encryptions "in the exponent" within asymmetric bilinear groups [15].

Table 1. The cryptographic approaches used in WebAssembly, comparing their efficiency, portability, and specific application areas.

Cryptographic method	Approach	Efficiency	Portability	Applications
Sun et al. [14]	Client–side encrypted storage	High	High	Cross–platform storage
Attrapadung et al. [15]	Homomorphic public–key encryption	Moderate	Moderate	Secure cloud computing
Riera et al. [16]	Client–side hashing library	High	High	Password hashing

Riera et al. [16] proposed Clipaha, a client–side hashing scheme designed to enable high–security password hashing on resource–constrained server devices. This approach addresses the limitations of modern password hashing algorithms, such as Argon2, which typically require significant computational power and memory, making them unsuitable for IoT devices. Clipaha offers enhanced resilience against a wider range of attacks and accommodates complex usage scenarios not addressed by prior methods. Their implementation as a web library demonstrates a 50% performance improvement over similar libraries, successfully running on devices where previous solutions fail.

Trusted Execution Environments (TEEs) in WebAssembly

Trusted Execution Environments (TEEs) offer a secure and isolated execution space designed to protect sensitive data and computational processes from potentially compromised operating systems and malicious software. These environments leverage hardware–based security features to ensure that the integrity and confidentiality of the executed code and the data it processes are maintained, even in the presence of a hostile environment.

In the context of WebAssembly, TEEs are utilized to establish secure enclaves that effectively safeguard data against unauthorized access. WebAssembly, being a low–level bytecode format, allows for the efficient execution of code across various platforms while maintaining performance and security. By integrating TEEs with WebAssembly, developers can ensure that sensitive operations – such as cryptographic key management or secure data processing – are executed within a protected environment that is isolated from the broader system, including the operating system and other applications. This synergy between TEEs and WebAssembly not only enhances security by providing a robust defense against a variety of threats, including side–channel attacks and code injection vulnerabilities, but also facilitates the development of privacy–preserving applications. As a result, the use of TEEs in conjunction with WebAssembly enables a new paradigm of secure computing that empowers developers to create applications that maintain user privacy and data integrity, thereby fostering trust in cloud–based and distributed computing environments.

Zhao et al. [17] proposed the integration of Trusted Execution Environments (TEEs) with WebAssembly to create secure enclaves that protect sensitive data and computations from potentially compromised operating systems and malicious software. This combination leverages the isolated execution space of TEEs to ensure the confidentiality and integrity of data processed within WebAssembly applications. By utilizing TEEs, developers can perform critical operations – such as cryptographic key management – safely, thus enhancing security against threats like side–channel attacks and code injection vulnerabilities. This approach enables the development of privacy–preserving applications, fostering greater trust in cloud–based and distributed computing [17].

Almstedt et al. [18] discussed the widespread integration of IoT devices in industrial applications, yet highlight persistent challenges in their implementation within rural regions, particularly concerning privacy, data integrity, accountability, and ownership of data when processed at the edge. To tackle these concerns, they propose ContractBox, a system that facilitates accountable and secure data sharing based on a publisher–subscriber model while employing trusted computing at the edge. ContractBox utilizes a Trusted Execution Environment (TEE) to ensure the confidentiality and integrity of client data and code. Additionally, it harnesses WebAssembly to execute smart contracts within a secure, isolated environment, safeguarding both the host and associated smart contracts from malicious actions. The system guarantees the immutability and accountability of published data by storing it on a blockchain. The authors demonstrate that ContractBox can manage thousands of publications per second with various payload types and support multiple smart contract runtimes on a single edge device, achieving up to 35 times greater throughput than a comparable Hyperledger Fabric deployment.

Ménétrey et al. [19] examined the significance of publish/subscribe systems in facilitating communication among numerous devices in distributed architectures. While these systems are widely used, their security often compromises portability in favor of enhanced integrity and attestation guarantees. Trusted Execution Environments (TEEs) offer a solution through enclaves that improve security; however, application development for TEEs is complex and often tied to specific architectures, limiting flexibility.

The authors propose a novel approach using WebAssembly to create a portable, fully attested publish/subscribe middleware system for secure distributed communication. Their implementation includes a broker running within Intel SGX, utilizing established frameworks like MQTT and TLS. Their enhanced TLS protocol protects attestation information, and experimental results show a 1.55× decrease in message throughput with the trusted broker. They have made their work open–source to encourage reproducibility in research [19].

Table 2. A comparison of different TEE implementations, highlighting their security features, speed, and compatibility with Wasm.

TEE Implementation	Features	Security Level	Speed	Compatibility with Wasm
Zhao et al. [17]	Enclave snapshot, rewinding, nested attestation	High	High	Excellent
Almstedt et al. [18]	Publish–subscribe middleware in Intel SGX	High	Moderate	Good
Ménétrey et al. [19]	Trustworthy publish–subscribe middleware in Intel SGX	High	Moderate	Good
Qiang et al. [20]	Two–way sandbox with Intel SGX	High	Moderate	Excellent

Qiang et al. [20] investigated serverless computing, a rising trend in cloud environments that allows users to deploy applications and process data without managing servers. However, this framework faces trust issues as neither cloud users nor providers can be fully trusted. To address these concerns, they introduce Se–Lambda, a serverless computing framework that enhances security by using an SGX enclave to protect the API gateway and a two–way sandbox combining the SGX enclave with a WebAssembly sandbox for the service runtime. In this model, untrusted user code is contained within the WebAssembly sandbox, while the SGX enclave safeguards privacy–sensitive data from malicious cloud providers. Additionally, a privilege monitoring mechanism is implemented in the SGX enclave to regulate access control for user function modules. Prototyped based on the open–source OpenLambda project, Se–Lambda demonstrates minimal performance impact while significantly improving security.

Privacy–Preserving Methods

Privacy–preserving methods in WebAssembly are essential for safeguarding user data while maintaining optimal system performance. As web applications increasingly handle sensitive information, such as personal data, financial records, and health–related details, the need for robust privacy measures becomes paramount. WebAssembly enables the execution of code at near–native speed, which allows developers to implement complex cryptographic algorithms and data protection techniques efficiently. By leveraging the capabilities of WebAssembly, privacy–preserving methods can operate in a secure, sandboxed environment, ensuring that sensitive data remains protected from unauthorized access. For instance, techniques such as client–side encryption can be effectively implemented within a WebAssembly module, allowing users to encrypt their data before it is transmitted over the internet. This approach minimizes the risk of data exposure, even if the network is compromised. Moreover, the use of WebAssembly facilitates the implementation of advanced privacy–preserving technologies, such as zero–knowledge proofs and homomorphic encryption, which enable computations on encrypted data without revealing the underlying information. These methods provide an additional layer of security, ensuring that user privacy is upheld without sacrificing the performance and responsiveness expected from modern web applications.

Table 3. Privacy–preserving methods in WebAssembly, focusing on encryption strength, quantum–resistance, and overall performance.

Privacy–Preserving method	Encryption strength	Quantum resistance	Performance
Seo et al. [21] (Crystals–Kyber)	High	Yes	High
Qiang et al. [20] (Two–way sandbox)	High	No	Moderate
Sun et al. [14] (Encrypted storage)	High	No	High

Seo et al. [21] addressed the security threats posed by quantum computers executing Shor's algorithm to public key algorithms by presenting a portable and efficient implementation of the Crystals–Kyber key encapsulation mechanism (KEM), which is the sole KEM algorithm selected in the NIST Post–Quantum Cryptography competition. Their implementation utilizes WebAssembly (Wasm) to enhance portability while maintaining performance, combining JavaScript for the overall structure and Wasm for performance–critical operations, such as secure hash algorithm–3–based processes and polynomial multiplication. They further optimized the number theoretic transform (NTT)–based polynomial multiplication using single instruction multiple data (SIMD) capabilities in Wasm, leading to significant performance improvements. Benchmarks show that their implementation outperforms the latest JavaScript reference version by up to 4.02 times in key generation, encapsulation, and decapsulation across major browsers, marking it as the first Kyber implementation using Wasm technology in a web environment [21].

Integration of Cryptography and TEEs

Integrating cryptographic methods with Trusted Execution Environments (TEEs) creates a robust framework for enhancing the security of WebAssembly (Wasm) applications [22]. TEEs provide an isolated execution space that protects sensitive data and operations from malicious actors, including compromised operating systems or external threats. This isolation is crucial for applications that handle confidential information, such as personal data, financial transactions, or proprietary algorithms. When cryptographic methods are employed alongside TEEs, several key benefits emerge:

- Data Protection – Cryptography safeguards data confidentiality and integrity. By encrypting data before it enters the TEE, developers can ensure that sensitive information remains secure, even if the host environment is compromised. This is particularly important in cloud–based applications, where data is transmitted over potentially insecure networks.
- Secure Key Management – TEEs can securely store cryptographic keys, protecting them from unauthorized access. This secure key management is essential for maintaining the effectiveness of cryptographic algorithms, as the exposure of keys can lead to significant security breaches.
- Enhanced Trust – By combining cryptographic protocols with TEEs, developers can create systems that not only execute code securely but also provide guarantees of data authenticity and integrity. This integration allows for attestation mechanisms, enabling users and systems to verify that the code being executed within the TEE has not been tampered with and is running in a secure environment.
- Performance Optimization – While cryptographic operations can be computationally intensive, TEEs are designed to perform these tasks efficiently. The secure execution environment can leverage hardware acceleration features to enhance performance, allowing cryptographic methods to be implemented without significant overhead.
- Support for Compliance – As data protection regulations become more stringent, integrating cryptographic methods with TEEs can help organizations meet compliance requirements. The ability to securely manage and process sensitive data can aid in adhering to standards such as the General Data Protection Regulation (GDPR) or the Health Insurance Portability and Accountability Act (HIPAA).

Table 4. Correlations between cryptographic techniques and TEE features in Wasm environments, illustrating how these two security layers complement each other.

Cryptographic method	TEE integration	Memory safety	Performance	Application area
Riera et al. [16]	Compatible with Intel SGX	High	High	Password management
Seo et al. [21]	Post–quantum secure TEEs	High	High	Quantum–safe encryption
Sun et al. [14]	Encrypted storage + TEE	High	Moderate	Secure data storage

M. Ţălu [23] provided valuable insights into vulnerability discovery in WebAssembly binaries, enhancing the understanding of data protection techniques in this context.

3. CONCLUSION

Advanced techniques for data protection in WebAssembly environments are continually evolving to meet the challenges of emerging technologies such as quantum computing, cloud–edge infrastructure, and secure distributed systems. Cryptographic techniques, Trusted Execution Environments (TEEs), and privacy–preserving methods are crucial to achieving robust security for WebAssembly applications. This review highlights several innovative approaches that demonstrate the potential of Wasm in secure environments, comparing their effectiveness, performance, and application areas.

References

[1] B. Eich. From asm.js to webassembly. https://brendaneich.com/2015/06/from–asm–js–to–webassembly/, 2008
[2] Mozilla. asm.js. http://asmjs.org/, 2013
[3] G. Perrone, S.P. Romano, WebAssembly and Security: a review, arXiv:2407.12297v1.
[4] WebAssembly official documentation. Available at: https://webassembly.org/ (accessed September 30, 2024).

[5] A. Rossberg, B.L. Titzer, A. Haas, D.L. Schuff, D. Gohman, L. Wagner, A. Zakai, J.F. Bastien, M. Holman. Bringing the Web Up to Speed with WebAssembly. Communications of the ACM, 61(12): 107 – 115

[6] P. Mendki. Evaluating Webassembly Enabled Serverless Approach for Edge Computing, 2020 IEEE Cloud Summit, Harrisburg, PA, USA, 2020, pp. 161–166

[7] M.N. Hoque, K.A. Harras. WebAssembly for Edge Computing: Potential and Challenges. IEEE Commun. Stand. Mag. 2022, 6, 68–73.

[8] P.P. Ray. An Overview of WebAssembly for IoT: Background, Tools, State–of–the–Art, Challenges, and Future Directions. Future Internet, 15(8): 275, 2023

[9] D. Lehmann, J. Kinder, M. Pradel. Everything old is new again: Binary security of WebAssembly. In Proceedings of the 29th USENIX Security Symposium (USENIX Security 20), Boston, MA, USA, 12–14 August 2020; pp. 217–234.

[10] Y. Yan, T. Tu, L. Zhao, Y. Zhou, W. Wang. Understanding the performance of webassembly applications. In Proceedings of the 21st ACM Internet Measurement Conference, Virtual Event, 2–4 November 2021; pp. 533–549.

[11] J. Dejaeghere, B. Gbadamosi, T. Pulls, F. Rochet. Comparing Security in eBPF and WebAssembly. In Proceedings of the ACM SIGCOMM 1st Workshop on eBPF and Kernel Extensions, New York City, NY, USA, 10 September 2023.

[12] M. Kim, H. Jang Y. Shin. Avengers, Assemble! Survey of WebAssembly Security Solutions, 2022 IEEE 15th International Conference on Cloud Computing (CLOUD), Barcelona, Spain, 2022, pp. 543–553

[13] J. Sun, D.Y. Cao, X.M. Liu, Z. Zhao, W.W. Wang, X.L. Gong. SELWasm: A Code Protection Mechanism for WebAssembly, 2019 IEEE Intl Conf on Parallel & Distributed Processing with Applications, Big Data & Cloud Computing, Sustainable Computing & Communications, Social Computing & Networking (ISPA/BDCloud/SocialCom/SustainCom), Xiamen, China, 2019, pp. 1099–1106

[14] S. Sun, H. Ma, Z. Song, R. Zhang. Webcloud: Webbased cloud storage for secure data sharing across platforms. IEEE Transactions on Dependable and Secure Computing, 19(3):1871 – 1884, 2022

[15] N. Attrapadung, G. Hanaoka, S. Mitsunari, Y. Sakai, K. Shimizu, T.i Teruya. Efficient two–level homomorphic encryption in prime–order bilinear groups and a fast implementation in webassembly. In Proceedings of the 2018 on Asia Conference on Computer and Communications Security, ASIA CCS '18. ACM, May 2018

[16] F.I. Riera, M. Almgren, P. Picazo–Sanchez, C. Rohner. Clipaha: A scheme to perform password stretching on the client. In Proceedings of the 9th International Conference on Information Systems Security and Privacy – ICISSP, pp. 58–69. INSTICC, SciTePress, 2023

[17] S. Zhao, P. Xu, G. Chen, M. Zhang, Y. Zhang, Z. Lin. Reusable enclaves for confidential serverless computing. In 32nd USENIX Security Symposium, USENIX Security 2023, vol. 6, pp. 4015–4032, 2023

[18] L. Almstedt, K. Bleeke, M. Mahhouk, L. Jehl, R. Kapitza, L. Wolf. Contractbox: Realizing accountable data sharing on the edge using a small scale blockchain. Computer Networks, 229, 2023

[19] J. Ménétrey, A. Grüter, P. Yuhala, J. Oeftiger, P. Felber, M. Pasin, V. Schiavoni. A Holistic Approach for Trustworthy Distributed Systems with WebAssembly and TEEs. In 27th International Conference on Principles of Distributed Systems (OPODIS 2023). Leibniz International Proceedings in Informatics (LIPIcs), vol. 286, pp. 23:1–23:23, Schloss Dagstuhl – Leibniz–Zentrum für Informatik (2024)

[20] W. Qiang, Z. Dong, H. Jin. Se–Lambda: Securing Privacy–Sensitive Serverless Applications Using SGX Enclave. In: Beyah, R., Chang, B., Li, Y., Zhu, S. (eds) Security and Privacy in Communication Networks. SecureComm 2018. Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering, vol 254. Springer, Cham.

[21] S.C. Seo, H.S. Kim. Portable and efficient implementation of crystals–kyber based on webassembly. Computer Systems Science and Engineering, 46(2): 2091–2107, 2023

[22] WebAssembly official documentation. Available at: https://webassembly.org/ (accessed September 30, 2024).

[23] M. Ţălu, A Review of Vulnerability Discovery in WebAssembly Binaries: Insights from Static, Dynamic, and Hybrid Analysis, ANNALS of Faculty of Engineering Hunedoara, International Journal of Engineering, Hunedoara, 2024. In press.

ISSN 1584 – 2665 (printed version); ISSN 2601 – 2332 (online); ISSN-L 1584 – 2665