A Holistic Approach for Trustworthy Distributed Systems with WebAssembly and TEEs

Jämes Ménétrey ✉ ⓘ
University of Neuchâtel, Switzerland

Aeneas Grüter ⓘ
University of Bern, Switzerland

Peterson Yuhala ✉ ⓘ
University of Neuchâtel, Switzerland

Julius Oeftiger ⓘ
University of Bern, Switzerland

Pascal Felber ✉ ⓘ
University of Neuchâtel, Switzerland

Marcelo Pasin ✉ ⓘ
University of Neuchâtel, Switzerland

Valerio Schiavoni ✉ ⓘ
University of Neuchâtel, Switzerland

Abstract

Publish/subscribe systems play a key role in enabling communication between numerous devices in distributed and large-scale architectures. While widely adopted, securing such systems often trades portability for additional integrity and attestation guarantees. Trusted Execution Environments (TEEs) offer a potential solution with enclaves to enhance security and trust. However, application development for TEEs is complex, and many existing solutions are tied to specific TEE architectures, limiting adaptability. Current communication protocols also inadequately manage attestation proofs or expose essential attestation information. This paper introduces a novel approach using WebAssembly to address these issues, a key enabling technology nowadays capturing academia and industry attention. We present the design of a portable and fully attested publish/subscribe middleware system as a holistic approach for trustworthy and distributed communication between various systems. Based on this proposal, we have implemented and evaluated in-depth a fully-fledged publish/subscribe broker running within Intel SGX, compiled in WebAssembly, and built on top of industry-battled frameworks and standards, i.e., MQTT and TLS protocols. Our extended TLS protocol preserves the privacy of attestation information, among other benefits. Our experimental results showcase most overheads, revealing a $1.55\times$ decrease in message throughput when using a trusted broker. We open-source the contributions of this work to the research community to facilitate experimental reproducibility.

1 Introduction

Publish/subscribe (pub/sub) systems [18] have become foundational for seamless intercommunication of a wide range of devices, from IoT ecosystems to large-scale cloud-based services, i.e. the cloud-edge continuum [65]. These systems enable efficient and scalable data

distribution among distributed entities by decoupling data producers from data consumers. Given their widespread adoption, notably in cloud computing [3, 24, 43], several pub/sub systems have been proposed with the clear goal of providing additional privacy guarantees [50]. However, the nodes participating in these systems are implicitly trusted, relegating security concerns primarily to the protection of communication channels or leveraging heavyweight cryptographic primitives [31]. This limited security approach leaves data on the processing components vulnerable to potential threats, especially in decentralised and heterogeneous environments. Notably, high-privileged actors within the pub/sub nodes, i.e., operating system or hypervisor, may compromise the confidentiality and integrity of data. Similarly, they could leak critical cryptographic material, e.g., private keys of certificates. Leaking certificate keys is especially concerning as these keys serve as the foundation for the authentication process among pub/sub participants, thereby putting user privacy and data integrity at stake.

In both the consumer market and cloud providers, trusted execution environments (TEEs) present a solution to strengthen the integrity and confidentiality of data in use, especially on nodes that may not be inherently trustworthy. TEEs provide *enclaves*, e.g., hardware-protected memory regions, where sensitive computations are completely isolated from other software executed on the same platform. Such secure enclaves significantly elevate the security posture of systems like pub/sub, safeguarding not just the communication but also the processing and data on such nodes from malicious actors. As a keystone feature of TEEs, attestation enables a remote entity to verify the authenticity, configuration, and state of a trusted environment using cryptographic proofs, ensuring the enclave runs the intended software without being tampered with or compromised [40]. In pub/sub systems deployed over untrusted infrastructures, attestation ensures data and its processing within the TEE enclave are kept confidential and untampered.

Nonetheless, developing applications for TEEs is challenging due to their specific programming paradigms and SDKs, requiring massive efforts when writing or porting existing software [25]. In addition, while many pub/sub systems exploit TEEs to protect data in use (covered in §3.1), these usually tie closely to a specific TEE architecture, which is limiting in the heterogeneous environments (e.g. using different CPUs) considered in this work. Beyond this, current secure communication protocols fail to transport attestation proofs, do not maintain the privacy of these messages when supported, or cannot expose X.509 certificates issued by global authorities (covered in §3.2). This leads to the use of ad-hoc implementations, which fall short in offering a consistent solution across the cloud-edge continuum.

In this paper, we solve the major challenges associated to writing secure and portable pub/sub systems by relying on WebAssembly (Wasm), an open-standard binary instruction format. Wasm's architecture-neutral design abstracts the complexity of hardware and TEE requirements, making it particularly suitable as a compilation target for pub/sub systems in heterogeneous hardware environments, such as the cloud-edge continuum. We further protect the communication channels by extending the industry standard TLS protocol for embedding attestation evidence in the handshake while maintaining compatibility with the original specifications. This ensures the authenticity of the executing code of parties within the pub/sub system, thereby mitigating the risk of malicious entities impersonating or modifying these components. Additionally, certificate keys are safeguarded within the TEEs, preventing potential leaks. Besides, we developed a proof-of-concept that encapsulates the full implementation of a standard pub/sub broker and a TLS library, enabling the termination of TLS channels directly in the TEE, which has been proven complex in prior work [8]. We achieve this by using Wasm as a compilation target for both software, limiting the number of code changes required to make them compile and run within TEEs. While our prototype is focused on cloud environments by leveraging Intel SGX, we outline the

trusted primitives required for our proposal to be compatible with other platforms, including edge and IoT devices. We summarise our contributions as follows: *(1)* a unified strategy that secures a standard pub/sub system using TEEs with attestation for security, and compiled in Wasm for seamless cloud-edge communication while minimising code changes, *(2)* an extension to the TLS communication protocol, facilitating the confidential exchange of attestation evidence, thereby affirming the authenticity of actors within the pub/sub system and *(3)* an open-source implementation of a pub/sub broker using Intel SGX for cloud environments, with a suite of benchmarks aimed at evaluating the impacts of Wasm, the TEE and attestation, in comparison with state-of-the-art work. Our evaluation reveals that our system delivers messages at $1.55\times$ slower than baseline throughput, yet provides portability and the robust security guarantees of TEEs.

2   Background

2.1   Pub/sub systems

A publish/subscribe system (often called pub/sub) is an asynchronous architecture for message passing that connects two different types of entities: publishers and subscribers. Publishers send messages (or events), being unaware of who is interested in receiving such messages. Messages sent are usually marked with an arbitrary type or a category. Subscribers express their interest in specific types of messages and receive them when they are published. Pub/sub is commonly used in event-driven distributed systems such as the Internet-of-things, since they simplify the communication between different entities.

Pub/sub messages are usually passed from publishers to subscribers using an intermediary broker. Brokers receive messages from publishers and forward them to subscribers who have expressed interest in the corresponding topics. Message brokers are responsible for routing and delivering messages to the appropriate subscribers. Examples of message brokers include Apache Kafka, RabbitMQ, MQTT brokers, and cloud-based pub/sub services like Amazon SNS (Simple Notification Service) or Google Cloud Pub/Sub.

One of the key advantages of pub/sub is decoupling since publishers and subscribers do not need to know each other. Pub/sub systems can also be designed to be highly scalable and fault-tolerant by using replicated, interconnected brokers. This flexible architecture allows components to be added or removed without affecting the system.

Security in pub/sub systems is dealt with when publishers and subscribers connect to their brokers. Brokers then usually implement authentication and access control before publishing or subscribing to messages. They can also establish encrypted connections (e.g. TLS) to implement message privacy. By implementing attestation, we offer a stronger guarantee for all the components. Publishers, subscribers, and brokers are guaranteed to be who they say they are, and all can ensure that their counterparts execute appropriate (correct) software.

2.2   Trusted execution environments

Trusted execution environments (TEEs) offer secure and isolated areas of a computer system. TEEs shield sensitive data and code from unauthorised access, compromised libraries or operating systems. This isolation is hardware-enforced, considerably minimising the software-based vulnerabilities. In contrast to trusted platform modules (TPMs) [1], which are limited to a specific and finite set of operations, TEEs enable the execution of arbitrary code, providing greater flexibility and versatility. Code executed inside a TEE can communicate with untrusted programs or libraries, but such interactions are tightly controlled. TEE-specific mechanisms ensure data is secure during transfers into and out of an enclave.

We focus on Intel Software Guard Extensions (SGX) [15, 47], a popular and widely used TEE architecture. Available in server-grade CPUs, Intel SGX manages and runs code within secure enclaves, which interact with the other processes and operating system by transitioning the enclave boundary using specific instructions, i.e., `ECALLs` and `OCALLs`. Enclaves use `ECALLs` to receive function calls from untrusted applications outside the enclave and `OCALLs` to invoke functions outside from within the enclave. The enclave's memory is encrypted by a dedicated MMU, ensuring protection against attackers with access to memory data.

2.3   Attestation

Attestation is a security mechanism enabling an *attester* to prove its identity and integrity to an *verifier*. In the realm of TEEs, attestation allows a TEE to prove its configuration, identity, and state to another device or service. Most TEE implementations support attestation, either built-in (as in SGX [32] and its successor TDX [59, 14], AMD SEV-SNP [4]) or by means of research prototypes [42]. The primary objective is to ensure the attester is genuine, unmodified, and trustworthy. To achieve this objective, attestation leverages proofs (i.e. *evidence*), a cryptographically signed document composed of *claims*, i.e., pieces of asserted information, like the hash of a code running in the TEE, i.e., *measurement*. The verifier is bootstrapped with *reference values*, compared against the received claims for validation.

In Intel SGX, attestation is paramount for establishing the authenticity and integrity of code and data inside an enclave. Intel controls a set of keys, intrinsically linked to the hardware and identity of the enclave. Such keys are inaccessible from the enclaves themselves, and are used by the processor and Intel signed software to issue trustworthy evidence. Intel is considered trusted, as they provision a subset of keys into the hardware. Within this assumption, third parties can trust evidence produced by the TEE.

2.4   Communication channel and attestation binding

Communication channel.   Over the past decades, several standards have emerged to establish trustworthy communication between two entities. Among them, Transport Layer Security (TLS) stands out as the most prominent protocol for such tasks. This cryptographic protocol ensures confidentiality, integrity, and authentication for secure communication. A typical TLS session begins with a phase known as the *handshake*. This TLS handshake acts as a negotiation process, where the two parties decide on encryption settings and authenticate one another before exchanging secure data. More precisely, the latest version of the TLS handshake (1.3 at the time of this writing) is composed of *1)* the *ClientHello* sent by the client, notably containing the supporting cipher suites, key agreement and anti-replay mechanisms, *2)* the server sends the *ServerHello* response, detailing analogous parameters, while signifying the handshake's conclusion, and finally *3)* the handshake wraps up as the client reciprocates with a similar acknowledgement, accompanied by data from a higher-level protocol, that is wrapped within TLS, e.g., HTTPS. Additionally, during steps 2 and 3, both entities can opt to present an X.509 certificate. These serve as identity validators, embedding trust through a chain of digital signatures rooted in trusted Certificate Authorities (CAs). Building on this, many existing attestation protocols that bind with TLS typically augment the handshake, the certificate, or a combination of the two, facilitating the negotiation of security parameters and the exchange of attestation evidence.

Channel binding.   Channel binding [7] ensures that an entity participating in a secure communication, typically via protocols such as TLS, is indeed the entity that has undergone attestation. It establishes that the communications are with the attested environment,

preventing possible relay attacks where a malicious party might relay attestation challenges to a genuine system and then claim the legitimate evidence as their own. To secure trusted communication, attestation evidence is typically integrated into the early phases of the communication protocol. However, combining attestation with these protocols introduces challenges, including increased latency and additional trade-offs like binding the system to a particular TEE technology.

As later outlined in this paper, the proposed solution refines the TLS protocol with minimal enhancements. Unlike traditional approaches that embed attestation in custom fields in the broker's X.509 certificates, our method leverages a custom TLS 1.3 encrypted extension. This approach reduces the need for additional communication roundtrips between the client and server. It preserves the broker's ability to use certificates issued by recognised CAs, an aspect overlooked in previous studies.

2.5 WebAssembly

WebAssembly (Wasm) [27] is an open standard that provides a portable and executable bytecode, tailored for running applications at near-native speed. While its initial design targeted web applications written in traditional native languages like C/C++, Rust, and Go, its remarkable performance has led to its adoption not just within browsers but also in standalone environments [27, 74]. Wasm was developed as a cross-platform, language-agnostic compilation target, optimised for easy integration in resource-limited environments, making it a prime candidate for TEE development. Notably, several specialised runtimes, such as TWINE [41, 17] for Intel SGX and WATZ [42] for Arm TrustZone, have been developed to leverage Wasm's capabilities within state-of-the-art TEE implementations, and natively support key TEE primitives such as attestation. For Wasm applications, particularly in standalone settings, the WebAssembly System Interface (WASI) [44] provides direct access to essential OS services, traditionally managed by POSIX and system calls. WASI facilitates seamless communication with a handful of system components, ranging from file systems and networking to time management and random number generation, regardless of the underlying platform. Runtimes optimised for TEEs map these system calls to trusted services, typically offered by the TEE manufacturers, ensuring that sensitive data remains secure and protected within the enclave.

We leverage Wasm as a versatile, cross-platform environment to model our pub/sub proposal seamlessly across the cloud-edge continuum and within TEEs as the execution context. This approach offers unmatched portability across many processor and TEE architectures. When combined with TEEs, Wasm hardens our solution against powerful threats, such as the operating system, the hypervisor or malicious system administrators. This security guarantee is especially important in edge computing scenarios where the local infrastructure typically lacks physical security measures as established in cloud environments.

3 Related Work

3.1 Pub/sub with TEEs

In the dynamic world of distributed systems, pub/sub mechanisms have consistently gained traction, acting as the core foundation for a variety of applications and architectures. Many approaches have been conducted in research to bring dependability and safety regarding pub/sub systems, and in many different directions [72]. An explored aspect is encryption schemes of data transferred through the brokers, preserving the privacy of the communicated

Table 1 Comparison of the state-of-the-art pub/sub systems shielded by TEEs. ●, ◐, ○ mean fully, partially and not implemented, respectively. ❓ denotes not disclosed details. Features description: *1) Communication protocol*: protocol used by peers to interact with the broker. *2) Fully enclaved broker*: broker operates within the TEE instead of only securing specific components within the secure environment. *3) Peer authentication*: broker authenticates the peers while establishing communication. *4) Peer attestation*: broker attests the peers while establishing communication. *5) Broker authentication*: peers authenticate the broker while establishing communication. *6) Broker attestation*: peers attest the broker while establishing communication; ◐ means that only publishers are attested. *7) Persistence of messages*: system's capability to store messages for future delivery (e.g., when subscribers might be temporarily offline). *8) Idiomatic pub/sub architecture*: system adheres to the principles of the pub/sub paradigm. *9) Open source*: implemented solution is freely available to the public via an open-source repository. *10) TEE technology*: denotes the TEE used by the proposed system, or its capacity to operate agnostically across various TEEs.

	[53] SCBR	[6] PubSub-SGX	[73] MagikCube	[61] MQT-TZ	[52] Pei et al.	This work
Comm. protocol	TLS	TLS	TLS	TLS	❓	TLS
Fully enclaved broker	●	●	●	○	○	●
Peer authentication	○	●	●	●	❓	●
Peer attestation	○	○	○	○	❓	●
Broker authentication	●	●	●	●	❓	●
Broker attestation	◐	●	●	○	❓	●
Persistence of messages	○	●	○	○	○	●
Idiomatic pub/sub arch.	○	●	●	●	●	●
Open source	○	○	○	●	○	●
TEE technology	SGX	SGX	SGX	TZ	SGX	*Agnostic*

TEEs references: SGX (Intel SGX), SNP (AMD SEV-SNP), TDX (Intel TDX), TZ (Arm TrustZone)

information [29, 46, 10, 12, 38, 21, 11]. Cryptographic-based privacy protection schemes focus on encrypting events and subscriptions, and then performing ciphertext matching between them. However, they often suffer from scalability issues as matching time complexity grows with the number of subscriptions, leading to diminishing performance.

More recently, researchers have investigated the potential of Intel SGX as a secure environment for confidential data processing. Leveraging TEEs has shown potential for enhanced performance over cryptography-based methods, as highlighted by SCBR [53]. Their study details a custom content-based routing engine operating within an SGX enclave. PubSub-SGX [6] introduced a scalable approach, using a load balancer that manages multiple matchers, each operating within individual enclaves. Following this, MagikCube [73] added an authentication service to the pub/sub system, thereby enhancing broker trust among publishers and subscribers using SGX. Finally, Pei et al. [52] further refined this paradigm by optimising subscription matching times using cryptographic methods, with SGX facilitating comparison tasks but does not disclose how secrets are exchanged. In contrast with previous work, our proposal prioritises establishing an initial trust across all pub/sub participants by mutual attestation. We do it by encapsulating conventional pub/sub systems within TEEs, ensuring genuine execution environments and trustworthy implementations.

Other prior studies have chosen a different strategy using TrustZone, Arm's TEE. In this setup, the device is divided into the *normal world* (the standard OS) and the *trusted world* (the TEE). MQT-TZ [61] migrated the broker's data management component within this trusted world. Publishers and subscribers negotiate symmetric keys with the broker,

which are generated inside the TEE. This approach ensures that data is encrypted during transmission. Conversely, our proposal enhances the threat model of MQT-TZ by relying on entity attestation in the pub/sub system and hosting the TLS endpoint directly within the TEE, avoiding handling cryptographic materials outside the TLS protocol.

Comparison.   Table 1 offers a comprehensive summary of state-of-the-art proposals for securing pub/sub systems with TEEs, focusing on features relevant to our study.

3.2   Binding Communication Protocols and Attestation using TEEs

Research has extensively explored the integration of secure communication protocols with attestation. While our focus primarily lies on solutions leveraging TEEs, we also recognise the significant contributions from prior work that leveraged TPMs as trust anchors [23, 64, 22, 63, 5, 78, 9, 51, 70, 69]. Readers can refer to [33, 48] for a comprehensive review of these works. Our analysis omits explicitly works that discuss attestation through TEEs but do not bind attestation evidence to a communication channel [39, 16, 20, 76].

Intel proposed a remote attestation protocol for key exchange based on SIGMA [34, 28], binding SGX enclaves with communication channels. Subsequent solutions aimed to establish trusted communication channels with enclaves using custom message exchanges [62, 35, 42].

Table 2 Comparison of the state-of-the-art channel binding solutions. ●, ○ mean fully and not implemented, respectively. ❓ denotes not disclosed details. "—" denotes not applicable comparisons. Features description: *1) Baseline protocol*: protocol upon which the proposal is built. *2) No change in TLS specification*: proposal respects the TLS specifications, ensuring compatibility with pre-existing TLS implementations. *3) Attestation privacy*: protocol maintains confidentiality of attestation evidence. *4) Mutual attestation*: communicating entities engage in a mutual attestation process. *5) Evidence per session*: each communication session is uniquely associated with specific attestation evidence. *6) Attestation Privacy*: all attestation-related messages are encrypted; ◑ means that only a portion of the messages remains confidential. *7) TEE-agnostic*: independent of any specific programming language or TEE-specific SDK; ◑ means theoretical approach proposed, but no agnostic implementation. *8) Endpoint in enclave*: endpoint application fully resides within the TEE. *9) Support global CAs*: broker-displayed certificates can be vouched for by globally recognised CAs. *10) Open source*: implemented solution is freely available to the public via an open-source repository. *11) TEE technology*: denotes the TEE that the attestation API currently supports, or its capacity to operate agnostically across various TEEs.

	[28] SGX EPID	[62] Shepherd et al.	[42] WaTZ	[33] RA-TLS	[26] PALÆMON	[48] TSL	This work
Baseline protocol	Custom	Custom	Custom	TLS	TLS	TLS	TLS
No change in TLS spec.	—	—	—	●	●	●	●
Attestation privacy	○	●	○	○	●	●	●
Mutual attestation	○	●	○	●	○	●	●
Evidence per session	●	●	●	○	○	●	●
Endpoint in enclave	●	●	●	●	●	○	●
Attestation privacy	○	◑	○	◑	❓	◑	●
TEE-agnostic	○	◑	◑	○	○	◑	●
Support global CAs	—	—	—	○	○	○	●
Open-source	●	○	●	●	○	○	●
TEE technology	SGX	TZ	TZ	SGX	SGX	*Agnostic*	*Agnostic*

TEEs references: SGX (Intel SGX), SNP (AMD SEV-SNP), TDX (Intel TDX), TZ (Arm TrustZone)

While these initial efforts in remote attestation provided valuable insights, their custom nature makes them challenging to integrate into existing software. In contrast, our approach harnesses TLS, the leading industry standard for secure communication.

More recently, further research [66, 26, 48, 33, 49] have suggested using TLS for communication and modifying the X.509 certificates in order to include additional fields related to attestation. Although this method takes advantage of the protocol's standardisation to address the earlier concern, it ties the attestation mechanism directly to the certificates exposed by the endpoints. This direct connection implies that certificates must be dynamically generated, which restricts them from being signed by global CAs like *Let's Encrypt*, commonly used for domain validation certificates. Opting for a different route, we used the encrypted extensions of the TLS protocol for carrying evidence of the server. Consequently, our approach is compatible with certificates endorsed by global CAs. This is particularly beneficial if the endpoint owns a separate network-level identity, like a DNS or domain name.

Comparison.   Table 2 offers a comprehensive summary of cutting-edge research dedicated to binding communication channels with attestation, focusing on features relevant to our study.

4   Design Considerations

We explain the threat model of our design, highlight its security goals and cover the trusted primitives that must be available on a system to support our proposal.

4.1   Threat Model

Our approach relies on a few key trusted components, which are essential for our system to be deemed trustworthy. We further discuss these elements in the remainder of this section.

TEEs.   Our proposal leverages the protection offered by TEEs for securing the execution of applications and enforcing strong isolation against powerful attackers, such as the OS or the hypervisor. Given that our proposal is TEE-agnostic, we highlight the minimal requirements to uphold trust in the pub/sub system and attestation mechanism. We assume the application code can be inspected but cannot be subverted. Data in use remains confidential and cannot be read unless granted by the TEE. The hardware and software strictly required to run a TEE instance are considered trusted. Furthermore, the TEE offers a remote attestation mechanism backed by genuine entities responsible for validating the trustworthiness of attestation evidence. Although we do not address side-channel or denial-of-service attacks [13, 77, 19], there exist measures for these in various TEE designs [2].

OS.   The OS follows an *honest-but-curious* threat model, posing no threat to the trusted environment but interested in gathering sensitive information. Consequently, it can monitor all communication within the pub/sub system. Assuming the OS starts to behave maliciously, the trusted computing base (TCB) remains confidential and doesn't malfunction, though it might become unresponsive. The applications running in the TEE are carefully developed to ignore abnormal responses and abandon execution in such cases.

Wasm.   We presume that the Wasm runtime is implemented correctly and does not contain vulnerabilities. The Wasm runtime acts as a shim library by encapsulating Wasm applications and uses trusted APIs from the TEE SDK for system interactions or sanitises the interaction with the untrusted OS when no secure option is available. While side-channel attacks might target Wasm and TEEs [54], they fall beyond the scope of this study.

Cryptography.   As we embed a TLS library and enhance it to integrate attestation concerns, we suppose the correct implementation of the cryptographic ciphers and operations. Additionally, we also presume that standard cryptographic techniques cannot be subverted and are hardened against side-channel attacks.

4.2   Security requirements

We propose a series of requirements for establishing trusted communication channels between the actors of pub/sub systems:

(SR1): Support global CAs certificates: brokers shall support exposing certificates issued by globally recognised CAs.

(SR2): Trust assurance: pub/sub actors shall attest the TCB of the participant they connect with and must be similarly verified in return.

(SR3): Channel and attestation bindings: communication channels must be linked to newly created attestation evidence, preventing replay or collusion attacks [48].

(SR4): Attestation privacy: attestation information shall remain confidential between the endpoints of a given communication channel.

(SR5): Pub/sub privacy: all the data bound to the pub/sub system shall be inaccessible to outside actors, including the OS, kernel and hypervisor of the broker and peers.

(SR6): Pub/sub narrow scope: the peers shall only publish or subscribe to the topics as required for their needs.

4.3   Trusted primitives

We identified a core set of trusted primitives that any system must support to host our proposal securely. These primitives are provided by the secure environment or the Wasm runtime:

- **Isolated execution context**: this ensures the runtime remains secure and unaffected by any other applications running concurrently on the system. All major TEE manufacturers support at least one Wasm runtime, though the isolation paradigm and threat model vary.
- **Attestation capabilities**: the TEE must expose two primitives to the Wasm runtimes: *generate* and *verify*. The former primitive generates evidence for proving the trustworthiness of the secure environment with additional data attached, such as a nonce and a public key, while the latter confirms the validity of this proof.
- **Network communication**: the Wasm runtimes require access to a network API that handles socket operations and the transfer of data.
- **File system access**: if the pub/sub broker needs to store publications for future delivery, the system should offer secure ways to save and access these files.

Other concerns, such as encryption and pub/sub protocol logic, are platform-independent and addressed using dedicated software compiled into Wasm.

5   Architecture

In this section, we outline the architecture and design rationale of our proposal. First, we explain the changes made to the TLS protocol. Next, we discuss the security enhancements of our pub/sub system. We wrap up with insights into specific implementation details.
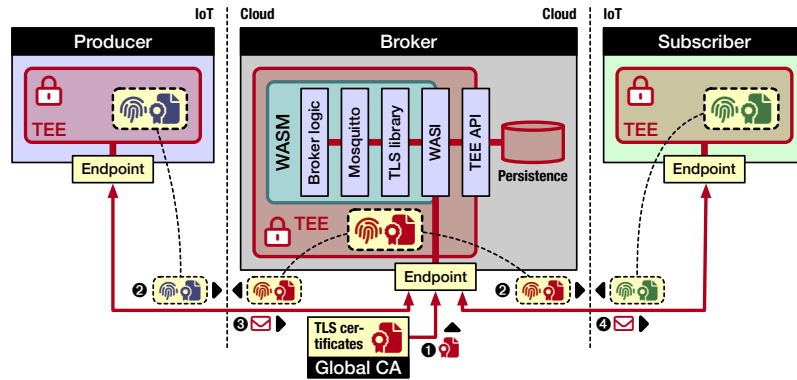
Figure 1 The overall architecture of our proposal. (⬕, ⬗) mean X.509 certificate and attestation evidence, respectively. The colours of these icons correspond to the actor owning them.

5.1 Overview

We designed our proposal as a versatile system capable of running on numerous processor architectures. The system isolates security-sensitive pub/sub operations of peers and brokers, ensuring the authenticity of connecting machines using TEEs and mutual attestation. A key aspect of our design is the adoption of Wasm, facilitating cross-platform support across various TEE architectures. More specifically, we rely on trusted Wasm runtimes [41, 42], to host a secure pub/sub system with its associated dependencies such as TLS libraries, thus covering the large spectrum of the cloud-edge continuum.

Figure 1 illustrates the key components and entities within our pub/sub solution. Serving as the central hub, the broker first acquires a certificate from a global CA for its TLS endpoint (❶). Peers then initiate secure communication with the broker via our enhanced TLS handshake, performing mutual remote attestation by exchanging their X.509 certificate for authentication and attestation evidence (detailed in Section 5.2). This ensures that the broker and peers are trustworthy (❷). The publisher generates data and transmits it to the broker's TEE (❸). The broker, in turn, relays this data to the subscriber's TEE (❹). Both the publisher and the subscriber use the same technology stack in their respective TEEs, although this detail is omitted in the figure for clarity.

5.2 Attesting communication channels

We enhanced the TLS protocol to integrate the exchange of attestation evidence when a peer is communicating with the broker. This exchange of information occurs in the TLS handshake, as depicted in Figure 2. Our enhancements are highlighted in bold.

Attestation protocol. The first message of the handshake (①) is sent by the peer, which is comprised of an encrypted attestation request (**AttReq**), indicating that the peer is establishing a TLS channel requiring attestation. Contrary to prior work, our protocol does not specify TEE architectures in this message, as our solution supports the verification of all the types of TEEs that can be used in the broker. Similarly, our protocol is the only proposal that studies the encryption of the attestation request, as further explicated below.

When received, the broker answers with a message (②) composed of its evidence (**AttServer**), freshly generated and bound to the TLS session. It is worth noting that we deliberately chose not to embed attestation data within the broker's X.509 certificate so the endpoint can use certificates endorsed by globally recognised CAs, satisfying (**SR1**).
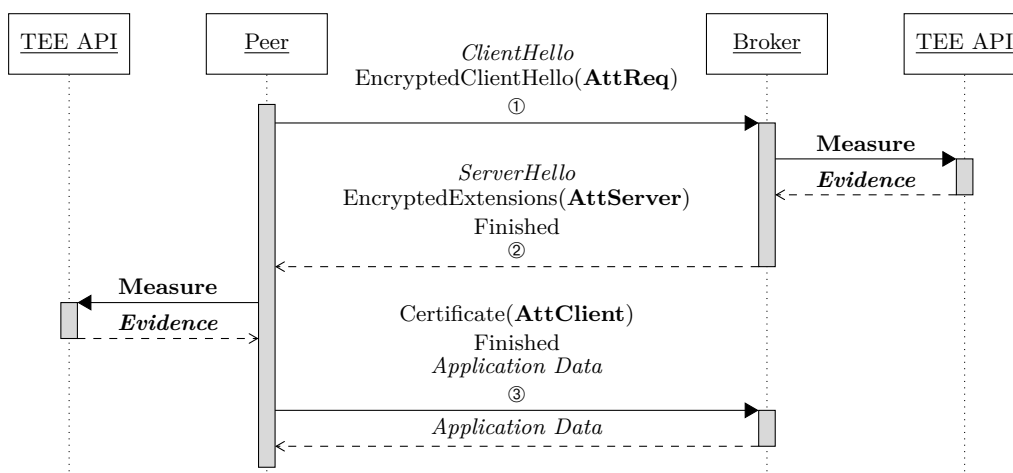
Figure 2 Enhanced TLS 1.3 handshake with attestation. New elements are mentioned in bold.

Subsequently, the peer verifies whether the evidence of the broker is genuine and checks if the evidence is bound to the current TLS session (to counter reuse attacks). Moreover, it compares the code measurement of the broker to a known reference value, indicating that the code of the broker is trusted. Since a global CA issues the broker's X.509 certificate, the peer verifies that the DNS or domain name of the broker matches the identifier exposed in the X.509 certificate [57]. If these conditions are fulfilled, the peer issues its evidence (**AttClient**) and sends it to the broker (③), also bound to the TLS session and embedded within the peer's X.509 certificate. We had to rely on a custom X.509 extension for the peer, as TLS 1.3 does not provide an extension point for the last message of the handshake.

Lastly, the broker also verifies whether the evidence of the peer is genuine, and checks if the evidence is bound to the current TLS session. Similarly, the broker ensures that the code measurement of the peer matches a known reference value, indicating that the code of the peer is trusted. Once completed, the peer and the broker mutually attest themselves, ensuring their respective TEE is trustworthy, satisfying **(SR2)**. As a result, the TLS handshake is ended, and the applications may start pub/sub communication.

Binding the handshake with attestation evidence.   Integrating the attestation mechanism in the TLS handshake has many benefits: the execution time is optimised since no additional round-trips are necessary, and the TLS 1.3 standard is respected by using extension points as designed by the protocol. Besides, we strongly bind freshly generated evidence to individual TLS sessions, using unique TLS keying materials computable by both parties (RFC 5705) [55], satisfying **(SR3)**. This prevents replay and collusion attacks, which would affect published past evidence otherwise. Since our pub/sub system and TLS library are compiled in Wasm, evidence binding within TLS handshakes is portable across different TEE architectures.

Securing the attestation information.   We use three distinct encryption mechanisms to ensure the confidentiality of the attestation information, to comply with **(SR4)**. First, we opted to use the recently-introduced *encrypted client hello* (ECH) for TLS [56], which is currently an IETF draft to communicate early information while preserving its privacy in the *ClientHello* message. To the best of our knowledge, we are the first to leverage this draft to protect the attestation request (**AttReq**) in the TLS handshake. In a nutshell, the *ClientHello* message is split into two parts: the outer message, which is in plain text, and

the inner message, which is encrypted using a public key, typically distributed using DNS infrastructure. Second, we rely on the TLS 1.3 encrypted extension in the broker reply (i.e., the `ServerHello` message), so the broker's attestation evidence remains confidential. This has been made possible because the two parties can derive a shared secret for encryption up to this point. Finally, the peer's certificate, which contains the peer's evidence, is also protected, since the entire third message of the handhake is encrypted by design.

5.3   Securing pub/sub systems

We leverage TEEs and trusted Wasm runtimes for implementing our pub/sub design, ensuring strong hardware isolation of both code and data. This design remains versatile, working with various TEE architectures as long as they offer the trusted primitives for the Wasm runtimes (as in §4.3). When paired with the mutually attested TLS protocol, we shield data in use and in transit, satisfying **(SR5)**. As a pub/sub software, we selected Mosquitto [37], a well-known and open-source message broker that implements the latest version of the MQTT protocol. We have chosen Mosquitto because it is lightweight and suitable for use on all devices, from low-power IoT devices to cloud servers. Besides, we used WolfSSL, an embeddable cryptographic library, enabling the host of the TLS termination directly in the TEE, so external systems cannot eavesdrop on the communication, nor alter the code that maintains the endpoint. We needed to compile Mosquitto in Wasm, which required slight modifications to the source code. Regarding WolfSSL, we reused one of the extension works of a trusted runtime [41], which already compiled this library in Wasm.

System interactions.   Mosquitto, like most software, is required to perform system calls for interactions with the outside world. Typically, this is the case for the socket API when exposing the TLS endpoint. For such usages, we rely on WASI, which translates the system calls to the underlying OS seamlessly. As the private and session keys of TLS are located within the TEE, the communication remains confidential against eavesdropping attempts, even when system calls are monitored. Besides, Mosquitto can persist undelivered messages in a database for later transmission to offline subscribers. The WASI specification includes a file system API for data storage, while TEEs typically provide means to save files via a trusted API securely. In contrast to the socket API, Wasm runtimes use that TEE API to transparently encrypt files, ensuring the confidentiality of the stored messages.

Securing and narrowing the pub/sub data access.   Our proposal ensures that brokers and peers are trustworthy as they are mutually attested. As such, we inherently restrict adversaries to propagate and observe messages going through the pub/sub system. Nonetheless, we propose to reduce further the scope of peers using access control lists (ACLs), which is a built-in functionality of Mosquitto. When paired with the X.509 certificates presented in the TLS handshake, this feature enables to authenticate peers without requiring additional usernames or passwords, as this is typically the case using Mosquitto. This narrowed publication and subscription scope satisfies **(SR6)**. Future work may further adapt the ACLs to be bound to evidence, provided that code measurements differ from each actor.

Use cases.   Our approach is applicable to a variety of distributed pub/sub systems, which supports scalable communication by decoupling publishers from subscribers and ensures secure messaging through mutual attestation. This was demonstrated in our European-funded project in collaboration with Siemens for secure data processing at edge nodes and capturing results on an MQTT messaging bus, as well as with the Byzantine fault-tolerant (BFT)

system of the University of Lisbon for maintaining and ensuring trust in the attestation reference values [67]. To balance the need for compactness with computational capability, we used Intel NUCs as edge nodes, which are small, efficient PCs equipped with processors that support Intel SGX. This allows the publisher on the edge nodes to verify whether the broker is genuine based on evidence and the code hash contained in that payload. This code hash is then compared to the trusted reference values stored in the BFT-resilient system. The evaluation section (§6) details how the broker performs when handling heavy loads from multiple publishers and subscribers, as encountered in these distributed environments.

5.4   Implementation

We developed a prototype of a trusted pub/sub broker using Intel SGX to better understand the practical impacts and performance overhead associated with our proposal. As such, we used TWINE [41], a trusted Wasm runtime designed to secure applications on Intel SGX. We target the latest Intel SGX technology, named *Intel Scalable*, for creating enclaves with an enclave page cache (EPC) up to 512 GiB, whereas prior work was limited to 128 MiB, which was a performance bottleneck when reaching that threshold.

Generating attestation evidence.   We provided the two trusted primitives of attestation (as in §4.3), i.e. *generate* and *verify*, which use the Intel SGX attestation capabilities. The former primitive calls the underlying TEE API to create a payload in the JSON format, which is later communicated in the TLS handshake. That payload notably contains the type of TEE and the evidence itself. The latter primitive receives a JSON and indicates whether the remote system is genuine. The runtime abstracts these primitives using *librats* [36], a library capable of generating and verifying attestation on Intel SGX, TDX and AMD SEV-SNP. For each attester type, a matching verifier type is implemented in librats, which avoids introducing TEE architecture-specific concerns in the Wasm applications. An enclave can produce a cryptographic summary of its state through the `EREPORT` instruction. This summary, or report, includes assertions to ensure the TCB is updated and secure. In addition, it also contains code measurements of the runtime and hosted Wasm application, along with the keying materials of the current TLS session. The report is then forwarded to an Intel component known as the *quoting enclave*, which fetches a report key using the `EGETKEY` instruction to authenticate the integrity of the report. Upon successful verification, the quoting enclave generates evidence, termed a *quote* in the context of SGX, allowing the other actor (e.g. the publisher, subscriber or broker) to attest the enclave [40, 58]. As a remote attestation mechanism, we use the Intel data center attestation primitives (DCAP) [60], which avoids involving Intel during the verification of the attestation evidence, speeding up the process and allows our pub/sub system to operate without an Internet connection.

Wasm.   We compiled Mosquitto using Clang with WASI-SDK [75], a toolchain that compiles C/C++ source code into Wasm for non-web environments. Mosquitto lacks support for a portable and Wasm-enabled TLS library. Therefore, we modified Mosquitto to use WolfSSL as a drop-in cryptographic library replacement. We leverage the compatibility layer of WolfSSL with OpenSSL, minimising henceforth the required code changes. Further, we disabled some of Mosquitto's system interaction layers (e.g., signals, dynamic library loading) as not supported by WASI. Turning off these features allowed a smooth embedding of Mosquitto for Wasm into the SGX enclave. The adaptation of Mosquitto required changes to 610 SLOC (in C source/header files), which accounts for 1.2% of the total codebase.

Securing the file system.   Our proposal relies on the trusted file system of Twine, which is backed by the Intel protected file system (IPFS). IPFS uses AES-GCM for encryption, taking advantage of hardware acceleration from the CPU. Files are encrypted into a Merkle tree structure of 4 KiB nodes and stored on the untrusted file system, with each node securing its children through encryption keys and tags. When the enclave requests data, IPFS decrypts tree nodes within the shielded memory of the TEE, ensuring the information stored on the untrusted file system remains confidential and unaltered. In our proof of concept, IPFS automatically generates the keys based on enclave signatures and processor-specific keys. For more sophisticated use cases, such as fault tolerance, Wasm programs can provide their encryption keys, which can be obtained from an external key management system.

Supporting the cloud-edge continuum.   We opted for Intel SGX for hosting our proof of concept because Twine supports all the trusted primitives outlined in Section 4.3, which are necessary for running our proposal. While alternative implementations are possible on other TEE architectures like Intel TDX and AMD SEV-SNP using the runtime WAMR [71], or Arm TrustZone using WaTZ [42], these options would require additional effort because they currently support only some of the trusted primitives we rely on. For example, WaTZ enhances Arm TrustZone with attestation capabilities but does not offer file persistence or a complete socket API. In the case of WAMR, its mechanisms for attestation have yet to be tested on AMD platforms. Nonetheless, we are confident that with some extensions to these runtimes, our prototype can be adapted to work with a variety of TEE architectures, enabling an agnostic approach for trustworthy pub/sub systems.

6   Evaluation

Our evaluations assess several aspects of our proof of concept. We intend to answer the following questions: *1)* What performance costs are associated with the use of Wasm, SGX, and the creation of attestation evidence on the broker during the connection process? *2)* How does our solution scale when increasing message throughput? *3)* How does our solution scale with a growing number of publishers? To answer these questions, we measure the connection times with different peers (§6.1), stress the broker with a high volume of messages (§6.2), and grow the number of publishers while observing the resulting latency (§6.3).

Scope.   Our evaluation is deliberately centred on the broker component, which is the core and critical part of distributed pub/sub architectures. As the broker orchestrates the message flow among publishers and subscribers, it largely influences the scalability of the system. Our analysis examines the broker in different variants: native execution without SGX (the baseline), Wasm outside SGX, and Wasm-SGX, running within SGX. To ensure a consistent comparison, publishers and subscribers are executed in their native environment without SGX. While our focus remains on the broker to evaluate performance benchmarks, a fully secure system requires that all entities operate within TEEs to protect the confidentiality of data, including publishers and subscribers.

Experimental setup.   The broker runs on a 16-core Intel Xeon Gold 6326 (2.9 GHz), running Ubuntu 20.04, SGX2 (EPC of 64 GiB) with SGX driver v1.41 (DCAP) and SDK v2.20. The publishers and subscribers are executed on an 8-core Intel Xeon E3-1270 v6 (3.8 GHz), running Ubuntu 20.04. These two machines are connected through a 1 Gbit/s switch. We modified Mosquitto v2.0.15 and WolfSSL v5.5.3, and compiled them using Clang 10 at
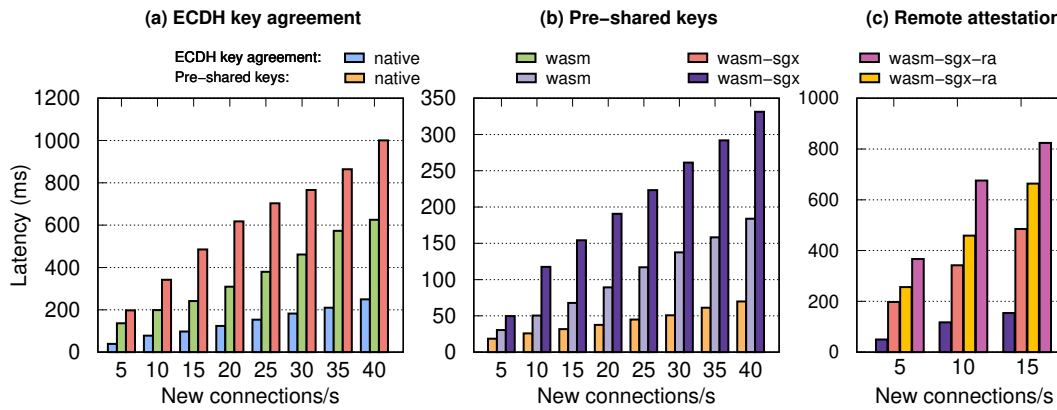
Figure 3 Latency for each new connection at varying connections per second.

maximum optimisation. For Wasm benchmarks, we initially compiled the application into Wasm bytecode with Clang, and used ahead-of-time compilation into assembly using WAMR's compiler at level size 1, as required for SGX. Time is measured using Linux's monotonic clock, including enclave exit and re-entry times ($22\,\mu s$). The MQTT broker is configured to match the minimum QoS offered, i.e. fire-and-forget mode.[1] Our implementation is open-source, and instructions to reproduce our experiments are available on GitHub [45].

6.1 Establishing new connections

Our initial evaluation measures the average latency incurred when a varying number of peers attempt to establish new connections with a broker. In this setup, we instruct a certain number of peers (depicted on the x-axis) to start a new connection every second. The y-axis displays the resulting latency. Figure 3a depicts the latency for connections made using TLS handshakes with Elliptic-curve Diffie-Hellman (ECDH) as a key agreement protocol. Figure 3b shows the latency when using a pre-shared key (PSK), effectively bypassing asymmetric cryptography. The latter approach is applicable when a peer has previously established a connection with the broker and already has a mutually agreed-upon secret key.

Across both (ECDH and PSK) scenarios, latency scales linearly for every variant as the number of peers connecting per second increases from 5 to 40. We observe higher latencies with Wasm (using ECDH) over native execution with an overhead of 2.57×, due to Wasm's performance constraints relative to native execution [30]. Additionally, Wasm-SGX (using ECDH) exhibits further overhead (4.72×) compared to native because of the enclave, which is expected given the added security benefits. A contributing source of this added overhead is the need for frequent switching between the enclave's secure and standard execution modes (`OCALLs`). This is particularly evident during operations that initialise sockets and manage peer communication in the TLS handshake. For instance, using a non-blocking socket API requires constant polling when awaiting client responses. Although such overhead is also present without SGX, it is accentuated due to the enclave transitions. We point out that the data remains confidential upon leaving the enclave, as the TLS protocol operates within a secure environment. More generally, SGX also performs slower than standard execution due to security mechanisms introduced in the microcode [68].

---

[1] MQTT supports two other operating modes, respectively delivering the message at least once with confirmation required and delivering it exactly once. Our proposal avoids the overheads of a four-step handshake while guaranteeing both the integrity and authenticity of the involved peers.
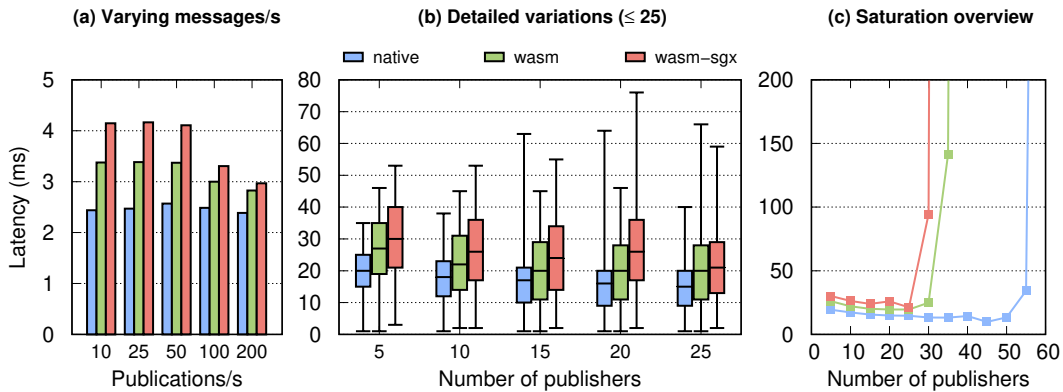
Figure 4 (a) shows latency with varying messages per second, (b) and (c) depict latency as publisher count grows.

Figure 3c explores the impact of incorporating broker attestation evidence in the TLS handshake. The overhead associated with Wasm-SGX (using ECDH) with attestation stands at $1.87\times$, whereas Wasm-SGX (using PSK) shows a $4.33\times$ overhead when compared to their counterparts without attestation. Notably, the system saturates when exceeding 15 new ECDH connections per second, while PSK performance remains stable. The added latency is primarily caused by the asymmetric operations involved in evidence generation and signing. To enhance scalability and manage this overhead over a larger set of peers, Mosquitto offers a feature called *broker bridge* that can distribute the workload across multiple brokers, although an in-depth analysis of this feature is left for future work.

6.2 Messages throughput

In Figure 4a, we evaluate the system's throughput by focusing on message delivery scalability through a specific broker. More specifically, we measure the average latency of delivering a set amount of messages per second (plotted on the x-axis) with a single publisher and subscriber. The resulting latencies are reported on the y-axis. Each test case spans 60 seconds and includes a 16 kB payload of random data.

Latency for the native variant demonstrates a slight rise at a rate of 50 messages per second, followed by a minor decline at 200 messages per second. In contrast, we note a more pronounced decrease in latency for the Wasm and Wasm-SGX variants as the rate of publications increases. To understand this behaviour, we instrumented the Wasm runtime to monitor the enclave's outgoing calls (`OCALLs`) and analysed the invoked POSIX functions. Enclave transitions typically incur performance costs, such as the intrinsic SGX transition and the copy of message buffers into secure memory. We discovered that POSIX functions related to network message transactions (i.e., `recvfrom` and `send`) were called less frequently as the messages-per-second rate increased. For instance, at a rate of 10 messages per second, the frequencies of `recvfrom` and `send` are 2601 and 1411, respectively, amounting to an average of 260 and 141 per individual message. Conversely, at a higher rate of 200 messages per second, these frequencies adjust to 49 156 and 23 498, respectively, meaning a smaller average of 246 and 117 per message. As these observations appear to be caused by the behaviour of Mosquitto's broker itself, we did not conduct additional investigations.

Overall, the Wasm and Wasm-SGX variants show performance slowdown factors of $1.30\times$ and $1.55\times$, respectively. Despite these factors, the system scales well and offers enhanced security and portability benefits.

6.3 Scaling the publishers

As our final experiment, we assess system scalability by increasing the number of publishers plotted on the x-axis, to observe the impact on message delivery latency, shown on the y-axis. The number of subscribers is held constant at 25, with each publisher sending 5 messages per second. Each test case lasts 60 seconds and uses a payload of 16 kB of random data.

Figure 4b reveals that all the variants follow a similar trend. Latency generally remains stable but decreases slightly as the number of publishers increases. The slowdown of Wasm compared to native execution is $1.31\times$, while for Wasm-SGX against native variant is $1.56\times$. Our analysis further investigates the point at which the system begins to saturate, thereby affecting its responsiveness. Figure 4c highlights this limit for each variant, indicating system saturation when exceeding 25 publishers for Wasm-SGX, followed by Wasm at 30 publishers. The native variant only reaches its limit beyond 55 publishers. Similarly to the first experiment, we could use Mosquitto's broker bridge feature to handle more publishers. This would increase the overall capacity and reduce the load of individual brokers.

7 Conclusion

Recent evolution in TEEs by leading CPU manufacturers highlights the growing trend in executing software within untrusted environments while processing increasingly sensitive data. The pub/sub model stands out as an effective mechanism to scale and distribute computations across varied architectures, and Wasm emerges as a suitable common environment for such tasks. However, a gap remains in establishing standardised protocols and tools that leverage the rapid research breakthroughs in trusted execution within the cloud-edge continuum.

Addressing this, we propose a secure and attested pub/sub system compatible with most of the state-of-the-art TEEs. We achieve this by using Wasm and trusted runtimes running in these TEEs. To incorporate mutual attestation in network communications, we suggest enhancing the TLS protocol. This modification embeds attestation evidence through extension points in the TLS handshake, preserving the original standard.

Our evaluation demonstrates that the security and portability improvements introduced by our approach effectively balance the additional overheads, mostly related to the TEEs. Moreover, leveraging Mosquitto's capability to distribute brokers among peers can further optimise the system, leading to a scalable and secure architecture suitable for large and real-world applications. Our implementation is freely distributed as an open-source project [45].

—— **References** ——

1 ISO/IEC 11889-1:2015(E). Information technology – trusted platform module library – part 1: Architecture. Standard, International Organization for Standardization, 2015.

2 A. K. M. Mubashwir Alam and Keke Chen. Making your program oblivious: a comparative study for side-channel-safe confidential computing. *CoRR*, abs/2308.06442, 2023. `doi:10.48550/ARXIV.2308.06442`.

3 Amazon. Pub/sub messaging, 2023. URL: `https://aws.amazon.com/pub-sub-messaging`.

4 AMD. AMD SEV-SNP: Strengthening VM isolation with integrity protection and more. *White Paper*, jan 2020.

5 Frederik Armknecht, Yacine Gasmi, Ahmad-Reza Sadeghi, Patrick Stewin, Martin Unger, Gianluca Ramunno, and Davide Vernizzi. An efficient implementation of trusted channels based on openssl. In Shouhuai Xu, Cristina Nita-Rotaru, and Jean-Pierre Seifert, editors, *Proceedings of the 3rd ACM Workshop on Scalable Trusted Computing, STC 2008, Alexandria, VA, USA, October 31, 2008*, pages 41–50. ACM, 2008. `doi:10.1145/1456455.1456462`.

6 Sergei Arnautov, Andrey Brito, Pascal Felber, Christof Fetzer, Franz Gregor, Robert Krahn, Wojciech Ozga, André Martin, Valerio Schiavoni, Fábio Silva, Marcus Tenorio, and Nikolaus Thummel. PubSub-SGX: Exploiting trusted execution environments for privacy-preserving publish/subscribe systems. In *37th IEEE Symposium on Reliable Distributed Systems, SRDS 2018, Salvador, Brazil, October 2-5, 2018*, pages 123–132. IEEE Computer Society, 2018. `doi:10.1109/SRDS.2018.00023`.

7 N. Asokan, Valtteri Niemi, and Kaisa Nyberg. Man-in-the-middle in tunnelled authentication protocols. In Bruce Christianson, Bruno Crispo, James A. Malcolm, and Michael Roe, editors, *Security Protocols, 11th International Workshop, Cambridge, UK, April 2-4, 2003, Revised Selected Papers*, volume 3364 of *Lecture Notes in Computer Science*, pages 28–41. Springer, 2003. `doi:10.1007/11542322_6`.

8 Pierre-Louis Aublin, Florian Kelbert, Dan O'Keffe, Divya Muthukumaran, Christian Priebe, Joshua Lind, Robert Krahn, Christof Fetzer, David Eyers, and Peter Pietzuch. TaLoS: Secure and transparent TLS termination inside SGX enclaves. Technical report, Department of Computing, Imperial College London, 2017. URL: `https://www.doc.ic.ac.uk/research/technicalreports/2017/DTRS17-5.pdf`.

9 NorazahAbd Aziz, Nur Izura Udzir, and Ramlan Mahmod. Extending TLS with mutual attestation for platform integrity assurance. *J. Commun.*, 9(1):63–72, 2014. `doi:10.12720/JCM.9.1.63-72`.

10 Raphaël Barazzutti, Pascal Felber, Hugues Mercier, Emanuel Onica, and Etienne Rivière. Efficient and confidentiality-preserving content-based publish/subscribe with prefiltering. *IEEE Trans. Dependable Secur. Comput.*, 14(3):308–325, 2017. `doi:10.1109/TDSC.2015.2449831`.

11 Stefano Berlato, Umberto Morelli, Roberto Carbone, and Silvio Ranise. End-to-end protection of IoT communications through cryptographic enforcement of access control policies. In Shamik Sural and Haibing Lu, editors, *Data and Applications Security and Privacy XXXVI – 36th Annual IFIP WG 11.3 Conference, DBSec 2022, Newark, NJ, USA, July 18-20, 2022, Proceedings*, volume 13383 of *Lecture Notes in Computer Science*, pages 236–255. Springer, 2022. `doi:10.1007/978-3-031-10684-2_14`.

12 Cristian Borcea, Arnab Deb Gupta, Yuriy Polyakov, Kurt Rohloff, and Gerard W. Ryan. PICADOR: end-to-end encrypted publish-subscribe information distribution with proxy re-encryption. *Future Gener. Comput. Syst.*, 71:177–191, 2017. `doi:10.1016/J.FUTURE.2016.10.013`.

13 Sébanjila Kevin Bukasa, Ronan Lashermes, Hélène Le Bouder, Jean-Louis Lanet, and Axel Legay. How TrustZone could be bypassed: Side-channel attacks on a modern system-on-chip. In Gerhard P. Hancke and Ernesto Damiani, editors, *Information Security Theory and Practice – 11th IFIP WG 11.2 International Conference, WISTP 2017, Heraklion, Crete, Greece, September 28-29, 2017, Proceedings*, volume 10741 of *Lecture Notes in Computer Science*, pages 93–109. Springer, 2017. `doi:10.1007/978-3-319-93524-9_6`.

14 Pau-Chen Cheng, Wojciech Ozga, Enriquillo Valdez, Salman Ahmed, Zhongshu Gu, Hani Jamjoom, Hubertus Franke, and James Bottomley. Intel TDX demystified: A top-down approach. *CoRR*, abs/2303.15540, 2023. `doi:10.48550/ARXIV.2303.15540`.

15 Victor Costan and Srinivas Devadas. Intel SGX explained. *IACR Cryptol. ePrint Arch.*, page 86, 2016. URL: `http://eprint.iacr.org/2016/086`.

16 Victor Costan, Ilia A. Lebedev, and Srinivas Devadas. Sanctum: Minimal hardware extensions for strong software isolation. In Thorsten Holz and Stefan Savage, editors, *25th USENIX Security Symposium, USENIX Security 16, Austin, TX, USA, August 10-12, 2016*, pages 857–874. USENIX Association, 2016. URL: `https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/costan`.

17 Enarx. Confidential computing with WebAssembly, 2019. URL: `https://enarx.dev`.

18 Patrick Th. Eugster, Pascal Felber, Rachid Guerraoui, and Anne-Marie Kermarrec. The many faces of publish/subscribe. *ACM Comput. Surv.*, 35(2):114–131, 2003. `doi:10.1145/857076.857078`.

19   Shufan Fei, Zheng Yan, Wenxiu Ding, and Haomeng Xie. Security vulnerabilities of SGX and countermeasures: A survey. *ACM Comput. Surv.*, 54(6):126:1–126:36, 2022. `doi:10.1145/3456631`.

20   Andrew Ferraiuolo, Andrew Baumann, Chris Hawblitzel, and Bryan Parno. Komodo: Using verification to disentangle secure-enclave hardware from software. In *Proceedings of the 26th Symposium on Operating Systems Principles, Shanghai, China, October 28-31, 2017*, pages 287–305. ACM, 2017. `doi:10.1145/3132747.3132782`.

21   Sarah Abdelwahab Gaballah, Christoph Coijanovic, Thorsten Strufe, and Max Mühlhäuser. 2PPS – publish/subscribe with provable privacy. In *40th International Symposium on Reliable Distributed Systems, SRDS 2021, Chicago, IL, USA, September 20-23, 2021*, pages 198–209. IEEE, 2021. `doi:10.1109/SRDS53918.2021.00028`.

22   Yacine Gasmi, Ahmad-Reza Sadeghi, Patrick Stewin, Martin Unger, and N. Asokan. Beyond secure channels. In Peng Ning, Vijay Atluri, Shouhuai Xu, and Moti Yung, editors, *Proceedings of the 2nd ACM Workshop on Scalable Trusted Computing, STC 2007, Alexandria, VA, USA, November 2, 2007*, pages 30–40. ACM, 2007. `doi:10.1145/1314354.1314363`.

23   Kenneth A. Goldman, Ronald Perez, and Reiner Sailer. Linking remote attestation to secure tunnel endpoints. In Ari Juels, Gene Tsudik, Shouhuai Xu, and Moti Yung, editors, *Proceedings of the 1st ACM Workshop on Scalable Trusted Computing, STC 2006, Alexandria, VA, USA, November 3, 2006*, pages 21–24. ACM, 2006. `doi:10.1145/1179474.1179481`.

24   Google. Pub/sub, 2023. URL: `https://cloud.google.com/pubsub`.

25   Christian Göttel, Pascal Felber, and Valerio Schiavoni. Developing secure services for IoT with OP-TEE: A first look at performance and usability. In José Pereira and Laura Ricci, editors, *Distributed Applications and Interoperable Systems – 19th IFIP WG 6.1 International Conference, DAIS 2019, Held as Part of the 14th International Federated Conference on Distributed Computing Techniques, DisCoTec 2019, Kongens Lyngby, Denmark, June 17-21, 2019, Proceedings*, volume 11534 of *Lecture Notes in Computer Science*, pages 170–178. Springer, 2019. `doi:10.1007/978-3-030-22496-7_11`.

26   Franz Gregor, Wojciech Ozga, Sébastien Vaucher, Rafael Pires, Do Le Quoc, Sergei Arnautov, André Martin, Valerio Schiavoni, Pascal Felber, and Christof Fetzer. Trust management as a service: Enabling trusted execution in the face of byzantine stakeholders. In *50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, DSN 2020, Valencia, Spain, June 29 – July 2, 2020*, pages 502–514. IEEE, 2020. `doi:10.1109/DSN48063.2020.00063`.

27   Andreas Haas, Andreas Rossberg, Derek L. Schuff, Ben L. Titzer, Michael Holman, Dan Gohman, Luke Wagner, Alon Zakai, and J. F. Bastien. Bringing the web up to speed with WebAssembly. In Albert Cohen and Martin T. Vechev, editors, *Proceedings of the 38th ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2017, Barcelona, Spain, June 18-23, 2017*, pages 185–200. ACM, 2017. `doi:10.1145/3062341.3062363`.

28   Intel. Intel software guard extensions remote attestation end-to-end example, jul 2018. URL: `https://software.intel.com/content/www/us/en/develop/articles/code-sample-intel-software-guard-extensions-remote-attestation-end-to-end-example.html`.

29   Mihaela Ion, Giovanni Russello, and Bruno Crispo. Design and implementation of a confidentiality and access control solution for publish/subscribe systems. *Comput. Networks*, 56(7):2014–2037, 2012. `doi:10.1016/J.COMNET.2012.02.013`.

30   Abhinav Jangda, Bobby Powers, Emery D. Berger, and Arjun Guha. Not so fast: Analyzing the performance of WebAssembly vs. native code. In Dahlia Malkhi and Dan Tsafrir, editors, *2019 USENIX Annual Technical Conference, USENIX ATC 2019, Renton, WA, USA, July 10-12, 2019*, pages 107–120. USENIX Association, 2019. URL: `https://www.usenix.org/conference/atc19/presentation/jangda`.

31   André Joaquim, Miguel L. Pardal, and Miguel Correia. Vulnerability-tolerant transport layer security. In James Aspnes, Alysson Bessani, Pascal Felber, and João Leitão, editors, *21st International Conference on Principles of Distributed Systems, OPODIS 2017, Lisbon, Portugal, December 18-20, 2017*, volume 95 of *LIPIcs*, pages 28:1–28:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2017. `doi:10.4230/LIPICS.OPODIS.2017.28`.

32   Simon Johnson, Vinnie Scarlata, Carlos Rozas, Ernie Brickell, and Frank Mckeen. Intel software guard extensions: EPID provisioning and attestation services. *White Paper*, 1(1-10):119, 2016. URL: `https://cdrdv2-public.intel.com/671370/ww10-2016-sgx-provisioning-and-attestation-final.pdf`.

33   Thomas Knauth, Michael Steiner, Somnath Chakrabarti, Li Lei, Cedric Xing, and Mona Vij. Integrating remote attestation with transport layer security. *CoRR*, abs/1801.05863, 2018. `arXiv:1801.05863`.

34   Hugo Krawczyk. SIGMA: The "SIGn-and-MAc" approach to authenticated Diffie-Hellman and its use in the IKE-protocols. In Dan Boneh, editor, *Advances in Cryptology – CRYPTO 2003, 23rd Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 2003, Proceedings*, volume 2729 of *Lecture Notes in Computer Science*, pages 400–425. Springer, 2003. `doi:10.1007/978-3-540-45146-4_24`.

35   Dayeol Lee, David Kohlbrenner, Shweta Shinde, Krste Asanovic, and Dawn Song. Keystone: an open framework for architecting trusted execution environments. In Angelos Bilas, Kostas Magoutis, Evangelos P. Markatos, Dejan Kostic, and Margo I. Seltzer, editors, *EuroSys '20: Fifteenth EuroSys Conference 2020, Heraklion, Greece, April 27-30, 2020*, pages 38:1–38:16. ACM, 2020. `doi:10.1145/3342195.3387532`.

36   Librats. Low level attester and verifier drivers for multiple TEEs, 2022. URL: `https://github.com/inclavare-containers/librats`.

37   Roger A. Light. Mosquitto: server and client implementation of the MQTT protocol. *J. Open Source Softw.*, 2(13):265, 2017. `doi:10.21105/JOSS.00265`.

38   Lukas Malina, Gautam Srivastava, Petr Dzurenda, Jan Hajny, and Radek Fujdiak. A secure publish/subscribe protocol for internet of things. In *Proceedings of the 14th International Conference on Availability, Reliability and Security, ARES 2019, Canterbury, UK, August 26-29, 2019*, pages 75:1–75:10. ACM, 2019. `doi:10.1145/3339252.3340503`.

39   Jonathan M. McCune, Yanlin Li, Ning Qu, Zongwei Zhou, Anupam Datta, Virgil D. Gligor, and Adrian Perrig. Trustvisor: Efficient TCB reduction and attestation. In *31st IEEE Symposium on Security and Privacy, S&P 2010, 16-19 May 2010, Berleley/Oakland, California, USA*, pages 143–158. IEEE Computer Society, 2010. `doi:10.1109/SP.2010.17`.

40   Jämes Ménétrey, Christian Göttel, Anum Khurshid, Marcelo Pasin, Pascal Felber, Valerio Schiavoni, and Shahid Raza. Attestation mechanisms for trusted execution environments demystified. In David M. Eyers and Spyros Voulgaris, editors, *Distributed Applications and Interoperable Systems: 22nd IFIP WG 6.1 International Conference, DAIS 2022, Held as Part of the 17th International Federated Conference on Distributed Computing Techniques, DisCoTec 2022, Lucca, Italy, June 13-17, 2022, Proceedings*, volume 13272 of *Lecture Notes in Computer Science*, pages 95–113. Springer, 2022. `doi:10.1007/978-3-031-16092-9_7`.

41   Jämes Ménétrey, Marcelo Pasin, Pascal Felber, and Valerio Schiavoni. TWINE: An embedded trusted runtime for WebAssembly. In *37th IEEE International Conference on Data Engineering, ICDE 2021, Chania, Greece, April 19-22, 2021*, pages 205–216. IEEE, 2021. `doi:10.1109/ICDE51399.2021.00025`.

42   Jämes Ménétrey, Marcelo Pasin, Pascal Felber, and Valerio Schiavoni. WATZ: A trusted WebAssembly runtime environment with remote attestation for TrustZone. In *42nd IEEE International Conference on Distributed Computing Systems, ICDCS 2022, Bologna, Italy, July 10-13, 2022*, pages 1177–1189. IEEE, 2022. `doi:10.1109/ICDCS54860.2022.00116`.

43   Microsoft. Publisher-subscriber pattern, 2023. URL: `https://learn.microsoft.com/en-us/azure/architecture/patterns/publisher-subscriber`.

44   Mozilla. Standardizing WASI: A system interface to run WebAssembly outside the web, mar 2019.   URL: `https://hacks.mozilla.org/2019/03/standardizing-wasi-a-webassembly-system-interface/`.

45   Jämes Ménétrey. A holistic approach for trustworthy distributed systems with WebAssembly and TEEs: code and benchmarks, 2023. URL: `https://github.com/JamesMenetrey/unine-opodis2023`.

46   Mohamed Nabeel, Stefan Appel, Elisa Bertino, and Alejandro P. Buchmann. Privacy preserving context aware publish subscribe systems. In Javier López, Xinyi Huang, and Ravi S. Sandhu, editors, *Network and System Security – 7th International Conference, NSS 2013, Madrid, Spain, June 3-4, 2013. Proceedings*, volume 7873 of *Lecture Notes in Computer Science*, pages 465–478. Springer, 2013. `doi:10.1007/978-3-642-38631-2_34`.

47   Tu Dinh Ngoc, Bao Bui, Stella Bitchebe, Alain Tchana, Valerio Schiavoni, Pascal Felber, and Daniel Hagimont. Everything you should know about intel SGX performance on virtualized systems. *Proc. ACM Meas. Anal. Comput. Syst.*, 3(1):5:1–5:21, 2019. `doi:10.1145/3322205.3311076`.

48   Arto Niemi, Vasile Adrian Bogdan Pop, and Jan-Erik Ekberg. Trusted sockets layer: A TLS 1.3 based trusted channel protocol. In Nicola Tuveri, Antonis Michalas, and Billy Bob Brumley, editors, *Secure IT Systems – 26th Nordic Conference, NordSec 2021, Virtual Event, November 29-30, 2021, Proceedings*, volume 13115 of *Lecture Notes in Computer Science*, pages 175–191. Springer, 2021. `doi:10.1007/978-3-030-91625-1_10`.

49   Arto Niemi, Sampo Sovio, and Jan-Erik Ekberg. Towards interoperable enclave attestation: Learnings from decades of academic work. In *31st Conference of Open Innovations Association, FRUCT 2022, Helsinki, Finland, April 27-29, 2022*, pages 189–200. IEEE, 2022. `doi:10.23919/FRUCT54823.2022.9770907`.

50   Emanuel Onica, Pascal Felber, Hugues Mercier, and Etienne Rivière. Confidentiality-preserving publish/subscribe: A survey. *ACM Comput. Surv.*, 49(2):27:1–27:43, 2016. `doi:10.1145/2940296`.

51   A Paverd. *Enhancing communication privacy using trustworthy remote entities*. PhD thesis, University of Oxford, 2015.

52   Jinglei Pei, Yuyang Shi, Qingling Feng, Ruisheng Shi, Lina Lan, Shui Yu, Jinqiao Shi, and Zhaofeng Ma. An efficient confidentiality protection solution for pub/sub system. *Cybersecur.*, 6(1):34, 2023. `doi:10.1186/S42400-023-00165-W`.

53   Rafael Pires, Marcelo Pasin, Pascal Felber, and Christof Fetzer. Secure content-based routing using Intel software guard extensions. In *Proceedings of the 17th International Middleware Conference, Trento, Italy, December 12 – 16, 2016*, page 10. ACM, 2016. `doi:10.1145/2988336.2988346`.

54   Ivan Puddu, Moritz Schneider, Daniele Lain, Stefano Boschetto, and Srdjan Capkun. On (the lack of) code confidentiality in trusted execution environments. *CoRR*, abs/2212.07899, 2022. `arXiv:2212.07899`, `doi:10.48550/ARXIV.2212.07899`.

55   Eric Rescorla. Keying material exporters for transport layer security (TLS). RFC 5705, mar 2010. `doi:10.17487/RFC5705`.

56   Eric Rescorla, Kazuho Oku, Nick Sullivan, and Christopher A. Wood. TLS encrypted client hello. Internet-Draft draft-ietf-tls-esni-16, Internet Engineering Task Force, apr 2023. Work in Progress. URL: `https://datatracker.ietf.org/doc/draft-ietf-tls-esni/16/`.

57   Peter Saint-Andre and Jeff Hodges. Representation and verification of domain-based application service identity within internet public key infrastructure using X.509 (PKIX) certificates in the context of transport layer security (TLS). RFC 6125, mar 2011. `doi:10.17487/RFC6125`.

58   Muhammad Usama Sardar, Rasha Faqeh, and Christof Fetzer. Formal foundations for Intel SGX data center attestation primitives. In Shang-Wei Lin, Zhe Hou, and Brendan P. Mahony, editors, *Formal Methods and Software Engineering – 22nd International Conference on Formal Engineering Methods, ICFEM 2020, Singapore, Singapore, March 1-3, 2021, Proceedings*, volume 12531 of *Lecture Notes in Computer Science*, pages 268–283. Springer, 2020. `doi:10.1007/978-3-030-63406-3_16`.

59   Muhammad Usama Sardar, Saidgani Musaev, and Christof Fetzer. Demystifying attestation in Intel trust domain extensions via formal verification. *IEEE Access*, 9:83067–83079, 2021. `doi:10.1109/ACCESS.2021.3087421`.

60   Vinnie Scarlata, Simon Johnson, James Beaney, and Piotr Zmijewski. Supporting third party attestation for Intel SGX with Intel data center attestation primitives. *White paper*, page 12, 2018. URL: `https://cdrdv2-public.intel.com/671314/intel-sgx-support-for-third-party-attestation.pdf`.

61   Carlos Segarra, Ricard Delgado-Gonzalo, and Valerio Schiavoni. MQT-TZ: Hardening IoT brokers using ARM TrustZone : (practical experience report). In *International Symposium on Reliable Distributed Systems, SRDS 2020, Shanghai, China, September 21-24, 2020*, pages 256–265. IEEE, 2020. `doi:10.1109/SRDS51746.2020.00033`.

62   Carlton Shepherd, Raja Naeem Akram, and Konstantinos Markantonakis. Establishing mutually trusted channels for remote sensing devices with trusted execution environments. In *Proceedings of the 12th International Conference on Availability, Reliability and Security, Reggio Calabria, Italy, August 29 – September 01, 2017*, pages 7:1–7:10. ACM, 2017. `doi:10.1145/3098954.3098971`.

63   Frederic Stumpf, Andreas Fuchs, Stefan Katzenbeisser, and Claudia Eckert. Improving the scalability of platform attestation. In Shouhuai Xu, Cristina Nita-Rotaru, and Jean-Pierre Seifert, editors, *Proceedings of the 3rd ACM Workshop on Scalable Trusted Computing, STC 2008, Alexandria, VA, USA, October 31, 2008*, pages 1–10. ACM, 2008. `doi:10.1145/1456455.1456457`.

64   Frederic Stumpf, Omid Tafreschi, Patrick Röder, Claudia Eckert, et al. A robust integrity reporting protocol for remote attestation. In *Proceedings of the Workshop on Advances in Trusted Computing (WATC)*, page 65, 2006.

65   Antero Taivalsaari, Tommi Mikkonen, and Cesare Pautasso. Towards seamless IoT device-edge-cloud continuum: Software architecture options of IoT devices revisited. In Maxim Bakaev, In-Young Ko, Michael Mrissa, Cesare Pautasso, and Abhishek Srivastava, editors, *ICWE 2021 Workshops – ICWE 2021 International Workshops, BECS and Invited Papers, Biarritz, France, May 18-21, 2021*, volume 1508 of *Communications in Computer and Information Science*, pages 82–98. Springer, 2021. `doi:10.1007/978-3-030-92231-3_8`.

66   Trusted Computing Group. *DICE Attestation Architecture*, 2021. URL: `https://trustedcomputinggroup.org/wp-content/uploads/DICE-Attestation-Architecture-r23-final.pdf`.

67   Robin Vassantlal, Eduardo Alchieri, Bernardo Ferreira, and Alysson Bessani. COBRA: dynamic proactive secret sharing for confidential BFT services. In *43rd IEEE Symposium on Security and Privacy, SP 2022, San Francisco, CA, USA, May 22-26, 2022*, pages 1335–1353. IEEE, 2022. `doi:10.1109/SP46214.2022.9833658`.

68   Sébastien Vaucher, Valerio Schiavoni, and Pascal Felber. Short paper: Stress-SGX: Load and stress your enclaves for fun and profit. In Andreas Podelski and François Taïani, editors, *Networked Systems – 6th International Conference, NETYS 2018, Essaouira, Morocco, May 9-11, 2018, Revised Selected Papers*, volume 11028 of *Lecture Notes in Computer Science*, pages 358–363. Springer, 2018. `doi:10.1007/978-3-030-05529-5_24`.

69   Paul Georg Wagner, Pascal Birnstill, and Jürgen Beyerer. Establishing secure communication channels using remote attestation with TPM 2.0. In Konstantinos Markantonakis and Marinella Petrocchi, editors, *Security and Trust Management – 16th International Workshop, STM 2020, Guildford, UK, September 17-18, 2020, Proceedings*, volume 12386 of *Lecture Notes in Computer Science*, pages 73–89. Springer, 2020. `doi:10.1007/978-3-030-59817-4_5`.

70   Kevin Walsh and John Manferdelli. Mechanisms for mutual attested microservice communication. In Ashiq Anjum, Alan Sill, Geoffrey C. Fox, and Yong Chen, editors, *Companion Proceedings of the 10th International Conference on Utility and Cloud Computing, UCC 2017, Austin, TX, USA, December 5-8, 2017*, pages 59–64. ACM, 2017. `doi:10.1145/3147234.3148102`.

71 WAMR. WebAssembly micro runtime, 2019. URL: `https://github.com/bytecodealliance/wasm-micro-runtime`.

72 Chenxi Wang, Antonio Carzaniga, David Evans, and Alexander L Wolf. Security issues and requirements for internet-scale publish-subscribe systems. In *Proceedings of the 35th Annual Hawaii International Conference on System Sciences*, pages 3940–3947. IEEE, 2002. `doi:10.1109/HICSS.2002.994531`.

73 Shuran Wang, Dahan Pan, Runhan Feng, and Yuanyuan Zhang. Magikcube: Securing cross-domain publish/subscribe systems with enclave. In *20th IEEE International Conference on Trust, Security and Privacy in Computing and Communications, Trust-Com 2021, Shenyang, China, October 20-22, 2021*, pages 147–154. IEEE, 2021. `doi:10.1109/TRUSTCOM53373.2021.00037`.

74 Wenwen Wang. How far we've come – A characterization study of standalone WebAssembly runtimes. In *IEEE International Symposium on Workload Characterization, IISWC 2022, Austin, TX, USA, November 6-8, 2022*, pages 228–241. IEEE, 2022. `doi:10.1109/IISWC55918.2022.00028`.

75 WASI-SDK. WASI-enabled WebAssembly C/C++ toolchain, 2019. URL: `https://github.com/WebAssembly/wasi-sdk`.

76 Samuel Weiser, Mario Werner, Ferdinand Brasser, Maja Malenko, Stefan Mangard, and Ahmad-Reza Sadeghi. TIMBER-V: Tag-isolated memory bringing fine-grained enclaves to RISC-V. In *26th Annual Network and Distributed System Security Symposium, NDSS 2019, San Diego, California, USA, February 24-27, 2019*. The Internet Society, 2019. `doi:10.14722/ndss.2019.23068`.

77 Jan Werner, Joshua Mason, Manos Antonakakis, Michalis Polychronakis, and Fabian Monrose. The severest of them all: Inference attacks against secure virtual enclaves. In Steven D. Galbraith, Giovanni Russello, Willy Susilo, Dieter Gollmann, Engin Kirda, and Zhenkai Liang, editors, *Proceedings of the 2019 ACM Asia Conference on Computer and Communications Security, AsiaCCS 2019, Auckland, New Zealand, July 09-12, 2019*, pages 73–85. ACM, 2019. `doi:10.1145/3321705.3329820`.

78 Yue Yu, Huaimin Wang, Bo Liu, and Gang Yin. A trusted remote attestation model based on trusted computing. In *12th IEEE International Conference on Trust, Security and Privacy in Computing and Communications, TrustCom 2013 / 11th IEEE International Symposium on Parallel and Distributed Processing with Applications, ISPA-13 / 12th IEEE International Conference on Ubiquitous Computing and Communications, IUCC-2013, Melbourne, Australia, July 16-18, 2013*, pages 1504–1509. IEEE Computer Society, 2013. `doi:10.1109/TRUSTCOM.2013.183`.