Journal of the Brazilian Computer Society, 2026, 32:1, doi: 10.5753/jbcs.2026.5556
This work is licensed under a Creative Commons Attribution 4.0 International License.

Redefining Digital Web Signature Secrecy: A Client-Side Model for Enhanced Security and Compliance

Wellington Fernandes Silvano [**Federal University of Santa Catarina** | ***wellington.fernandes@posgrad.ufsc.br***]
Lucas Mayr [**Federal University of Santa Catarina** | ***lucas.mayr@posgrad.ufsc.br***]
Enzo Brum [**Federal University of Santa Catarina** | ***enzo.brum@grad.ufsc.br***]
Gabriel Cabral [**Federal University of Santa Catarina** | ***gabriel.aristeu.cabral@gmail.com***]
Frederico Schardong [**Federal Institute of Rio Grande do Sul** | ***frederico.schardong@rolante.ifrs.edu.br***]
Ricardo Custódio [**Federal University of Santa Catarina** | ***ricardo.custodio@ufsc.br***]

LabSEC | Federal University of Santa Catarina , R. Eng. Agronômico Andrei Cristian Ferreira, s/n - Trindade, Florianópolis, SC, 88040-900, Brazil.

Received: 15 February 2025 • **Accepted:** 21 November 2025 • **Published:** 22 April 2026

Abstract Web-based digital signature platforms prioritize convenience but often compromise secrecy, exposing sensitive documents and private keys to third-party systems. This paper introduces a client-side cryptographic model that eliminates such vulnerabilities by performing all cryptographic operations within the user's browser. Leveraging One-Time Certificates and adhering to Claude Shannon's secrecy principles, the model ensures that documents and keys remain secure by never leaving the client environment. The proposed approach addresses critical risks, including document exposure, metadata leakage, and key compromise, while maintaining compatibility with public key infrastructure standards and legacy systems. Performance evaluations show efficient signing and verification processes, with documents up to 5 MB signed in approximately 1 second and verified in 0.15 seconds. By removing reliance on external servers for sensitive operations, the model mitigates platform vulnerabilities, reduces liability, and ensures compliance with regulations like GDPR and LGPD. Key contributions include enhanced secrecy, simplified key management, and scalable real-world use-case performance. This work redefines digital signature security, offering a robust, privacy-preserving alternative for secrecy document signature and verification workflows.

Keywords: Digital signature, secrecy, Web signature, client-side signature

1 Introduction

Web-based digital signature platforms have revolutionized modern workflows, offering unparalleled convenience. These tools enable individuals, professionals, and organizations to sign documents with ease and efficiency. However, in prioritizing convenience over secrecy, many platforms require users to upload sensitive documents to external servers and rely on outsourced key management services. This approach exposes critical information-such as professional records (e.g., legal, medical, or notarial documents), classified governmental policies, corporate strategies, and personal data to significant risks. These include unauthorized document access, metadata leakage, and cryptographic key compromise, particularly when analyzed through the lens of Shannon's seminal work on secrecy principles [Shannon, 1949].

Beyond individual concerns about confidentiality, platform operators face significant liabilities such as data leaks, which can lead to penalties under regulations such as the EU GDPR [European Union, 2018] and Brazil's LGPD [Brasil, 2018]. Additionally, platforms must navigate a complex legal landscape that includes professional secrecy obligations [OAB, 2015; CFM, 2010], espionage laws [United States, 1917; United Kingdom, 1989], guarantees of correspondence confidentiality [Brazil, 1996], and regulations concerning classified governmental information [United Kingdom, 2000; Brazil, 2011]. These liabilities arise from the interplay between inadequate secrecy mechanisms and strict regulatory requirements. At the same time, the emergence of modern security paradigms, such as Zero Trust [Stafford, 2020; NIST, 2020], underscores the importance of minimizing implicit trust by rigorously verifying all entities, including systems. This principle is especially relevant when addressing the flaws of existing web-based solutions that centralize critical operations.

Claude Shannon in his seminal work called *Communication Theory of Secrecy Systems*, published in 1949, laid the foundation for modern cryptography. Shannon presented three secrecy types: **SST-1**, hiding the existence of the message; **SST-2**, requiring specialized techniques to retrieve it; and **SST-3**, obscuring message content through ciphers [Shannon, 1949]. Applied to digital signature platforms, these principles reveal vulnerabilities such as document exposure to unauthorized entities (SST-1), metadata leakage (SST-2), and potential access to encrypted content (SST-3). Addressing these challenges requires a comprehensive approach that integrates Shannon's secrecy definitions with modern paradigms like Zero Trust to ensure usability and compliance with legal requirements.

As previously mentioned, existing solutions in the signa-

ture and document secrecy literature attempt to address these issues but face several limitations. Offline systems [Luan *et al.*, 2015] offer strong security in isolated environments but compromise usability and interoperability. Document tracing frameworks [Choi *et al.*, 2017] detect unauthorized modifications and use steganographic marks to prevent leaks, but cannot prevent intentional data exfiltration by malicious insiders. Embedded certificates [Shatnawi *et al.*, 2017] enhance document self-sufficiency but face interoperability issues with traditional verification systems. Blockchain-based frameworks [Aciobăniei *et al.*, 2024] ensure traceability but suffer from scalability challenges and complications with long-term certificate validity. One-Time Certificates (OTCs) [Mayr *et al.*, 2023, 2024] simplify key management but fail to fully address document secrecy requirements.

To address these challenges, this paper introduces a novel client-side cryptographic model for web-based digital signature platforms. The proposed model mitigates the main limitations of prior approaches by combining the isolation guarantees of offline systems with convenience, and the interoperability of online platforms. All cryptographic keys are generated locally within the user's browser, removing the need to trust third-party platforms for key creation or storage. Client-side creation of cryptographic key, signing, and verification ensures that neither documents nor private keys ever leave the user's environment, eliminating document exposure and metadata leakage. By embedding the document hash in the One-Time Certificate (OTC), on client-side, request, each certificate is cryptographically bound to a single document, preventing long-term key compromise. This architecture achieves high secrecy and legal compliance, rooted in Shannon's secrecy principles and aligned with the Zero Trust paradigm, the proposed model enhances document confidentiality and simplifies key management through the innovative use of OTCs. The model demonstrates that heightened secrecy can be obtained without compromising compatibility or interoperability and adhering to relevant Public Key Infrastructure (PKI) standards.

Furthermore, this work addresses critical challenges such as optimizing client-side performance, ensuring scalability for real-world applications, and integrating seamlessly with legacy systems. By mitigating risks such as document leakage and cryptographic key compromise, the proposed model advances document secrecy while providing a methodology for evaluating secrecy in web-based signature systems.

This paper explores the proposed model in depth, offering a practical implementation and analysis. Section 2 reviews related work on digital signature platforms and document secrecy. Section 3 provides the theoretical foundation, while Section 4 examines limitations in current platforms. Section 5 introduces the One-Time Certificate mechanism, with implementation details and security implications discussed in Section 6. Finally, results and future directions are presented in Sections 7 and 8.

2 Related Work

This section provides an overview of existing research on safeguarding the integrity of sensitive documents using digital signatures. We examine studies that explore alternative key management strategies to traditional PKIs.

[Luan *et al.*, 2015] highlight the complexities of managing confidential documents in isolated environments where internet-based PKI systems are unfeasible. Their model uses an offline server for signing and encrypting documents, requiring physical document transfer via USB drives to ensure confidentiality. This approach highlights the unique challenges of securing information without internet access and effectively mitigates risks tied to online PKI systems.

[Choi *et al.*, 2017] underscore the growing challenges of securing confidential documents in cloud environments. Their Doc-Trace framework addresses this issue by implementing document tracing at the hypervisor level, a software layer below the virtual machines' operating systems. Steganographic marks embedded within documents enable the detection of content modifications and the logging of access events. However, the solution's effectiveness is limited to identifying unauthorized alterations, as it cannot prevent intentional data exfiltration by malicious insiders. Doc-Trace is thus valuable for maintaining comprehensive audit trails but falls short as a standalone defense against deliberate data breaches. This highlights the ongoing need for multifaceted security approaches to safeguard sensitive information in cloud environments.

[Shatnawi *et al.*, 2017] address the critical challenge of maintaining integrity and non-repudiation for sensitive offline documents, including medical records, legal reports, and financial assets. Their framework embeds digital certificates directly within documents, enabling detailed tracking of modifications, including author, device, and timestamp. Integrated as a Microsoft Word plugin, this approach offers self-sufficiency in secure environments. However, the reliance on embedded certificates may hinder compatibility with traditional verification systems expecting externally accessible certificates. While adhering to X.509 and RFC 5280 certificate standards [Boeyen *et al.*, 2008], the solution faces limitations in revocation verification and interoperability. These factors highlight the need for a balanced approach to document security, considering both robust protection and practical usability across diverse operational contexts, especially for frequently modified, highly sensitive documents.

[Aciobăniei *et al.*, 2024] address the critical need for secure and private remote qualified electronic signatures (RQES) for confidential documents. Their proposed framework leverages a Trust Service Provider (TSP) to digitally sign document hashes, preserving content confidentiality. A browser plugin facilitates hash generation, user authentication, and secure hash transmission to the TSP. The resulting digital signature and TSP certificate are embedded within the document. To ensure long-term integrity and validity, digital signatures, certificates, and user metadata are registered on the Ethereum blockchain [Foundation, 2024]. While this approach protects content privacy and aligns with regulations like eIDAS [European Union, 2014], it introduces challenges. The document itself does not inherently identify the signer, requiring external consultation. Additionally, potential conflicts may arise between the certificate's temporal validity and the blockchain's long-term data storage. Balancing security, privacy, and practical considerations remains essential for

effective online document management.

[Mayr *et al.*, 2024] introduce a novel key management approach as an alternative to traditional PKIs. Their core innovation is the One-Time Certificate, where signature verification by a certificate is restricted to a single document through an embedded cryptographic hash of the target document inside the certificate. This model generates a key pair for each signed document, thus requiring the issuance of a new OTC for each new signature. After the certificate is issued and the document is signed, the private key of the key pair is deleted. The document hash is incorporated into an X.509 extension before submission to the Certificate Authority (CA) for issuance. Embedding the hash within the certificate ensures its immutability and directly links the certificate to the specific document. A key advantage of this approach is the removal of ongoing certificate and key management burdens for end-users, reducing costs and technical complexities. Unlike traditional methods requiring persistent certificate and key pair maintenance, OTCs mandate the creation of a new key pair and certificate for each document, enhancing security and simplifying the signing process.

3 Background

This section provides foundational knowledge of digital signatures to support reader comprehension.

3.1 Digital Signatures

Digital signatures are cryptographic mechanisms that assert the integrity and authenticity of digital data. Functioning as electronic counterparts to handwritten signatures, they safeguard against tampering and impersonation. By employing asymmetric cryptography, digital signatures establish proof of origin, identity, and document status. The signing process involves a private key held by the signer to generate a signature, which can then be verified by anyone using its public key counterpart [Goldreich, 2001].

The key generation algorithm produces an asymmetric key pair consisting of a private key sk and a public key pk, formally expressed as $\text{Gen}(1^\lambda) \to (sk, pk)$, where λ is the security parameter determining the computational complexity of the algorithms.

Typically, the message m is first run through a cryptographic hashing algorithm, producing a fixed-length digest h. Then, the private key sk is used to compute a digital signature σ using a digital signature scheme such that $\text{Sign}(sk, h) \to \sigma$.

Lastly, signature verification checks whether a given public key pk truly is the counterpart to the private key sk used in the signature process using the message m and signature σ as part of the comparison process, i.e., $\text{Verify}(pk, m, \sigma) \to \{0, 1\}$; where a result of 1 indicates a valid signature, and 0 indicates an invalid one.

3.2 Digital Certificate

Digital certificates, typically conforming to the X.509 standard, are electronic credentials used to bind an entity's identity to a public key. This binding is cryptographically asserted by a trusted third party known as a Certificate Authority (CA). The issued certificate, along with its certification path, can then be traced back to a trusted root CA, ensuring its legitimacy.

To obtain such a digital certificate, an entity must generate a Certificate Signing Request (CSR) and submit it to a CA for issuance. Created after generating the key pair, the CSR includes the public key and essential metadata needed for the certificate. Particularly, the CSR is authenticated using the entity's private key to verify its legitimacy. Two primary formats are commonly used to handle digital certificates and related cryptographic information: PKCS#7 [Nystrom and Kaliski, 2000] for structures like signed data and certificate collections, and PKCS#12 [Moriarty *et al.*, 2014] for storing a private key along with its associated certificate chain.

3.3 Zero Trust

Zero Trust represents an evolving set of cybersecurity paradigms that shift defenses from static, network-based perimeters to a dynamic focus on users, assets, and resources. It operates on the foundational premise that trust is never implicit based solely on physical or network location, such as an internal corporate network versus the internet or asset ownership. Instead, a Zero Trust architecture assumes that a network may always be potentially compromised, both externally and internally. This approach contrasts sharply with traditional security models, where the internal network is considered a relatively safe zone and trusted entities are defined without enabling proper verification [NIST, 2020; Colomb *et al.*, 2022].

At its core, Zero Trust aims to minimize uncertainty when enforcing accurate, least-privilege per-request access decisions in information systems and services. This requires entities to assume no implicit trust, continually evaluate risks, and make fewer secrecy assumptions, while enacting protections to mitigate these risks. NIST SP 800-207 outlines several fundamental tenets, of which the following are particularly relevant:

(i) All data sources and computing services are considered resources, including documents, infrastructure, and services;
(ii) all communication must be secure regardless of network location;
(iii) access to individual resources is granted on a per-session basis. The trustworthiness of the requester is strictly limited to the minimum privileges necessary for the task; authentication for one resource does not automatically confer access to others;
(iv) no asset is inherently trusted; and
(v) all resource authentication and authorization are dynamic and strictly enforced before access is allowed. This embodies a continuous cycle of requesting access, scanning and assessing threats, and continually re-evaluating trust in ongoing communications.

Zero Trust emerged as a response to the growing complexity of modern computing environments, characterized by remote users, personal devices, and cloud-based resources that extend beyond traditional organizational boundaries.

Rather than protecting static network segments, ZT focuses on safeguarding individual assets—documents, data, and services—by enforcing continuous verification and minimizing implicit trust [NIST, 2020; Colomb *et al*., 2022]. These principles directly inspire the design of our model: sensitive operations and document validations are executed exclusively within the user's trusted client-side environment, reducing the attack surface and avoiding reliance on intermediary servers or network segments.

In summary, Zero Trust provides the architectural rationale for the client-side design adopted in this work. By enforcing explicit verification of every cryptographic operation and removing implicit trust from intermediary servers, our model translates these principles into a practical mechanism for achieving document secrecy. This alignment is not merely conceptual—it establishes the technical foundation for ensuring that sensitive data, including documents and private keys, remain confined to the user's controlled environment throughout both signing and verification processes.

4 Digital Signature Web-platforms

Traditional digital signature applications, such as Adobe Acrobat [Adobe Inc., 2024], operate locally on a user's device, offering PDF editing, creation, and document signing features. These applications require a valid digital certificate accessible on the device or via external media, such as a smart card. For the certificate to be recognized as valid, the certificate chain, including any root and intermediate certificates, must be installed and trusted. Users are responsible for managing their certificates and ensuring the certification chain is properly configured. Additionally, they must follow procedures to mitigate risks, such as device loss and requesting the revocation of any certificate associated with a compromised private key.

In contrast to traditional desktop applications, numerous digital signature platforms are accessible via web browsers. These cloud-based solutions offer the convenience of signing documents from any internet-connected device without requiring local software installation. Web platforms simplify the signing process by eliminating the need for users to manage certificates and private keys. However, this approach introduces new security challenges as the platform provider stores and manages sensitive cryptographic materials. While these platforms often employ medium-term certificates reused across multiple users, the centralized management of keys and certificates necessitates robust security measures to protect against unauthorized access and data breaches. Examples of such platforms include Cryptomathic [2023]; Bit4id [2021]; Ascertia [2018]; NextSense [2023]; DigitalSign [2023]; UFSC [2019]; Brazil, Economy Ministry [2021]. In our study of signature web platforms, we meticulously examined the often implicit flowcharts underpinning their operations.

4.1 Sign

Figure 1 presents a simplified overview of the general digital signature process, assuming a pre-registered user. The standard sequence of events is as follows: ⓐ The user logs in to the platform; ⓑ The user uploads the document intended for signing; ⓒ The user selects either platform-based or user-owned certificate signing. For platform-based signing, a platform-generated X.509 certificate and key pair are often managed by a Hardware Security Module (HSM). For user-owned certificate signing, the user can upload a new X.509 certificate or utilize an existing one stored on the platform; d_1 The document's hash is computed, following a process similar to that outlined in [Aciobăniei *et al*., 2024]. This hash is then signed using a back-end library, and the resulting signature is embedded in the document alongside the platform's certificate; d_2 Alternatively, if the user opts for their own certificate, the private key and certificate are made accessible via the web platform; ⓔ The user's certificate and private key are transmitted to the back-end signature library; ⓕ The document is retrieved; and ⓖ The document is signed using the user's private key.

Regardless of whether option d_1 (platform-based signing) or d_2 (user-owned certificate signing) is chosen, the document is inherently exposed to the platform, even when using a secure communication channel. Furthermore, option d_2 also exposes the user's private key to the platform. Conversely, while option d_1 does not compromise the user's private key, it introduces the concept of a proxy signature, where the platform effectively signs on behalf of the user. This type of signature is only valid when verified by the platform itself, as the user's identity is neither independently authenticated nor explicitly associated with the signed document via a digital certificate. The examined signature web platforms lack transparency regarding document storage methods. While some platforms may employ encryption, the encryption keys are likely managed by the platform and stored within the database or archive infrastructure.

Shannon's Secrecy: *Signature* analysis Signature web platforms raise significant concerns when analyzed through the lens of Shannon's secrecy types. Firstly, a violation of **SST-1** occurs as the mere existence of a document is revealed to entities beyond the signatories. This vulnerability exposes the risk of targeted attacks on the platform's infrastructure and its trusted operators, providing adversaries with opportunities to exploit weaknesses through phishing or social engineering. Furthermore, the absence of robust obfuscation mechanisms leads to a violation of **SST-2**, which occurs when sensitive metadata, such as file names, timestamps, signatory details, or file size, are not adequately protected, allowing adversaries to infer critical information about the document's nature or context without direct access to its content. Finally, the limited transparency regarding document storage suggests a high probability that **SST-3** is compromised, meaning adversaries could potentially access or deduce the document's content if they breach the platform's infrastructure. Although many platforms implement basic access control, they often fail to enforce robust key management protocols, prioritizing user experience over security. One possible improvement is the implementation of *Identity-Based Encryption (IBE)* [Boneh and Franklin, 2001], which simplifies key management. However, this approach introduces the risk that a compromised

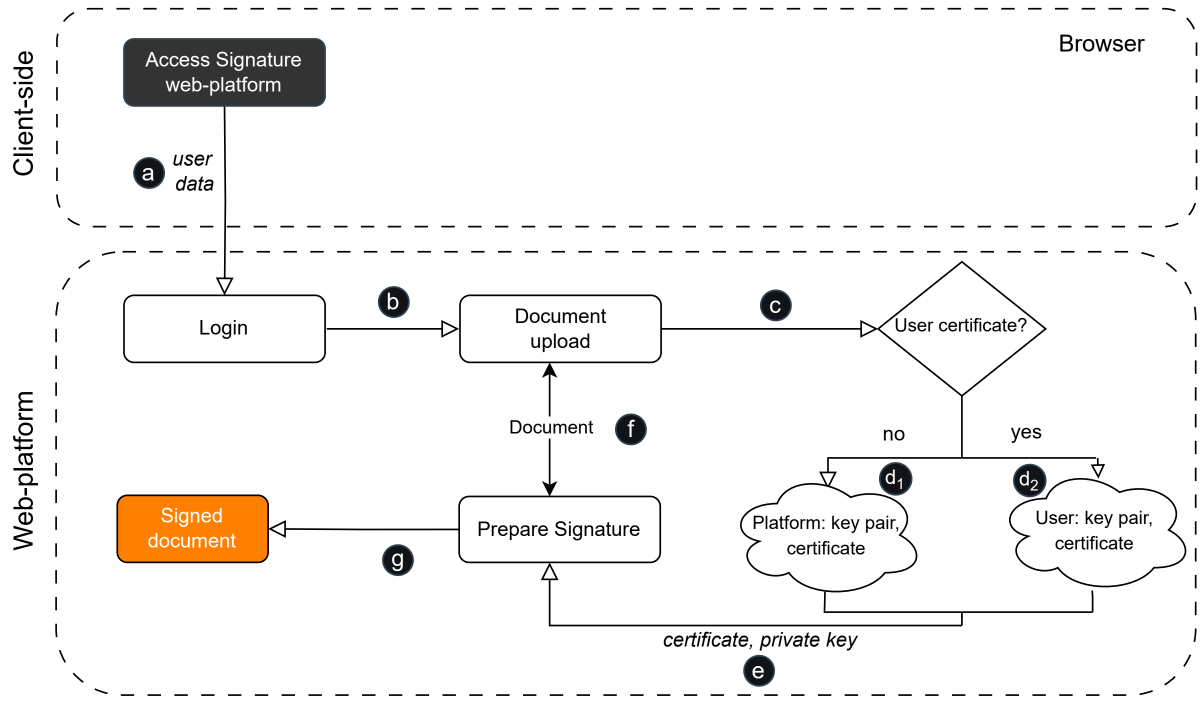

Figure 1. Simplified flowchart of the digital signature process on signature web platforms.

master key could jeopardize the security of the entire system.

4.2 Verify

A simplified overview of the general document verification process on digital signature platforms, assuming the user has access to a signed document. The standard sequence of events is as follows: i) The user uploads the signed document to the platform; ii) the platform extracts the digital signature and metadata from the document, including the signer's certificate and the signature's hash; iii) the platform validates the signer's certificate by checking the certificate chain against a trusted root CA, often using Trusted Service Lists (TSLs) or Certificate Revocation Lists (CRLs); iv) the platform recalculates the hash of the signed document to verify that it matches the hash embedded within the signature; v) if the hash matches and the certificate is valid, the platform confirms the signature's authenticity, integrity, and validity; vi) the platform generates a verification report detailing the results, including any warnings or errors (e.g., expired or revoked certificates, mismatched hashes, or untrusted certificate authorities); vii) the report is presented to the user, confirming whether the document's signature is valid and the document has not been tampered with.

Unlike the scenario of querying TSLs in Step iii for each individual signature operation, digital signature verifiers typically rely on cached or scheduled TSL retrievals to remain efficient. The frequency of TSL updates depends on the relevant jurisdiction or policy requirements, often ranging from every 48 hours to weekly in some regions, such as in the European Union under the eIDAS regulation.

Shannon's Secrecy: *Verify* analysis During the verification process, the document is exposed to the server for processing. This alone constitutes a **violation of SST-1**, as the mere act of involving an external entity (the platform) means that the existence of the document is known by a party outside the restricted group of individuals or systems that should have access to it. Even if the document is processed securely and discarded immediately afterward, the knowledge of its existence introduces the potential for targeted attacks on the platform's infrastructure or personnel.

Moreover, while some platforms claim to process documents only temporarily in memory and discard them immediately after verification, others may store the uploaded documents for purposes such as audit logs or future user access. Permanent storage exacerbates secrecy violations, particularly with respect to **SST-3**, as adversaries could infer or access the document's contents if the platform's infrastructure is compromised.

Even when platforms do not store the full document, they often retain **logs and metadata** of the verification process. These records can include:

- Information about who signed the document, such as the signer's name, email address, or certificate details.
- Metadata about the certificate issuer (e.g., the Certificate Authority), the certificate chain, and the timestamp of the signature.
- The name of the uploaded document and the hash of its contents.
- Logs of operations performed during the verification process.

Such metadata can reveal significant information about the document, its context, and the involved parties, even without

direct access to the document itself. This storage of metadata aligns with a **violation of SST-2**, where adversaries could infer critical information without accessing the document's actual content. For example, the document name, timestamp, or signer's details could expose sensitive relationships, operational timelines, or the nature of the document (e.g., classified policies or strategic agreements).

5 Proposed Model

We present a novel model for enhancing document secrecy within online signature platforms, inspired by the One-Time Digital Certificates approach [Mayr *et al.*, 2024]. Our model significantly bolsters security by enabling users to generate cryptographic key pairs and CSRs directly within their web browsers. In doing so, all sensitive operations, including document signing and verification, occur entirely on the client side, ensuring that the actual document is never transmitted or processed by the platform. This approach aligns with the Zero Trust paradigm by eliminating implicit trust in external servers and minimizing potential attack surfaces.

5.1 Sign

The proposed model for document signing is illustrated in Figure 2, in contrast to the traditional model shown in Figure 1. This method shifts most processing to the client side, where a browser-based library performs all necessary functions for document signing. Once the user is authenticated, all cryptographic operations occur locally, and the document remains in strict secrecy, with no sensitive document content leaving the client environment.

Initially, (a) the user logs into the signature web platform. Subsequently, (b) the platform transmits a signature library to the client's browser. (c) A document selection form is displayed within the web platform's front-end; upon selecting a document, the form activates the signature library directly in the browser. (d) The library generates a key pair, creates a document hash, embeds it within a CSR extension, and signs the CSR using the generated private key. (e) The CSR is sent to the CA (which, in this case, is the web platform). (f) The CA issues a One-Time Certificate, accompanied by the trust certification chain, and sends it to the client's browser. (g) The signature library retrieves the document and the One-Time Certificate with its trust chain, (h) proceeding to sign the document. We remark that in this process, the only information leaked to the CA is the hash of the document the user wishes to sign, which can be further obscured by using a nonce as per the recommendations made in [Mayr *et al.*, 2024] regarding OTCs privacy characteristics.

Our model requires user authentication only at step (a), with optional additional verification at step (e) based on specific requirements. The identity provider might enforce multi-factor authentication to robustly establish user identity [Jacomme and Kremer, 2021; Prabakaran and Ramachandran, 2022; Perottoni *et al.*, 2023]. Furthermore, the CA must employ robust key storage for its own keys and robust certificate issuance mechanisms, such as HSMs, specifically designed for PKI [Barker and Barker, 2018].

Shannon's Secrecy: *Signature* analysis Furthermore, because no sensitive metadata (such as file names, timestamps, or file sizes) is transmitted, adversaries cannot infer critical details about the document, thus preserving **SST-2**. Finally, as no portion of the document is transmitted to the platform, the system achieves cryptographic isolation and complies with **SST-3**. Even if an adversary were to access the platform or intercept communications, robust cryptographic protocols confined to the client environment prevent any reconstruction of the document's content. This client-side isolation is a practical application of Zero Trust, eliminating a central point of failure.

The digital document intended for signing remains within the execution environment alongside the generated key pair, with only the public key transmitted within the CSR. This ensures that the document never leaves the client environment, preserving its secrecy and ensuring **SST-1**, as neither the web platform nor any external entity possesses knowledge of the document being signed. Additionally, as the document never leaves the client environment, no sensitive metadata-such as file names, timestamps, or file size-is exposed, ensuring that adversaries cannot infer critical information about the document's nature or context, thereby preserving **SST-2**. Furthermore, cryptographic isolation is achieved since neither the document's content nor any portion of it is transmitted to the platform. This aligns with Shannon's concept of a "true" secrecy system, ensuring compliance with **SST-3**. Even if an adversary gains access to the platform or intercepts signals, they are unable to deduce the document's content due to the use of robust cryptographic protocols confined entirely to the client environment. The independence from the platform's infrastructure removes a potential point of failure, further reinforcing that no information useful for deducing the document's content is ever exposed. This client-side isolation, where the document and sensitive operations are confined to the user's environment and no implicit trust is extended to the platform for these operations, directly reflects the Zero Trust principle of minimizing trust zones and verifying interactions explicitly (See Section 3.3).

5.2 Verify

In addition to enhancing the signing process, our model proposes a fully client-side approach to document verification. This ensures that sensitive documents remain confined to the user's environment and eliminates the need for external servers to process or store document-related information.

Initially, (1), the user accesses the verification interface and selects the signed document. (2) The verification library, provided by the web platform, is loaded directly into the user's browser. (3) The browser retrieves TSLs from trusted sources for each verified document, establishing the validity of certificate chains. The TSL lookup is performed in real-time for each verified document, ensuring the system remains compatible with external or standard signers. However, if multiple signatures exist within the same PDF, only one TSL query is required for the entire PDF, rather than one query per signature. This practice maintains compatibility, especially for documents signed in other regions or with different CAs, as external TSL sources can be consulted to validate the certifi-

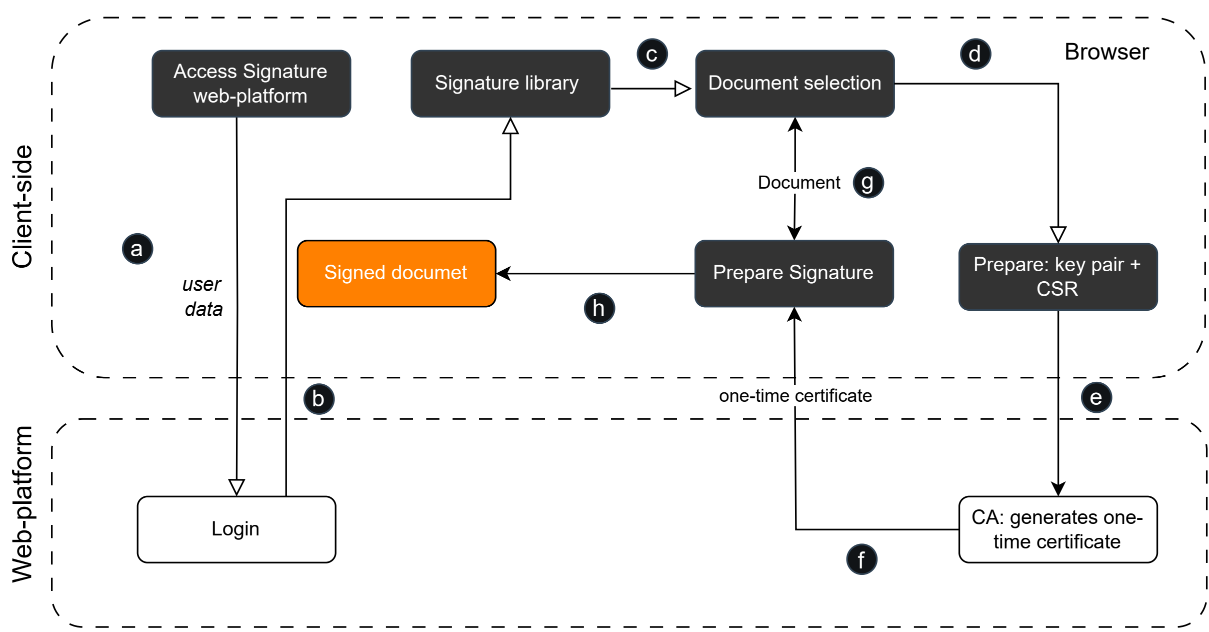

Figure 2. The proposed signature secrecy model for web platforms.

cate chain. ❹ The library extracts the signer's certificate, the certificate chain, and the signature hash from the document. ❺ The library verifies the integrity by recalculating the hash of the signed portion of the document and comparing it with the hash embedded in the OTC. ❻ The certificate chain is validated against the TSLs and CRLs to check for revocation or expiration. ❼ The library verifies the documents' signatures. ❽ The results are presented to the user, confirming the document's authenticity, integrity, and compliance with secrecy principles.

Shannon's Secrecy: *Verify* analysis. In the proposed client-side verification model, the signed document remains entirely within the user's browser, thus preventing the platform from learning about its existence and ensuring **SST-1** is upheld. Because all cryptographic operations-such as PDF parsing and signature hash verification-occur locally, the platform neither receives nor stores logs or metadata (e.g., document name, timestamps, or file size). This approach preserves **SST-2**, as no information is available for an attacker to infer the document's contents or context.

Moreover, external queries for TSLs and CRLs do not expose the document itself since only certificate-related data are exchanged with trusted external sources. Consequently, even if the platform or these sources are compromised, they cannot reconstruct the document or derive its content. This client-side verification strategy, aligned with Zero Trust principles, ensures the integrity and authenticity checks remain transparent to the user while robustly protecting document secrecy throughout the entire verification workflow without compromising the document's secrecy.

6 Model Implementation

Our implementation aims to sign and verify PDF signatures following the ISO-32000 and PAdES standards [ISO, 2020; ETSI, 2024], performing all cryptographic operations client-side in the user's browser to preserve document secrecy as specified in Section 5. This implementation was built using *JavaScript* [Eich, 1995] due to its presence in modern web browsers and its amenability to transparency and auditability, as code executed client-side can be readily inspected. While *WebAssembly* presents an alternative for client-side computation, JavaScript's direct interpretability was favored for this research prototype, where clarity of the client-side processes is paramount. This decision influenced the selection of available cryptographic libraries and frameworks.

While the proposed model is designed to be algorithm-agnostic with respect to asymmetric signature algorithms, the current model implementation uses RSA-based signatures due to their widespread adoption in traditional PKIs and the robust support within the selected libraries. In our implementation, the user's browser generates a 2048-bit RSA key pair for every document signature. The CA, in turn, uses a 4096-bit RSA private key for issuing the corresponding One-Time Certificate.

> *The evaluation of other signature schemes, such as EdDSA (which could offer performance benefits and smaller signature sizes), was considered outside the scope of this initial validation, which focuses on the viability of the client-side secrecy model itself. Similarly, addressing the long-term threat of quantum computers through the integration of Post-Quantum Cryptography (PQC) is an important area for future development but is not covered in the current performance evaluations. These choices, influence the performance results presented in Section 7, but the primary aim here is to demonstrate the architectural feasibility and secrecy advantages of the proposed client-side model.*

We executed a combination of the *Web Crypto API* [W3C, 2017] and established JavaScript libraries such as *pkijs* [Glob-

alSign and Ventures, 2014], *asn1js* [Ventures, 2013], and *Norge-Forge* [Digital Bazaar, 2010], specializing in key generation, CSR generation, and PDF signing. To manipulate PDF structures and create signatures, we created a customized fork of *sign-pdf-lib* [Remdra, 2023], adapted for One-Time Certificates and with a fix for a small issue that prevented compatibility with PAdES[1]. The signature scheme employed is RSASSA-PKCS1-v1.5, using SHA-384 as the hash function. [Jonsson and Kaliski, 2016; Hansen and Eastlake 3rd, 2011].

> *We acknowledge that RFC 8017 recommends RSASSA-PSS for new applications [Jonsson and Kaliski, 2016]. In this model implementation, the use of RSASSA-PKCS1-v1.5 was primarily determined by the available functionalities and mature support within the chosen PDF manipulation and signing libraries (specifically,* sign-pdf-lib *and its underlying dependencies).*

For further details and to access the complete source code of our Web Signature Secrecy Signer implementation, as described in this paper, please refer to our public repository[2][3].

6.1 Application Flow

Sign Upon accessing the prototype, users are presented with the application's home screen and receive the cryptographic libraries needed for signing documents within the user's browser. Clicking the authentication button redirects users to an external identity provider. Successful authentication returns users to the signature screen, where they can browse for documents on their computer. Clicking the *sign* button initiates the execution of the PDF signature pseudo-algorithm outlined in Algorithm 1.

First, a placeholder is inserted into the document provided by the user. This placeholder may include various data attributes, such as the signing date, the reason for the signature, and the name of the software used for generating it. However, the most critical components are Byte Range, which defines the portion of the document that will be signed using the private key, and `/Contents`, which contains the ASN.1 structure holding the actual signature, known as Cryptographic Message Syntax (CMS). As the application doesn't know the exact value of the CMS yet, `/Contents` is initially filled with zeros.

Subsequently, a newly created RSA keypair is used to generate a CSR containing the document's cryptographic hash. This hash is embedded to cryptographically bind the certificate request to the specific document, ensuring a one-to-one association between the Certificate (OTC), the user's key pair, and the document (via its hash) that will be signed. A request, bundled with the CSR and the authentication data is transmitted to the server. The server relays this request to the CA micro-service. Upon validating the authentication token, the CA issues a One-Time Certificate and returns a PKCS#7 with the full certificate chain to the server, which in turn returns it to the originating browser code. After receiving the PKCS#7, the browser code creates a PKCS#12 with both the certificate chain inside the PKCS#7 and the user's RSA private key.

Finally, as denoted by Housley [2009], a CMS is created containing the certificate chain, the digest calculated previously, and the signing time [4]. Those attributes are then signed by the user's private key and the resulting signature is stored inside the CMS structure. After the creation process ends, the CMS is then directly inserted inside the */Content* attribute, thus ending the signature algorithm.

Algorithm 1: Signing PDF documents in the client's browser using One-Time Certificate

1: **function** Sign($Document$, $userIdentity$)
2: $PreparedDocument \leftarrow$ `PrepareDocSignature(Document)`
3: $BytesPDF \leftarrow$ `ExtractContentToBeSigned`(PreparedDocument)
4: $DocHash \leftarrow$ `HashByteRange(BytesPDF)`
5: $KeyPair \leftarrow$ `GenerateKeyPair()`
6: $CSR \leftarrow$ `CreateCSR( KeyPair, DocHash, userIdentity)`
7: $PKCS7 \leftarrow$ `SendCSRToCA(CSR)`
8: $PKCS12 \leftarrow$ `CreatePKCS12(KeyPair, PKCS7)`
9: $CMS \leftarrow$ `SignCMS(BytesPdf, PKCS12)`
10: $SignedDoc \leftarrow$ `ConstructSignedDoc(BytesPdf, SignedCMS)`
11: **return** $SignedDocument$
12: **end function**

Each step of the pseudocode outlined in Algorithm 1, is described below:

`PrepareDocForSignature:` This function prepares the PDF document to accommodate the digital signature by adding the necessary placeholder without altering any pre-existing content that may already be signed.

`HashByteRange:` This function computes the cryptographic hash of the PDF document that should be signed.

`GenerateKeyPair`: This function generates a public-private key pair for the signing process.

`CreateCSR`: This function creates a CSR containing the public key, user identity, and document hash (in an extension). The CSR is signed with the private key.

`SendCSRToCA`: This function sends the CSR to the CA and receives a PKCS#7 containing the certificate chain.

`CreatePKCS#12`: This function converts the PKCS#7 into a PKCS#12 format using the private key.

`SignCMS`: This function creates and signs the CMS using the provided certificate, the BytesPDF, and the private key from the PKCS#12. Returns the signed CMS.

`ConstructSignedDoc`: This function replaces the placeholder introduced on `PrepareDocForSignature` by the signed CMS, thus creating a new signed document.

`ExtractContentToBeSigned`: This function retrieves the content of the prepared document that will be signed

[1] `https://github.com/enzoBrum/otc-sign-pdf-lib`
[2] `https://github.com/wellisilvano/web-secrecy-signature-PKI`
[3] `https://codigos.ufsc.br/100000000343066/web-secrecy-signature`

[4] Despite it already being present inside the One-Time Certificate, it's important to include the digest directly on the CMS to remain compliant with International Organization for Standardization [2008] and Housley [2009]

by the user's private key.

Verify Upon accessing the prototype, users are presented with the application's home screen and receive the cryptographic libraries needed for verifying documents within the user's browser. By clicking the *verify* button and selecting a signed document, users can initiate the verification process entirely within the browser. The PDF signature verification pseudo-algorithm outlined in Algorithm 2

Subsequently, the client browser begins by fetching TSLs from trusted sources to establish the trustworthiness of certificates. For each signature embedded on the document, the verification process then performs a series of operations to validate both the authenticity of the signer and the integrity of the signed data. Specifically, the client extracts the CMS structure embedded within the document's */Contents* object, retrieves the certificates used in the signature and uses the */ByteRange* to obtain the portion of the document that was signed by the private key. To ensure the integrity of the CMS itself, the application uses the public key within the One-Time Certificate to verify if the signature inside the CMS is valid. Then, it calcutes the cryptographic hash of the document's signed portion and compares it with two stored values: the hash inside the CMS and the hash within the One-Time Certificate. Validating the CMS signature ensures that none of the signed attributes have been altered since the signature was created; matching the signed portion hash to the CMS's hash confirms that the document itself remains unchanged; comparing this same hash with the hash in the One-Time Certificate establishes that the certificate is uniquely bound to that specific document.

Additionally, the application validates the certificate chain against the TSLs and checks the revocation status of each certificate using the x509 extension. To maintain comparability with the legacy and verify signed documents with another signer, we also consult CRLs. To retrieve external resources such as TSLs, CRLs, and parent certificates, the client uses the server as a proxy, which facilitates the retrieval process without requiring the document itself to be sent externally. This design circumvents Cross-Origin Resource Sharing (CORS)[5] restrictions, ensuring smooth communication with external domains.

By completing all these steps locally for each individual signature, the application confirms the validity of the document while maintaining strict confidentiality of its content, ensuring its integrity, authenticity, and trustworthiness.

Each step of the pseudocode outlined in Algorithm 2, is described below:

`FetchTSLs:` This function retrieves all certificates from each TSL specified by $urlTSLs$ or by the <PointersToOtherTSL> tags within the TSLs.

`ParsedPdf:` This function parses the PDF into a structured object for easier information retrieval.

[5] CORS (Cross-Origin Resource Sharing) is a browser security mechanism that blocks web pages from directly fetching resources from different origins (domains, protocols, or ports). To avoid this restriction without sending the document externally, the verifier sends only the resource URL to the platform server, which fetches it on behalf of the client, returning it as a same-origin response.

Algorithm 2: Verifying PDF documents in the client's browser using One-Time Certificate

```
1: function VerifySignature(ParsedPdf, TSLs, SignatureIdentifier)
2:     BytesPDF ←
   ExtractSignedContent(ParsedPdf, SignatureIdentifier)
3:     CMS ←
   ExtractCMS(ParsedPdf, SignatureIdentifier)
4:     CertChain ← ExtractCertChain(CMS)
5:     SignerCert ← ExtractSignerCert(CMS)
6:
7:     CmsIntegrity ← VerifyCMSIntegrity(CMS)
8:     DocHash ← HashByteRange(BytesPDF)
9:     DigestDoc ← ExtractDocHashSignature(CMS)
10:    DocIntegrity ←
   VerifyDocIntegrity(DOCHash, DigestDOC)
11:
12:    CRLs ← FetchCRL(CertChain)
13:    VerifyCertAu ←
   VerifyCertAu(DocHash, SignerCert)
14:
15:    TrustedChain ←
   VerifyCertChain(CertChain,TSLs,CRLs)
16:
17:    return (CmsIntegrity, DocIntegrity,
   TrustedCertChain, VerifyCertAu)
18: end function
19: function Verify(Document, urlTSLs)
20:    TSLs ← FetchTSLs(urlTSLs)
21:    ParsedPdf ← ParsePDF(Document)
22:    SignaturesPdf ← ExtractSignatures(ParsedPdf)
23:    VerificationResults ← ∅
24:    for SignatureIdentifier ∈ SignaturesPdf do
25:    VerificationResult ←
   VerifySignature(ParsedPdf, TSLs, SignatureIdentifier)
26:    VerificationResults ←
   Append(VerificationResults, VerificationResult)
27:    end for
28:    return VerificationResults
29: end function
```

`HashByteRange:` This function calculates the cryptographic hash of the PDF document within the byte range specified by the current signature. Each signature is then verified individually, which includes computing the hash over its specific coverage /ByteRange

`ExtractCMS:` This function extracts the CMS from the Contents entry inside the signature dictionary.

`ExtractCertChain:` This function extracts the certificate chain inside the CMS. If there is a missing parent certificate, it will be fetched from the URL defined by the Authority Information Access extension inside the child certificate.

`ExtractSignerCert:` This function retrieves the certificate used to sign the CMS and, by extension, the document.

`VerifyCMSIntegrity:` This function verifies the integrity of the CMS as defined in [Housley, 2009].

`ExtractDocHashSignature:` This function extracts the message digest attribute within the CMS.

`VerifyDocIntegrity:` This function ensures that the message digest attribute ($DigestDoc$) is equal to the interval defined by the signature's /ByteRange entry.

`FetchCRL:` For each certificate in the certificate chain, this function retrieves the CRLs from the URLs specified in the CRL Distribution Points extension.

`VerifyCertChain:` This function verifies whether the certificate chain is trusted.

`VerifyCertAu:` This function verifies whether the digest of the interval defined by the /ByteRange matches the one contained within the One-Time Certificate.

`ExtractSignedContent:` This function retrieves the content of the document that was signed by the user's private key.

7 Results and Discussion

To assess the validity of the implementation, it is crucial to ensure that the signature created is interoperable, meaning it is recognized as a valid advanced signature by well-known verifiers. Additionally, since a significant portion of the code execution occurs within the client's browser, the execution times are also important for evaluating potential uses. We also discuss the advantages and disadvantages of the proposed model.

7.1 Signature Interoperability

This section assesses the compatibility of the generated digital signatures with industry-standard verification systems. The goal is to demonstrate adherence to current digital signature standards and ensure broad recognition across various platforms and applications. To demonstrate interoperability, we employed *pdfsig*, a command-line verification tool on Ubuntu [Poppler Utils, 2024]. Successful validation using *pdfsig* confirmed compatibility with legacy verification systems, as illustrated in Figure 3.

The signature, issued to a fictitious user, Alice Silva, included metadata like the signer's email (provided by the identity provider). Verification revealed signature timestamp, hash function, and full document coverage, demonstrating backward compatibility with traditional verification tools.

```
pdfsig signed_document.pdf
Digital Signature Info of: signed_document.pdf
Signature #1:
- Signer Certificate Common Name: Alice Silva
- Signer full Distinguished Name: E = alice.
    silva@secrecy.com, CN = Alice Silva, C = BR
- Signing Time: Dec 28 2024 13:24:17
- Signing Hash Algorithm: SHA-384
- Signature Type: adbe.pkcs7.detached
- Signed Ranges: [0-23448], [43450-44167]
- Total document signed
- Signature Validation: Signature is Valid.
```

Figure 3. *pdfsig* tool usage example to verify PDF digital signatures.

The Adobe software also validated the signatures, further confirming interoperability and compliance. We successfully tested and verified multiple signatures using the same methods.

7.2 Client-Side Execution Times

This section presents the performance analysis of the signature algorithm within a client-side browser environment.

We evaluated scenarios involving both signing and verification tasks. First, we examined the effect of adding multiple signatures to the same PDF file, using documents of approximately 1 MB, 5 MB, and 10 MB. Afterward, we conducted large-scale testing on 1,633 PDFs from a public library[6] to facilitate reproducibility. All evaluations were performed on a desktop system running Linux NixOS 25.05.20241218.d70bd19, equipped with 32 GB of RAM and an AMD Ryzen 7 5700x @ 4.66 GHz CPU. Automated tests were executed using *Selenium* and *ChromeDriver*.

7.2.1 Multiple Signatures in a Single File

In this scenario, we investigated the impact of inserting multiple signatures into a single PDF document for three file sizes: approximately 1 MB (Figure 4a), 5 MB (Figure 4b), and 10 MB (Figure 4c). We select five documents for each approximate size to minimize potential variability from unique PDF characteristics and measure the average execution time for each step.

Sign Figure 4 illustrates the signing time as a function of the number of signatures in a stacked bar chart for each step in the process. Our measurements show that the maximum signing time is approximately 3 seconds per signature for documents up to 10 MB in size. We also observe that as the PDF size increases, the total signing time grows accordingly-particularly due to the preparation, Signature, and Hash steps. By contrast, adding a new signature to a PDF that already contains one or more signatures does not significantly affect overall signing time, indicating that the application handles multiple signatures efficiently. We note that the hash calculation step becomes more prominent for files of approximately 10 MB (Figure 4c).

Figures 4–5 present controlled experiments with 1, 5, and 10 MB files, selected to isolate per-step behavior and analyze

[6] `https://github.com/tpn/pdfs`

the effect of multiple signatures within the same document. While these experiments help illustrate scalability trends for multiple signatures, the majority of real-world PDFs are is examined later in Figures 6–7.

Verify Figure 5 illustrates the verification time as a function of the number of signatures in a stacked bar chart, detailing each step in the process. As shown, verifying a PDF file of approximately 10 MB can take up to around 0.3 seconds. There is also a noticeable increase in total verification time as the number of signatures grows. Unlike the signing phase, verification requires individual reading and validation of each existing signature, including determining the certificate Byterange to consider. These repeated checks introduce additional overhead, particularly in the PDF parsing and DocHash steps. Another important factor is the TSL retrieval time, which will be examined in more detail within the context of batch signing.

7.2.2 Large-Scale Testing

In order to thoroughly evaluate our application, we performed tests on PDF files of multiple sizes and structures obtained from the public library previously mentioned. We examined the impact of various signing and verification steps through scatter plots (Figures 6 and 7), where the vertical axis represents execution time and the horizontal axis represents file size. The library mainly contains relatively small documents (many under 1 MB), which is consistent with the size distribution reported on the used library. As a more precise breakdown, 62.45% of the files are smaller than 1 MB, 79.14% are smaller than 2 MB, 91.29% are smaller than 5 MB, 96.01% are smaller than 10 MB, and 98.83% are smaller than 20 MB.

Signing Process From Figure 6, we observe that *Preparation*, *Hash*, and *Signature* noticeably affect the overall signing time in proportion to the size of the PDF. These stages, identified as `PrepareDocSignature`, `HashByteRange`, and `SignPDF` in Algorithm 1, require reading PDF data, making document size a relevant factor. We remark that while *Signature* could theoretically reuse the bytes already read by *Preparation*, doing so would involve major modifications to well-established libraries; thus, we chose not to implement such changes.

Conversely, *Key Generation*, *CSR Creation*, and *WebApp Response* steps exhibit independence from file size. RSA key-pair generation shows some variability due to its inherent randomness. Still, it remains effectively constant, averaging around 49.57 ms with a standard deviation of 25.47 ms, which is negligible when compared to the total signing time. The *WebApp Response* step depends on an external environment that provides the certificate after receiving the CSR; thus, it is largely influenced by potential network load than by the client machine's performance. Its average time is approximately 351.65 ms with a standard deviation of 113.81 ms. The *CSR Creation* step takes about 5.28 ms on average, with a standard deviation of 0.49 ms, indicating minimal variation relative to the other stages.

Since approximately 80% of the PDFs in the library we used are smaller than 2 MB, we performed a separate analysis on this subset to ensure robust statistical representation. We grouped the signed documents into intervals of 0.2 MB, up to 2 MB because there are statistical variations - whether due to the file size itself or specific PDF content such as spreadsheets, graphics, and text, we employed a stacked bar chart combined with boxplots (showing whiskers, quartiles, and distributions) to capture these differences (Figure 8).

From Figure 8, we see that client-side signing remains efficient even with a large number of documents; for files up to 2 MB, the average signing time does not exceed 1.5 seconds. Moreover, a clear correlation emerges between execution time and file size, which aligns with the discussion in Subsection 7.2.1 and is further validated here by the larger sample of tested documents. Another noteworthy observation is the presence of outliers., defined here as total signing times lying outside the 75th percentile of each group. An examination of these higher-end outliers shows that they often correspond to PDFs containing numerous tables.

Verification Process From Figure 7, we observe that *ParserPDF*, *DocHash*, *CMSVerification*, *CMSParsing*, and *CRLs* significantly influence the overall verification time, which increases with the size of the PDF. These stages represented by the functions `loadPdf`, `HashByteRange`, `VerifyCMSIntegrity`, CMSParsing (`ExtractCMS`, `ExtractSignerCert`), and `FetchCRL` in Algorithm 2, rely on reading PDF structures, similar to the signing process. Notably, `CMSVerification` shows a positive correlation of 0.456, and `FetchCRL` (CRL retrieval) shows 0.480, indicating a moderate relationship between these steps and overall verification time.

To quantify the effect of file size on total verification time more accurately, we selected PDF files up to 20 MB-accounting for 98.82% of our dataset-and grouped them in 1 MB intervals to reduce extreme distortion. Because variations may arise from file size or specific PDF content, we used a stacked bar chart combined with boxplots, showing whiskers, quartiles, and distributions (Figure 9).

In Figure 9, the thin red line near the top of each bar indicates the mean total verification time, showing a correlation between file size and total time, particularly above 10 MB, where fewer documents are available, reducing statistical relevance. Outliers already noted in the signing process are even more pronounced during verification. While the average verification time for files between 0 and 1 MB is around 0.1 seconds, maximum values can reach 0.7 seconds. A closer probe reveals that these elevated times generally stem from network delays affecting the retrieval of the TSL.

The chart also highlights a practically constant dependence of total verification time on TSL retrieval. TSLs are crucial for backward compatibility of signature verification and interoperability with legacy systems, and different jurisdictions (e.g., the European Union or Brazil) maintain distinct TSLs of varying sizes and availability. Consequently, if verifying older or externally signed documents is required, verification times may increase due to repeated TSL lookups. However, if external compatibility of Web Verifier with signed documents in other Web Signers is not essential, verification times will remain closer to those reported by the library implementation in our study. Overall, excluding outliers, the maximum

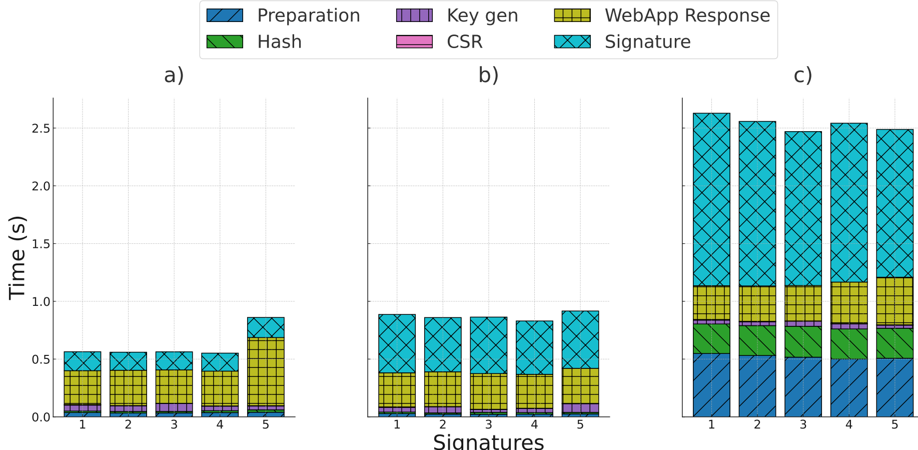

Figure 4. Signature time steps from one to five signatures in a document. Stacked bar charts show execution time for each step in the signing process: *Preparation*, *Hash*, *Key generation*, *CSR creation*, *WebApp response*, and *Signature generation*, across three document sizes: (a) 1 MB, (b) 5 MB, and (c) 10 MB.

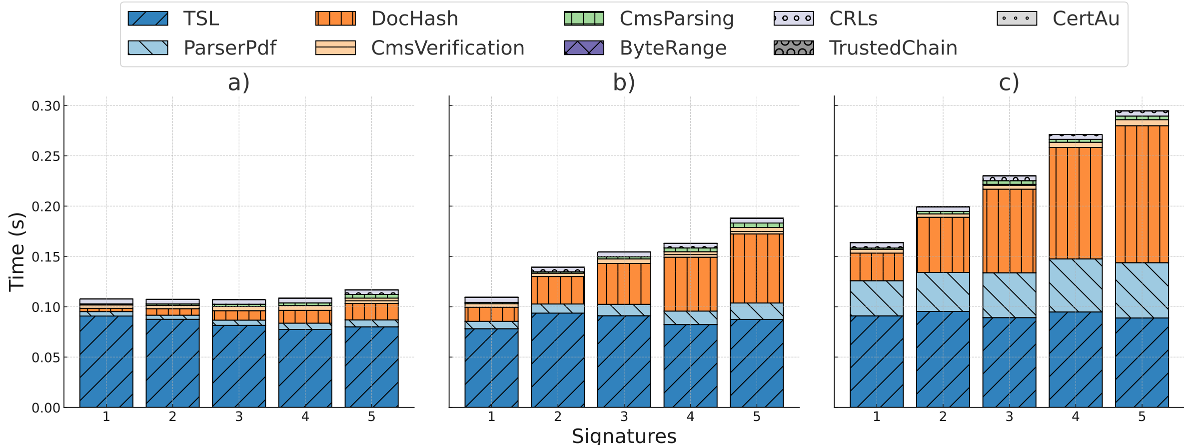

Figure 5. Verification time steps from one to five signatures in a document. Stacked bar charts display the execution time for each step in the verification process: *TSL retrieval*, *ParserPDF*, *DocHash*, *CMSVerification*, *CMSParsing*, *ByteRange*, *CRLs*, *TrustedChain*, and *CertAu*. The charts correspond to three document sizes: (a) 1 MB, (b) 5 MB, and (c) 10 MB. Note that the verification time includes one hash computation per embedded signature (one per coverage /ByteRange)

verification time for files up to 20 MB is under 0.4 seconds.

7.3 General Analysis

A comparative table has been created to evaluate the key features of the proposed model in relation to traditional web-based signature platforms. Table 1 provides an analysis of seven critical aspects, including document secrecy, private key secrecy, key management complexity, certificate handling, support for multiple signatures, and performance across both large and small documents.

Table 1. Comparison between traditional signature web-platforms with our model

Feature	Traditional Web Platforms	Our model
Document Secrecy	Conditional	Unconditional
Private Key Secrecy	Conditional	Unconditional
Private Key Management	Complex	Simplified
Certificate Handling	Resource-Intensive	Streamlined
Multiple Signers	Link-Based	Out-of-Band
Signature Performance	Server Power Dependency	Client Machine Dependency
Large documents performance	Slower	Faster
Small documents performance	Faster	Slower

7.3.1 Document Secrecy

Traditional signature web platforms provide conditional document secrecy, as they depend on access control mechanisms susceptible to attacks, especially those targeting the platform's infrastructure or trusted operators. The need to upload documents to these platforms exposes sensitive data to all violations of Shannon's secrecy principles (see Section 4). In contrast, our platform ensures unconditional document secrecy by eliminating the need to upload documents to a server. This removes the risk of secrecy violations, eliminating the platform as a potential point of failure (See Section 5). Although user-side vulnerabilities, such as browser exploits, remain a possible threat, the overall risk is reduced by restricting document handling to the client environment.

Unlike traditional verification platforms, where the document must be uploaded to a remote server, the client-side verifier ensures that the document remains entirely on the user's device. This avoids all Shannon secrecy violations (**SST-1**, **SST-2**, **SST-3**). By not transmitting the document or any metadata (e.g., timestamps, filenames, signer details), the verifier eliminates the risk of data leakage, even if external servers are compromised. Our client-side verifier completely

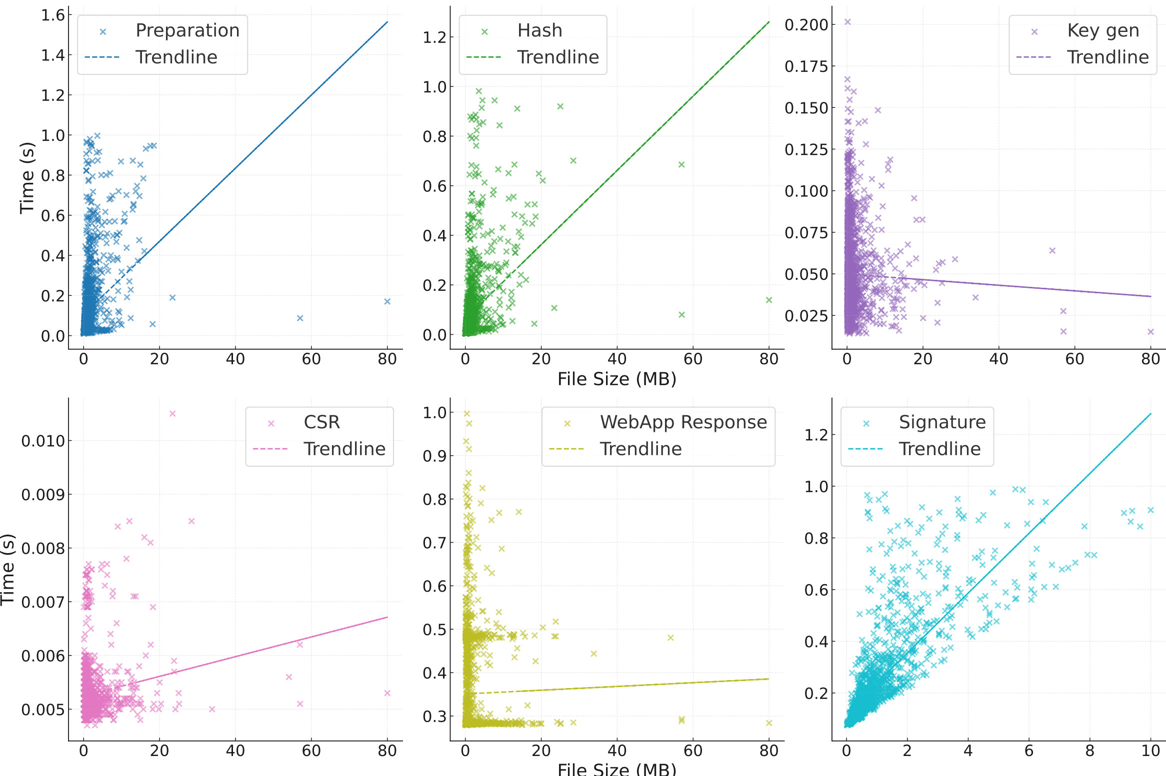

Figure 6. Signature steps (Large scale experiment). Scatter plots showing the relationship between file size (in MB) and execution time (in seconds) for the six main steps in the signing process: *Preparation*, *Hash*, *Key generation*, *CSR creation*, *WebApp response*, and *Signature generation*

avoids these pitfalls by keeping the document and all cryptographic operations local to the user's device; it eliminates the need to transmit the document or its metadata.

7.3.2 Private Key Secrecy

Platforms that utilize x.509 user certificates (as described in Section 4, step d_2) encounter challenges in maintaining the confidentiality of private keys, which are under the user's control rather than the platform's. This presents inherent risks, as the platform may inadvertently expose or mishandle private keys, leading to potential breaches of secrecy. In contrast, our proposal ensures unconditional private key secrecy by employing one-time signatures and generating CSR entirely on the client side. The only data transmitted to the signature web platform is the cryptographic hash of the document to be signed. The certificates are used only once; there is no risk associated with long-term key storage (see Section 5).

7.3.3 Private Key Management

Ensuring that private keys are kept safe and secure is of the highest concern when dealing with digital signatures, especially considering that end users might not be as versed in security protocols as desired. Therefore, to lessen the burden of key management on their user base and as an attempt to increase overall security, some web signature platforms allow users to store their private keys on their platforms. These private key management delegation systems clearly violate Shannon's secrecy principles by exposing sensitive cryptographic material. In contrast, our model eliminates the need for continuous key management by leveraging One-Time Certificates with client-generated CSRs. In addition, the certificate is restricted to a single document and does not require further management as it will not be reused for additional signatures. Consequently, our approach eliminates the complexities traditionally associated with managing private keys while ensuring high security through robust authentication.

7.3.4 Multiple Signers

Traditional web signature platforms offer a convenient user experience, especially for documents requiring multiple signatures. Users upload the document, and signers receive a link or email to authenticate and sign, with the platform automating document distribution and signature collection. In contrast, the proposed model, while more secure, adds complexity. Each signer must transmit the document securely (e.g., via encrypted email or USB drive) and pass it to the next. A document management system can help streamline this process, but it introduces additional steps and potential delays compared to web signature platforms.

We remark that this constraint is present only in formats that do not allow parallel signatures, such as PDF, where each subsequent signature must sign its predecessor. Formats that do allow these types of signatures, such as XML, allow the original message to be sent to every signer simultaneously. After the signatures have been collected, they can be appended to the signed document with no issues in regard to signature ordering [ISO, 2020].

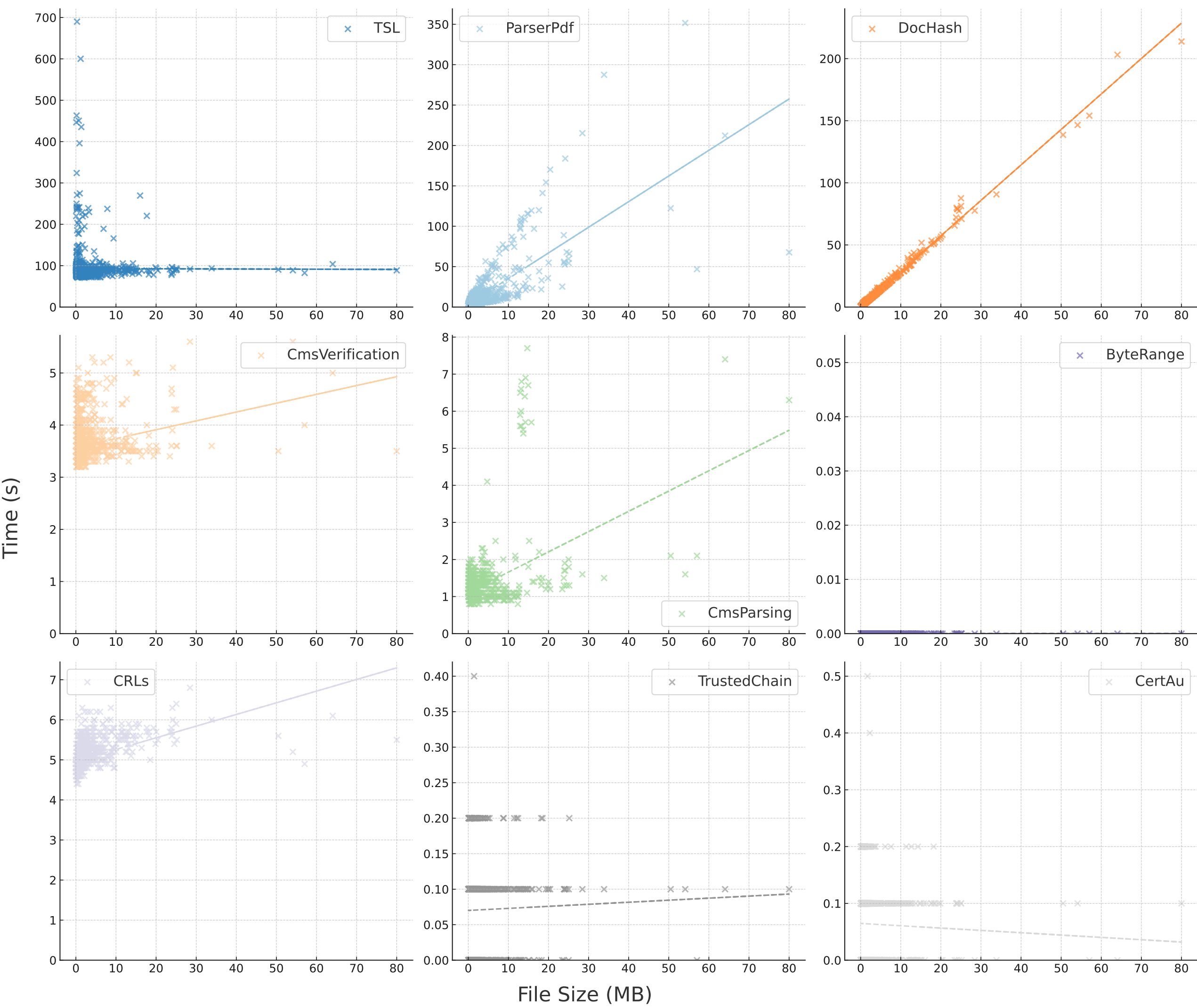

Figure 7. Verification steps (Large-scale experiment). Scatter plots showing the relationship between file size (in MB) and execution time (in seconds) for the nine main steps in the verification process: *TSL retrieval*, *ParserPDF*, *DocHash*, *CMSVerification*, *CMSParsing*, *ByteRange*, *CRLs*, *TrustedChain*, and *CertAu*.

7.3.5 Signature Performance and Document Size

Traditional platforms typically leverage powerful servers to expedite signing processes, often outperforming our approach for standard document sizes. However, network upload times can become a bottleneck for exceptionally large files. In such cases, our method presents a potential advantage. For example, a preliminary analysis of WikiLeaks data indicates that most classified documents are under 5MB, which our system can sign in approximately 1 second, while verification can be performed in approximately 0.15 seconds.

Signing performance depends largely on two main points: the hardware used for the signature and data transmission. In hardware comparisons, signatures are usually generated either by cryptographic tokens, which are small, specialized key management devices, or hardware security modules (HSMs); in this model, signatures are produced with consumer-grade hardware, which is typically faster than cryptographic tokens and either faster or comparable to HSMs. On the data transference side, our model should consume less bandwidth for documents with sizes lower than a digital certificate, as we need to download the OTC for each signed document.

8 Conclusion

This study introduces a novel client-side cryptographic model for digital signature web platforms, leveraging One-Time Certificates to establish a secure environment for signing sensitive documents. By confining all cryptographic operations to the user's browser, our model safeguards document secrecy and eliminates the need for key pair management on the client side. Performance evaluations demonstrate that the model effectively handles documents of varying sizes, providing efficient signing and verification processes, even for large files. The key contributions of this study include, but are not limited to:

Enhanced Document Secrecy: Protecting sensitive information by eliminating the need to upload documents to external platforms, preserving Shannon's secrecy principles.
Improved Private Key Management: Eliminating the risk

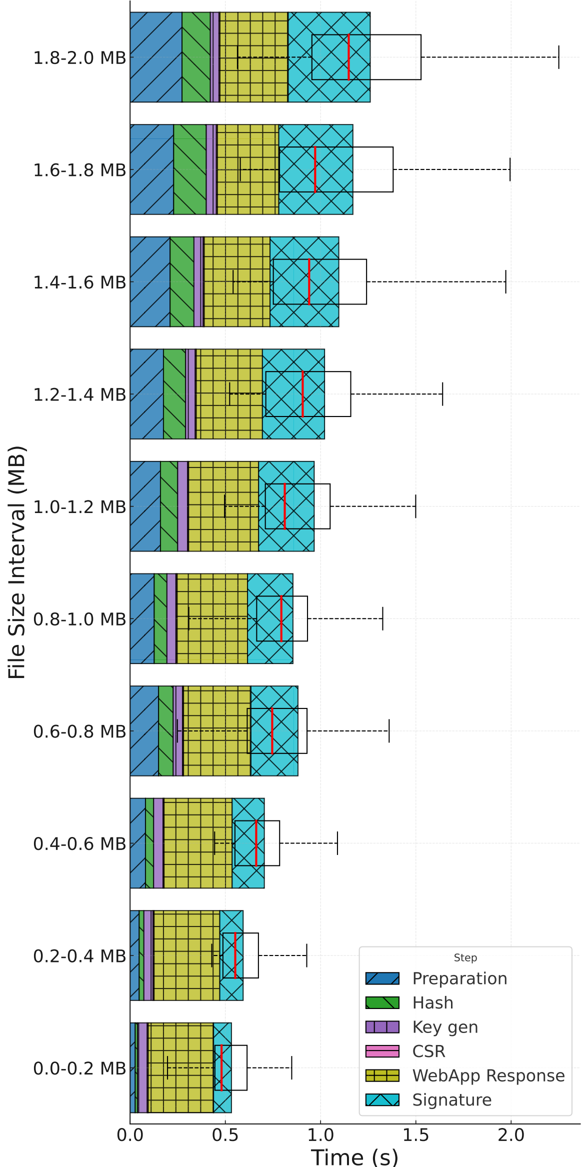

Figure 8. Signature large scale experiment. Stacked bar chart with boxplots showing signing time distribution for PDF documents between 0.2 MB and 2 MB, grouped in 0.2 MB intervals. Each bar represents the average execution time across the signing steps, with boxplots displaying quartiles, whiskers, and variability. The red line indicates the mean total signing time.

of private key exposure by managing key generation and usage locally, with no long-term key storage required.

Efficient Verification: Ensuring document secrecy during verification by performing all operations locally and avoiding the transmission of sensitive metadata, even when fetching external artifacts such as TSLs or CRLs.

User Experience: Simplified certificate management and competitive signing and verification times for real-world document sizes.

Through the adoption of this model, digital signature platforms can mitigate liability, enhance compliance with secrecy regulations, and offer robust secrecy assurances. This approach not only aligns with legal secrecy and data protection mandates but also provides platforms with a competitive edge, fostering trust and user adoption.

Challenges and Future Work

While our model offers substantial security improvements, challenges arise in scenarios involving multiple signers and documents. Future research will prioritize optimizing usability, particularly for documents requiring sequential signatures or batch signing. This includes investigating efficient methods for handling multiple documents within a single signing session, exploring secure out-of-band communication methods, and enhancing browser-based signing capabilities for improved performance. Additionally, a comprehensive analysis of web browser security is crucial to ensure the overall resilience of our client-side cryptographic approach. Lastly, potential vulnerabilities such as man-in-the-middle and denial-of-service attacksl, were considered beyond the scope of this study. Although these threats do not compromise document secrecy, they could affect system availability or operational continuity. In practice, our implementation can mitigates man-in-the-middle risks through authenticated HTTPS communication and short-lived session tokens bound to each certificate request. Future work will formalize these protections and evaluate the model's resistance to such attack scenarios under adversarial conditions.

Declarations

Authors' Contributions

Wellington Fernandes Silvano: Conceptualization, Project administration, Methodology, Data curation, Analysis, and Writing - Original Draft and Final review.
Lucas Mayr: Conceptualization, Writing - Review & Editing, providing technical and scientific support.
Enzo Brum: Software (principal developer), Validation (test implementation), and Writing - Review.
Gabriel Cabral: Software (initial code development), Conceptualization, and technical support.
Frederico Schardong: Writing - Review & Editing, and technical and scientific support.
Ricardo Custódio: Conceptualization, providing technical and scientific support, Resources (creation of the infrastructure and ecosystem), and Writing - Review.
All authors read and approved the final manuscript.

Competing interests

The authors declare that they have no competing interests

Acknowledgements

The author would like to thank the Rede Nacional de Ensino e Pesquisa (RNP), the Operador Nacional do Registro Civil de Pessoas Naturais (ON-RCPN), and Conselho Nacional de Desenvolvimento Científico (CNPq). This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001.

Availability of data and materials

Our source code for the *Web-Platform Secrecy Signer* is available at `https://github.com/wellisilvano/`

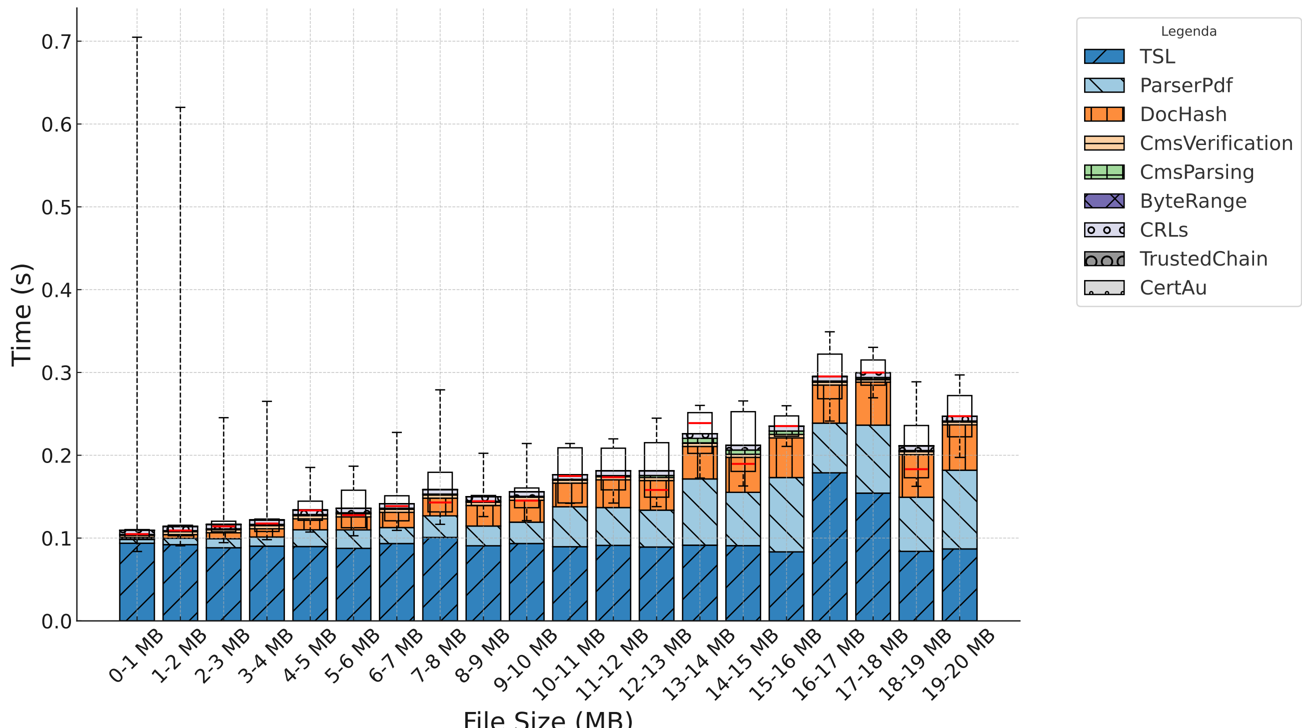

Figure 9. Verification large scale experiment. Stacked bar chart with boxplots showing verification time distribution for PDF files grouped in 1 MB intervals (0-20 MB). Each bar represents the mean time for individual verification steps, while boxplots indicate variability and quartiles. The red line marks the mean total verification time.

`web-secrecy-signature-PKI` or `https://codigos.ufsc.br/100000000343066/web-secrecy-signature`.

The PDFs public library is available at `https://github.com/tpn/pdfs`

References

Aciobăniei, I., Arseni, S.-C., Bureacă, E., and Togan, M. (2024). A comprehensive and privacy-aware approach for remote qualified electronic signatures. *Electronics*, 13(4). DOI: 10.3390/electronics13040757.

Adobe Inc. (2024). Adobe acrobat. Available at:`https://acrobat.adobe.com/us/en/`. Accessed: 2024-08-19.

Ascertia (2018). Signinghub: Architecture and Deployment Guide. Available at:`https://manuals.ascertia.com/SigningHub/8.6/Architecture-Deployment/`. Accessed: 2024-06-08.

Barker, E. and Barker, W. (2018). Recommendation for key management. Part 2: Best Practices for Key Management Organization. Avaialble at:`https://www.semanticscholar.org/paper/Recommendation-for-Key-Management%3A-Part-2-Best-for-Barker-Barker/f9c87f4e147bd40cf2c450d8b495d963fdd5c005`.

Bit4id (2021). Signcloud. Remote digital signature and key management. Available at:`https://www.bit4id.com/wp-content/uploads/2021/12/signcloud_DS_4.0_EN_LQ.pdf`. Accessed: 2024-06-08.

Boeyen, S., Santesson, S., Polk, T., Housley, R., Farrell, S., and Cooper, D. (2008). Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile. (5280):151. DOI: 10.17487/RFC5280.

Boneh, D. and Franklin, M. (2001). Identity-based encryption from the weil pairing. In *Annual international cryptology conference*, pages 213–229. Springer. DOI: 10.1007/3540-446478_13.

Brasil (2018). Lei Geral de Proteção de Dados Pessoais (General Data Protection Law. Lei n° 13.709, de 14 de agosto de 2018. Available at:`https://legislacao.presidencia.gov.br/atos/?tipo=LEI&numero=13709&ano=2018&ato=293QzZ61UeZpWT79e`.

Brazil (1996). Lei de Propriedade Industrial (Industrial Property Law). Lei n° 9.279, de 14 de maio de 1996. Available at:`http://www.planalto.gov.br/ccivil_03/leis/L9279.htm`.

Brazil (2011). Lei de Acesso à Informação (Freedom of Information Law). Lei n° 12,527, de 18 de novembro de 2011. Available at:`http://www.planalto.gov.br/ccivil_03/_ato2011-2014/2011/lei/l12527.htm`.

Brazil, Economy Ministry (2021). Portaria SEDGG/ME n° 2.154, de 23 de fevereiro de 2021. Available at:`https://www.in.gov.br/en/web/dou/-/portaria-sedggme-n-2.154-de-23-de-fevereiro-de-2021-304916270`. Institui normas de gestão de integridade, riscos e controles internos no âmbito da Administração Pública Federal direta, autárquica e fundacional.

CFM (2010). Código de Ética Médica. Resolução CFM n° 1.931/2009. Available at:`http://www.portalmedico.org.br/novocodigo/integra.asp`.

Choi, S.-H., Yun, J., and Park, K.-W. (2017). Doc-

trace: Tracing secret documents in cloud computing via steganographic marking. *IEICE TRANSACTIONS on Information and Systems*, 100(10):2373–2376. DOI: 10.1587/transinf.2016INL0002.

Colomb, Y., White, P., Islam, R., and Alsadoon, A. (2022). Applying zero trust architecture and probability-based authentication to preserve security and privacy of data in the cloud. In *Emerging trends in cybersecurity applications*, pages 137–169. Springer. DOI: 10.1007/978-3-031-09640-2_7.

Cryptomathic (2023). Signer. Freedom to digitally sign documents remotely. Available at:https://www.cryptomathic.com/hubfs/Documents/Product_Sheets/Cryptomathic_Signer_-_Product_Sheet.pdf. Accessed: 2024-06-11.

Digital Bazaar, I. (2010). Node-forge: A native implementation of TLS in JavaScript and Tools to Write Crypto-Based and Network-Heavy web apps. Available at:https://github.com/digitalbazaar/forge. JavaScript library for cryptographic and network tools.

DigitalSign (2023). Signingdesk solution. Available at:https://www.digitalsign.pt/en/pt/signingdesk/. Accessed: 2024-06-08.

Eich, B. (1995). Javascript. Available at:https://developer.mozilla.org/en-US/docs/Web/JavaScript. Programming language for web development.

ETSI (2024). Electronic Signatures and Infrastructures (ESI): PAdES digital signatures; part 1: Building blocks and PAdES baseline signatures. Available at:https://www.etsi.org/deliver/etsi_en/319100_319199/31914201/01.02.01_60/en_31914201v010201p.pdf. Accessed: 2024-08-16.

European Union (2014). Regulation (EU) No 910/2014 of the European Parliament and of the Council of 23 July 2014 on electronic identification and trust services for electronic transactions in the internal market and repealing Directive 1999/93/EC. Available at:https://eur-lex.europa.eu/eli/reg/2014/910/oj.

European Union (2018). General data protection regulation, regulation (eu) 2016/679. Available at:https://gdpr-info.eu/.

Foundation, E. (2024). Ethereum. Available at:https://ethereum.org/en/. Accessed: 2024-08-16.

GlobalSign and Ventures, P. (2014). Pkijs: A public key infrastructure library for javascript. Available at:https://pkijs.org/. JavaScript library for working with X.509 certificates and cryptographic standards.

Goldreich, O. (2001). *Foundations of cryptography: volume 2, basic applications*, volume 2. Cambridge university press. DOI: 10.1017/CBO9780511546891.

Hansen, T. and Eastlake 3rd, D. E. (2011). US Secure Hash Algorithms (SHA and SHA-based HMAC and HKDF). (6234). DOI: 10.17487/RFC6234.

Housley, R. (2009). Cryptographic message syntax (cms). (5652). DOI: 10.17487/rfc5652.

International Organization for Standardization (2008). Iso 32000: Portable document format (pdf). Available at:https://www.iso.org/standard/51502.html. International Standardization Organization.

ISO (2020). ISO 32000-2: Portable document format (PDF) — part 2. Available at:https://www.iso.org/standard/75839.html. International Standardization Organization.

Jacomme, C. and Kremer, S. (2021). An extensive formal analysis of multi-factor authentication protocols. *ACM Transactions on Privacy and Security (TOPS)*, 24(2):1–34. DOI: 10.1145/3440712.

Jonsson, J. and Kaliski, B. (2016). PKCS #1: RSA Cryptography Specifications Version 2.2. RFC 8017. DOI: 10.17487/RFC8017.

Luan, H., Wang, C., Zhou, Z., and Yang, Z. (2015). Cross-access method for team confidential document based on offline key management. *International Journal of Security and Its Applications*, 9(1):97–108. DOI: 10.14257/ijsia.2015.9.1.11.

Mayr, L., Palma, L., Zambonin, G., Silvano, W., and Custódio, R. (2023). Monitoring key pair usage through distributed ledgers and one-time signatures. *Information*, 14(10):523. DOI: 10.3390/info14100523.

Mayr, L., Zambonin, G., Schardong, F., and Custódio, R. (2024). One-time certificates for reliable and secure document signing. *arXiv preprint*. DOI: 10.48550/arXiv.2208.03951.

Moriarty, K., Nystrom, M., Parkinson, S., Rusch, A., and Scott, M. (2014). PKCS#12: Personal information exchange syntax v1.1. PKCS Standard 12, RSA Laboratories. Available at:https://tools.ietf.org/html/rfc7292.

NextSense (2023). Signing suite. Available at:https://nextsense.com/signing-suite.nspx. Accessed: 2024-06-08.

NIST (2020). Zero trust architecture. Technical Report SP 800-207, NIST. Available at:https://doi.org/10.6028/NIST.SP.800-207.

Nystrom, M. and Kaliski, B. (2000). PKCS#10: Certification request syntax specification version 1.7. PKCS Standard 10, RSA Laboratories. Available at:https://datatracker.ietf.org/doc/html/rfc2986.

OAB (2015). Código de Ética e disciplina da OAB, provimento no. 117/2000. Available at:https://www.oab.org.br/Content/pdf/ced/cedoab.pdf.

Perottoni, E. D., Costa, B. P., Müller, F. L., dos Santos Camargo, V., Schardong, F., Silvano, W., Mayr, L., Custódio, R. F., Rocha, L., Lyra, C., *et al.* (2023). Menos certificação digital e mais identidade eletrônica: Icpedu e cafe em um assinador digital inclusivo. In *Anais Estendidos do XXIII Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais*, pages 93–96. SBC. DOI: 10.5753/sbseg_estendido.2023.235947.

Poppler Utils (2024). pdfsig: Verify digital signatures in PDF documents. Available at:https://manpages.ubuntu.com/manpages/jammy/man1/pdfsig.1.html. Accessed: 2024-08-19.

Prabakaran, D. and Ramachandran, S. (2022). Multi-factor authentication for secured financial transactions in cloud environment. *CMC-Computers, Materials & Continua*,

70(1):1781–1798. DOI: 10.32604/cmc.2022.019591.

Remdra (2023). Sign-pdf-lib. Available at:`https://github.com/remdra/sign-pdf-lib`. Accessed: 2024-11-11.

Shannon, C. E. (1949). Communication theory of secrecy systems. *The Bell System Technical Journal*, 28(4):656–715. DOI: 10.1002/j.1538-7305.1949.tb00928.x.

Shatnawi, A., Munson, E. V., and Thao, C. (2017). Maintaining integrity and non-repudiation in secure offline documents. In *Proceedings of the 2017 ACM Symposium on Document Engineering*, pages 59–62. DOI: 10.1145/3103010.3121038.

Stafford, V. (2020). Zero trust architecture. *NIST special publication*, 800:207. DOI: 10.6028/NIST.SP.800-207.

UFSC (2019). Portaria normativa nº 276/2019/gr, de 18 de setembro de 2019. Available at:`https://arquivos.ufsc.br/f/e28396694cc642a88d2e/?dl=1`.

United Kingdom (1989). Official Secrets Act 1989. Available at:`https://www.legislation.gov.uk/ukpga/1989/6/contents`.

United Kingdom (2000). Freedom of Information Act 2000. Available at:`https://www.legislation.gov.uk/ukpga/2000/36/contents`.

United States (1917). Espionage Act of 1917. Available at:`https://www.govinfo.gov/content/pkg/COMPS-1420/pdf/COMPS-1420.pdf`.

Ventures, P. (2013). Asn1js: A pure javascript library for parsing and serializing asn.1 data. Available at:`https://github.com/PeculiarVentures/ASN1.js/`.

W3C (2017). Web cryptography api. Available at:`https://www.w3.org/TR/WebCryptoAPI/`. Accessed: 2024-11-11.