TEE-Receipt: A TEE-based Non-repudiation Framework for Web Applications

Mahmoud Hofny[1⋆], Lianying Zhao[2], Mohammad Mannan[1], and Amr Youssef[1]

[1] Concordia University, Montreal, Canada
mahmoud.hofny@concordia.ca
m.mannan@concordia.ca
youssef@ciise.concordia.ca
[2] Carleton University, Ottawa, Canada
lianyingzhao@cunet.carleton.ca

Abstract. In web applications, transactions are vulnerable to repudiation by service providers seeking to evade legal and financial responsibilities. To safeguard users against such repudiation, verifiable evidence is essential in establishing transaction origin and confirming receipt. This paper introduces an asymmetric non-repudiation framework TEE-Receipt that leverages a Trusted Execution Environment (TEE) to counter potential server transaction repudiation and collect evidence without dependence on a third party. By considering the threat of dishonest server operators (often not considered in the state-of-the-art), we protect server secrets and conduct password-based user authentication and evidence collection all inside the TEE. This approach eliminates the need for a user to own a certified key pair. Also, TEE-Receipt offers a cost-effective deployment, requiring small software changes and a TEE-capable device. A prototype of TEE-Receipt is developed using Intel SGX as the TEE, and tested using WordPress. Performance evaluations demonstrate that TEE-Receipt introduces trivial overhead and provides satisfactory response time from the user's perspective.

Keywords: Web Applications · Non-Repudiation · Digital Signature · TEE.

1 Introduction

The rapid growth of online applications, including online banking, e-commerce, government services, and social media, has been remarkable in recent years because of their user-friendly nature and convenience. Unsurprisingly, conducting online transactions also exposes the users and services to security risks. Aside from the risk caused by traditional illegitimate actors (i.e., attackers), which has been comprehensively studied [50, 49, 22, 23, 33, 43], there exist also cases where

⋆ On leave from Assiut University, Egypt.

legitimate participants of the online transactions can deny their involvement to evade legal or financial responsibilities, which is known as repudiation [12, 36, 16, 20, 10, 15, 14]. Repudiation can have significant and detrimental impacts on users across various domains, such as finance [31, 2], auctions [28], and e-voting systems [51].

Non-repudiation protocols are crucial to safeguard users against such denials [24], as they establish rules for collecting verifiable evidence from communicating parties. This evidence is useful in dispute resolution as it ensures the transaction's origin and confirms its content and receipt.

However, the fact that web applications can be hosted or operated by a third party [3, 21, 47, 26], complicates non-repudiation. What is further worse, even within the trusted party's organization, there could also be *dishonest operators* having access to server secrets (e.g., certificate private keys) or privileged access in the traditional threat model. For instance, a dishonest operator in a financial institution [31] might manipulate transaction records or steal funds. Similarly, in online auctions [28], insiders or auctioneers with privileged access may manipulate bids after the submission completion, leading to unfair advantages for certain participants and undermining the credibility of the auction platform. The same can also happen in e-voting systems [51]. Transactions caused by such dishonest operators will render state-of-the-art non-repudiation protocols ineffective because of the threat model shift.

Our literature review has revealed a multitude of non-repudiation solutions [30, 7, 37, 27, 52, 41, 19, 39, 6, 18], most of which utilize digital signatures as a common approach. However, each of these protocols exhibits certain drawbacks. First, none of these solutions adequately accounts for the scenario involving a dishonest operator on a legitimate server. Second, certain protocols rely on a trusted third party (TTP) for evidence collection and management [7, 37, 27, 52, 41], while others employ a TTP for maintaining users' private keys and generating signatures on their behalf [37]. Relying on a TTP introduces additional multi-party communication overhead and elevates the risk of colluding attacks and potential privacy violations. Another issue pertains to the security of private keys. Certain protocols require users to manage their private keys using local storage [7, 52, 41, 39, 6] or external devices like USB and smart cards [19]. Last, the deployment of specialized devices, such as smart cards [19] or biometric devices [41], for all users increases the solution's cost and complexity.

TLS-N [39] and ROZEN [6] emerged as extensions to TLS, the predominant cryptographic protocol in web technology. These extensions introduced a non-repudiation feature, incorporating additional functionalities such as redacting sensitive data from collected evidence. However, neither TLS-N nor ROZEN addresses the protection issue of server's secrets and users' credentials from dishonest operators because they assume the server side is trusted. They provide non-repudiation for all exchanged TLS records, including potentially unimportant data that becomes part of the non-repudiation evidence, thus requiring additional computation. Furthermore, users are responsible for safeguarding and handling their private keys and managing their TLS certificates.

Trusted Execution Environment (TEE) technologies, like Intel SGX [9] and ARM TrustZone [44], offer a viable solution to address the trust concerns by serving as a substitute for a trusted third party and eliminate dishonest operators impacts with hardware support. TEEs are designed to isolate and safeguard code and data from other software on the same machine, including the operating system and hypervisor. Additionally, many TEEs offer remote attestation, proving authenticity and integrity to a remote party.

Guan et al. proposed an attack [18] using a TEE to break the deniability of protocols like OTR [5]. However, applying this method in the web context does not resolve the issue of protecting users' credentials from dishonest operators and requires porting the entire protocol to the TEE. Additionally, it attests to all exchanged messages, increasing overhead.

This paper presents an asymmetric non-repudiation framework TEE-Receipt for web-based transactions, taking advantage of a TEE on the server. TEE-Receipt does not rely on a TTP for authentication or evidence collection and management. Users do not have to maintain long-term certified key pairs on multiple devices. All the user and server secrets (including user passwords) are handled within the TEE and the evidence collection is conducted in the TEE as well. We show that TEE-Receipt can be deployed with existing web technologies and platforms with manageable efforts.

TEE-Receipt consists of client-side (browser extension) and server-side (running inside a TEE) components. The client component ensures the server's authenticity, verifies that it is running the correct software within a TEE, and establishes a secure communication channel with the TEE before user registration and login. Meanwhile, the server component stores the Cipher-based Message Authentication Code value, generated by the TEE, of the user's password in its database for user authentication purposes. Upon successful user login, the TEE assigns a temporary signing key for the user. Subsequently, the client and the TEE sign the transaction request, providing evidence of its origin and receipt. Our contributions can be summarized as follows.

- We propose a non-repudiation framework TEE-Receipt tailored for web-based transactions using the TEE, effectively eliminating the necessity for a trusted third party, and also able to deal with the dishonest operator threat.
- We implement a prototype of TEE-Receipt[1], comprising a client-side Chrome browser extension and Intel SGX as the server-side TEE.
- We integrate TEE-Receipt with the popular content management platform, WordPress, and conduct a performance evaluation and a security analysis of the integration to assess efficiency and effectiveness in real-world scenarios.

The subsequent sections of this paper are organized as follows: Section 2 presents an overview of the fundamental concepts of non-repudiation and Intel SGX. In Section 3, the system model and essential requirements are outlined to establish a clear foundation for discussions. Built upon this, Section 4 provides a detailed exposition of the design intricacies of the proposed framework.

[1] Available at https://github.com/TEE-Receipt

The practical implementation aspects are covered in Section 5. Subsequently, in Sections 6, 7, and 8, we evaluate TEE-Receipt's deployability, assessing its alignment with the specified security requirements, and its performance, respectively. To enrich the research context, Section 9 offers a review of relevant literature. Finally, Section 10 concludes the work.

2 Background

2.1 Non-repudiation

Non-repudiation is a critical concept that determines whether a particular action has been executed to resolve disputes among parties [30]. By preventing individuals from denying their involvement in actions, such as denying sending or receiving a message, non-repudiation ensures the integrity and authenticity of the communication. The generation and collection of evidence, such as sender and receiver signatures for a transaction, are essential aspects of non-repudiation.

When considering communication between Alice and Bob, an effective non-repudiation protocol should include both Non-Repudiation of Origin (NRO) and Non-Repudiation of Receipt (NRR) [30]. NRO safeguards Bob against Alice's denial of message origination, while NRR protects Alice from Bob's denial of message receipt. Additionally, Non-repudiation of conversation (NRC) [39] encompasses both NRO for all sent messages and NRR for all received messages within the communication considering the ordering of messages. Therefore, NRC fits with diverse applications, such as document submission, public data feeds, and web archiving.

Definition 1. ***(Non-repudiation of origin) [30].*** *A non-repudiation protocol provides proof of origin if it generates evidence of origin intended for Bob, which can be presented to a neutral party (an adjudicator), who can definitively determine whether Alice is the sender of a specific message.*

Definition 2. ***(Non-repudiation of receipt) [30].*** *A non-repudiation protocol guarantees non-repudiation of receipt if it creates evidence of receipt that is intended for Alice and can be presented to an arbitrator who can determine if Bob received the message in question or not.*

Definition 3. ***(Non-repudiation of conversation) [39].*** *Non-repudiation of conversation provides proof of the total order of messages sent and received by a party. Intuitively, NRC specifies the conversation and the party's role in it, from the perspective of its system. The specified party is not able to later deny a claim of having sent and received the message in the conversation or the order of messages within the conversation.*

2.2 Intel-SGX

Intel Software Guard Extensions (SGX) is a popular TEE available in Intel desktop and server CPUs [9]. SGX is a set of CPU extensions that protect the execution of an application and its data from other software on the same platform, including the operating system and hypervisor.

A user-space application can use SGX to establish a hardware-enforced TEE, called an enclave. Importantly, data within an enclave can only be accessed from the code of that enclave. Enclave functions can be called from outside via a predefined set of Enclave Calls (ECALLs) while the enclave can call outside functions through a set of Out Calls (OCALLs). Enclave data is stored in an encrypted form inside the Enclave Page Cache (EPC). Note that EPC is a special region of memory that can be accessed only by the CPU. Before enclave data leaves the CPU (e.g. is written to RAM), it is integrity-protected and encrypted using a CPU-only accessible key [17]. This protects the integrity and confidentiality of the enclave's data against privileged software and an adversary with physical access.

During enclave initialization, the CPU measures the enclave's code and configuration generating the MRENCLAVE value. MRENCLAVE forms the enclave identity and can be used to identify the correct enclave.

SGX comes with the feature of protected data exporting, called sealing, to store the enclave sensitive data outside the enclave securely [1]. The enclave data is sealed by encrypting it using a CPU-protected key that can only be accessed by enclaves running on the same CPU with the same MRENCLAVE value. SGX also supports monotonic counters that prevent rollback attacks on the sealed data.

Last, SGX provides remote attestation [1] so that a remote party ascertains that a message comes from an enclave with precise code and hardware configuration. The enclave provides the remote party with a signed quote including its MRENCLAVE and public key values. The remote party validates this quote by calling Intel Attestation Service (IAS) and then constructs a secure encrypted channel directly to the enclave.

3 System Model and Requirements

3.1 System Model

In web-based applications, the interaction between a user and a server is called web transaction [42]. A simple web transaction can comprise a user request, a web server response, and database access for read or write.

TEE-Receipt aims to enhance the security of web transactions by ensuring the non-repudiation of the request messages sent by the user to the web server (i.e., NRO and NRR). The core functionalities of TEE-Receipt are as follows:

- **Server authentication:** The server's identity and its running code are verified through TEE remote attestation.
- **Secure channel establishment:** The communication channel between the client and the server's TEE is secured via a secure key exchange established via the attestation.

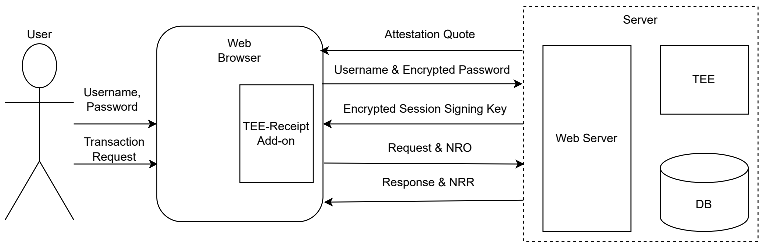

Fig. 1. The system model of TEE-Receipt framework

- **User authentication:** User authenticity is verified by validating the encrypted user credentials inside the TEE to ensure that only authorized users can access the web application and prevent leaking user secrets to dishonest operators. After successful authentication, the server's TEE grants the user a temporary signing key for transaction request integrity and authenticity.
- **Non-repudiation evidence collection:** Each transaction request is signed within the server's TEE to serve as NRR evidence, providing proof of receipt of the transaction request by the server.

3.2 Security and Trust Assumptions

In this paper, the service provider may intend to deny its involvement in transactions. It is semi-trusted where some of its functions are conducted inside a TEE while other functions are accessible by dishonest operators. A dishonest operator has access to and control over server operations, including accessing databases and audit logs. This allows the dishonest operator to manipulate and tamper with databases and audit logs to remove any evidence related to specific transactions.

However, the adversary, a dishonest server operator, is limited by its computational resources, making it unable to break well-implemented cryptographic protocols and guess cryptographic keys. This adversary cannot penetrate or manipulate the client-side software such as the operating system and web browser. The TEE hardware and its attestation service are trusted, and the Public Key Infrastructure (PKI) can be relied on for correctly binding entities to public keys used in TLS. Additionally, all potential side-channel attacks on the TEE are mitigated using appropriate solutions.

3.3 Requirements

TEE-Receipt aims to provide non-repudiation for web-based applications, considering the potential presence of dishonest operators. To achieve this goal, TEE-Receipt must meet certain security requirements. First, the server's secrets must be protected from dishonest operators. Second, user passwords must be protected from dishonest operators when exchanged with the server's TEE in the registration and login process. Third, server identity must be verified to ensure the linkability between the collected NRR evidence and the server's TEE and to avoid the user providing passwords to the wrong TEE. Finally, TEE-Receipt must preserve the integrity of the data by detecting alteration attempts made by any entity, including a server operator. These security requirements are essential to ensure the effectiveness of the non-repudiation guarantee.

4 Design

TEE-Receipt consists of a client-side (i.e., browser extension) and a server-side TEE. Fig. 1 shows the interactions between the client and server and the subsequent sections delve into these interactions and the complete operations in more detail.

4.1 Initialization

As a preliminary step, the TEE initializes the necessary cryptography assets. First, it generates a secret key, denoted as k, used in the Cipher-based Message Authentication Code (CMAC) [13] calculation to protect the user's password as explained in Section 4.3. Moreover, it generates a signing key-pair (sk_s, pk_s) for NRR generation purposes. Importantly, the generated secrets are intended for long-term use and would reside on the server side, hence the TEE takes measures to seal them securely beforehand. For linking TEE's instance to a specific server, the verification key, pk_s, can be embedded in both the server's TLS certificate (as a subject unique identifier [4]) and the TEE's attestation quote.

4.2 Server Authentication and Secure Channel Establishment

Users first verify the legitimacy of the remote server incorporating TEE and establish a secure communication channel with the TEE before engaging in interactions with the server. After the browser completes TLS certificate verification, the server authentication process still requires verifying the identity of the TEE and the software running within it. The TEE's identity can be verified using a non-interactive remote attestation protocol, as detailed in [32]. This non-interactive approach enables users to verify the remote server, establish a secure channel, and submit a request in a single round-trip. Server authentication and secure channel establishment operate as follows. Initially, the TEE generates an Elliptic-Curve Diffie-Hellman (ECDH) key pair (a, A) every time it starts up. When a user starts a session with a server, it requests the TEE quote from the server. The TEE subsequently generates a quote containing its public key A, a unique cryptographic representation of the running code, and pks. After receiving the quote, the user submits it to the remote attestation server for verification. Upon successful verification, the software identity inside the TEE is verified by comparing the provided code representation (from the attestation quote) with the approved TEE-RECEIPT representation.

After passing the above verifications, the user generates an ECDH key pair (b, B) and calculates the session shared key $K_s = b \cdot A$. The session key K_s to ensure the confidentiality of the user's credentials and session signing key, as explained in Section 4.3.

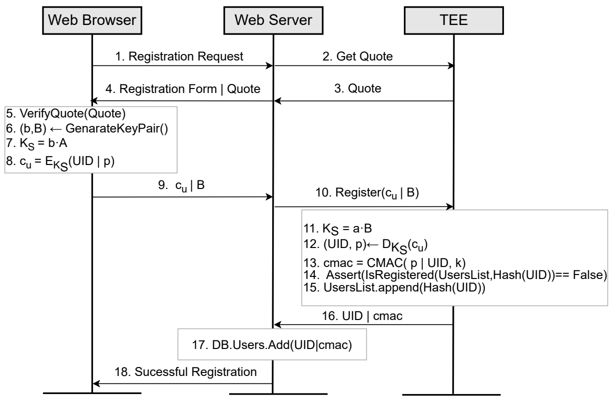
Web Browser
Web Server
TEE
1. Registration Request
2. Get Quote
3. Quote
4. Registration Form | Quote
5. VerifyQuote(Quote)
6. (b,B) ← GanarateKeyPair()
7. K_S = b·A
8. c_u = E_KS(UID | p)
9. c_u | B
10. Register(c_u | B)
11. K_S = a·B
12. (UID, p)← D_KS(c_u)
13. cmac = CMAC(p | UID, k)
14. Assert(IsRegistered(UsersList,Hash(UID))== False)
15. UsersList.append(Hash(UID))
16. UID | cmac
17. DB.Users.Add(UID|cmac)
18. Sucessful Registration

Fig. 2. A sequence diagram of a new user registration process.

4.3 User Authentication

In traditional web applications, the server stores the output of password hashing. Although HTTPS provides confidentiality and integrity between the browser and the server, the password can still be exposed to a potentially dishonest server operator and be susceptible to online/offline guessing. TEE-Receipt addresses this concern by encrypting the user's password and submitting it to the server TEE, storing the password's CMAC value instead of the hash value, and implementing rate-limiting as proposed in SafeKeeper [29]. Furthermore, all the operations of password decryption, CMAC generation and verification, and rate limiting are conducted inside the TEE.

Fig. 2 shows the steps of the new user registration process. The user submits his User IDentification (e.g., email) to the server TEE in encrypted form as $c_u = E_{K_s}(UID|p)$, using the session shared key K_s. Next, the TEE decrypts the UID and password pair, $(UID, p) = D_{K_s}(c_u)$. If the TEE did not find the UID in the registered users' list, it would compute the CMAC of the UID and password pair using the TEE CMAC key k and append the UID's hash to the registered users' list. Computing the CMAC instead of the normal hash makes it more difficult for anyone, including the server operators, to guess the password without the TEE CMAC key, k. Finally, the calculated CMAC value, *cmac*, is stored in the database.

The authentication process involves several steps, as shown in Fig. 3. First, the user encrypts their password using the shared session key, $c_p = E_{K_s}(p)$, and

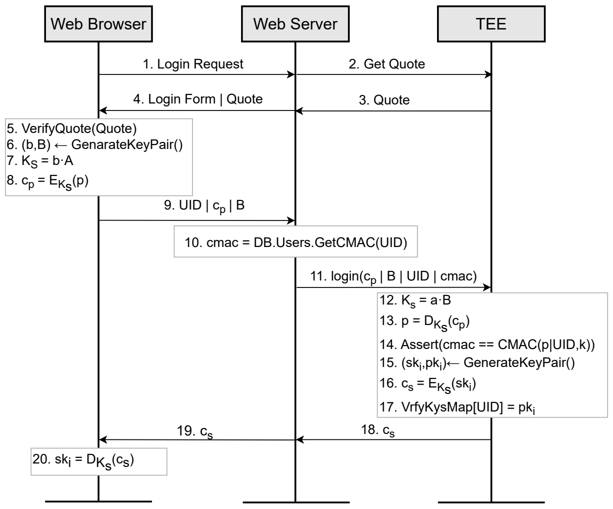
Web Browser
Web Server
TEE
1. Login Request
2. Get Quote
3. Quote
4. Login Form | Quote
5. VerifyQuote(Quote)
6. (b,B) ← GenarateKeyPair()
7. K_S = b·A
8. $c_p = E_{K_S}(p)$
9. UID | c_p | B
10. cmac = DB.Users.GetCMAC(UID)
11. login(c_p | B | UID | cmac)
12. K_s = a·B
13. $p = D_{K_S}(c_p)$
14. Assert(cmac == CMAC(p|UID,k))
15. (sk_i,pk_i)← GenerateKeyPair()
16. $c_s = E_{K_S}(sk_i)$
17. VrfyKysMap[UID] = pk_i
18. c_s
19. c_s
20. $sk_i = D_{K_S}(c_S)$

Fig. 3. A sequence diagram of a user authentication process.

submits their UID, encrypted password, and public key, B, to the server. The server then retrieves the CMAC value, *cmac*, from the database and passes it along with the UID, encrypted password, and B to TEE. TEE generates the shared session key, $K_s = a \cdot B$, decrypts the password, $p = D_{K_s}(c_p)$, and computes the CMAC of the provided password and UID, *cmac* $= CMAC_k(p, UID)$. If the computed *cmac* matches the stored *cmac*, TEE assigns a short-lived signing key pair (sk_i, pk_i) to the authenticated user. The TEE then adds the map of UID and the verification key, pk_i, to the verification keys list of the currently running session, encrypts the signing key, $c_s = E_{K_s}(sk_i)$, and returns the resulting encrypted signing key, c_s, to the server for delivering it to the user. Finally, the user gets the signing key, $sk_i = D_{K_s}(sk_i)$, and keeps it in memory for later use during the running session.

Using only the password to calculate the CMAC's value, as done in SafeKeeper [29], enables a dishonest server operator to compromise the user authentication process. Such an operator can provide the TEE with a specific user's UID, a newly generated public DH key, and valid values of the password and CMAC belonging to a different registered user under the operator's control. With this, the operator can bypass the CMAC validation step and acquire a valid signing key, allowing operators to impersonate the original users and perform actions on their behalf. As a solution for that problem, the CMAC value is calculated using a combination of both the password and UID, instead of just the password. This solution forces the server to have both the password and UID of the target user to compromise the authentication process.

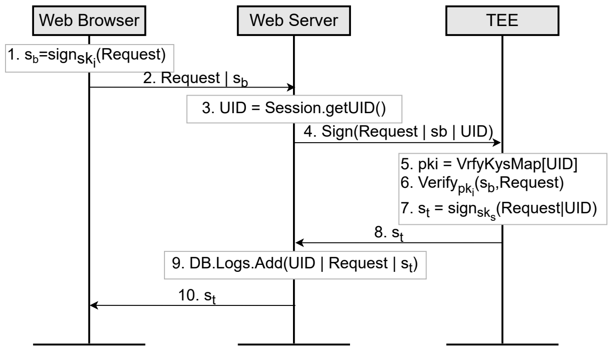
Web Browser
Web Server
TEE
1. s_b=sign$_{sk_i}$(Request)
2. Request | s_b
3. UID = Session.getUID()
4. Sign(Request | sb | UID)
5. pki = VrfyKysMap[UID]
6. Verify$_{pk_i}$(s_b,Request)
7. s_t = sign$_{sk_s}$(Request|UID)
8. s_t
9. DB.Logs.Add(UID | Request | s_t)
10. s_t

Fig. 4. A sequence diagram of a request's content signing process.

4.4 NRO and NRR collection

To ensure asymmetric non-repudiation for a transaction, the user and server's TEE sign the content of the transaction request as NRO and NRR evidence, respectively. Thus, the NRO evidence is proof of the request's integrity and origin authenticity. Conversely, the NRR evidence is utilized in dispute resolution to verify that the server TEE received the transaction request. As shown in Fig. 4, the user first signs the content of the transaction request, $s_b = Sig_{sk_i}(Request)$, and sends the transaction request along with the signature to the server. The server then sends the received request, signature, and the user's UID to the TEE. TEE uses the UID value to obtain the user's verification key, verifies the user's provided signature, $Verify_{pk_i}(Request, s_b)$, and signs the combination of the request and user UID. TEE then returns its signature as NRR to the server. The server adds a new transaction log, including the user request log text, UID, and the TEE signature. Finally, the server forwards the NRR evidence to the user. To preserve the order of NRR evidence, the TEE can maintain a monotonic counter of the received requests and add this counter value to the NRR evidence. Since the temporary signing key pair (sk_i, vk_i), is securely linked to the UID and the UID is tied to the user's password via CMAC calculation, as explained in Section 4.3, a dishonest operator cannot access the user's password. This ensures that only the user who has the correct password can initiate the relevant transaction request.

4.5 Sensitive Data Exclusion

To address the issue of sensitive data contained within the collected NRO and NRR evidence, it is crucial to consider mechanisms that exclude such data from the generated evidence. This sensitive data may include personally identifiable information such as social security numbers, birth dates, place of birth, and financial information like bank account details, credit card numbers, and tax-related information. Therefore, it becomes necessary to develop a mechanism that allows for the exclusion of sensitive data while generating the NRO and NRC evidence.

Both TLS-N [39] and ROZEN [6] have recognized the sensitive data issue and provided the client and server with the capability to redact sensitive data during the generation of non-repudiation evidence. However, the method used in these protocols is complex and lacks flexibility. It involves dividing the TLS records into smaller chunks to exclude sensitive data selectively. This approach introduces challenges, especially when dealing with small-sized sensitive data, as it requires increasing the number of chunks. Consequently, this leads to a higher processing overhead, which can impact the overall efficiency of the system.

TEE-Receipt can be extended to offer a simpler and more flexible approach to exclude sensitive data fields. Initially, the sensitive data field can be identified and marked as such, for example, by adding a class attribute. The client then generates and sends two signatures to the TEE. The first signature encompasses all the fields of the request and is used by the TEE to validate the integrity of the request's content. The second signature, on the other hand, includes only the non-sensitive data fields and is used as NRO evidence. This approach provides a straightforward and elastic solution for excluding sensitive data from the generated evidence, ensuring privacy and security in the non-repudiation process.

5 Implementation

In this section, we introduce the implemented prototype that demonstrates the functionality of TEE-Receipt. The prototype includes a client-side implementation of the framework as a Google Chrome extension and a server-side implementation as an SGX enclave running on the server machine. It's worth noting that any TEE vendor (not necessarily providing attestation-based key derivation) that offers isolated execution, sealed storage, and remote attestation can be used for server-side implementation. However, we use Intel SGX due to its popularity and ease of use [29]. The following sections discuss more details of the client and server sides.

5.1 Server-side SGX Enclave

The server-side SGX enclave provides its functions through 5 ECALLs: initialize, quote, register, login, and sign. The Initialize ECALL function configures the enclave's initial state by setting the necessary cryptographic keys. The Quote ECALL generates an enclave quote. The Register ECALL and Login ECALL handle user registration and authentication, respectively. Finally, the Sign ECALL verifies the user's NRO evidence and generates the server-side NRR evidence.

Initialize ECALL The Initialize ECALL function is responsible for configuring the starting state of the enclave. It sets the values of the CMAC key k, signing key pair (sk_s, pk_s), and session ECDH key pair (a, A). For the first run, Initialize ECALL generates random values for k, the (sk_s, pk_s) pair, and the (a, A) pair. It then seals k and sk_s. In subsequent runs, when k and the (sk_S, pk_S) pair are initialized, Initialize ECALL sets k and sk_s from the sealed inputs and generates the (a, A) pair only. In terms of required memory, the value of k is stored in a 16-byte array structure, while the values of a and sk_S are stored in 32-byte array structures. The values of A and pk_s are stored in two array structures, each with a length of 32 bytes.

Quote ECALL The quote ECALL generates an enclave quote that serves as proof of the authenticity of the server and enclave code during remote attestation. This quote is signed by the enclave and includes the public ECDH session key, A, and MRENCLAVE.

Register ECALL The register ECALL generates the CMAC value for a newly registered user. It accepts the user's encrypted UID and password pair and public ECDH key B as inputs and returns the UID and calculated CMAC. The public key is stored in 64 array structures while the resulting CMAC is stored in a 16-byte array structure. The encrypted pair of UID and password is stored in a byte array with a variable length. The Advanced Encryption Standards (AES) [38] algorithm with the counter mode is used in password decryption and the Rijndael [40] algorithm is used to compute the CMAC value for the combination of the user UID and password.

Login ECALL The login ECALL performs password-based user authentication. It accepts the user's encrypted password, public ECDH key B, UID, and the stored CMAC value (from the server database) as inputs. The login ECALL returns a temporary signing key, in an encrypted form, used for the user's NRO generation purposes. Like the register ECALL, the login ECALL utilizes the AES algorithm with counter mode and the Rijndael algorithm for encryption/decryption and CMAC operations, respectively.

Sign ECALL The sign ECALL is responsible for verifying the user's NRO evidence and generating the server-side NRR evidence. It takes the user request, signature, and session UID as inputs and outputs the enclave signature and the user session verification key. The Elliptic Curve Digital Signature Algorithm (ECDSA) [25] is used for both verifying the user's signature and generating the enclave signature.

5.2 Client Side Chrome Extension

The Chrome extension of the client side contains two elements: a background script and a content script [8]. The background script is responsible for server authentication and secure channel establishment, temporary signing key decryption, and backing up collected NRR evidence in a cloud storage service. On the other hand, the content script is injected into every webpage to perform the operations of password encryption and NRO evidence generation. Further details on the main functions of the extension will be explained in the following sections.

Server Authentication and Secure Channel Establishment The background script validates the enclave quote and identity for server authentication. First, it extracts the enclave quote from the headers of the server HTTP response and checks the validity of that quote using the IAS service API. After successful quote verification, it compares the provided enclave MRENCLAVE with the approved value of the TEE-Receipt enclave. Once the server is authenticated, the background script generates a new ECDH key pair (b, B) and calculates the shared session key, K_{AB}, between the user and the server enclave. At the end, the background script sends the session key, K_{AB}, and the user public key, B, to the content script for later use, such as password encryption.

Password Encryption The content script of the extension handles password encryption for web pages that prompt the user to enter their password, such as registration and login pages. All password fields are encrypted by the shared session key, K_{AB}, using the AES algorithm with counter mode. Eventually, the encrypted password and user public key are represented in hexadecimal string format and submitted to the server.

NRO Evidence Generation Once the user has successfully authenticated and obtained the encrypted session signing key, c_s, the background script decrypts that key using AES with counter mode. Afterward, the signing key, sk_i, is passed to the content script. When the user issues a new transaction request, the content script signs the request's content with sk_i, generating NRO evidence, using the ECDSA algorithm before submission.

6 Deployment

This section first discusses the requirements of TEE-Receipt to be integrated with real systems. Then, it presents more details about TEE-Receipt integration with WordPress.

6.1 Requirements

The integration with the framework's TEE enclave and access to its ECALLs, as detailed in Table 1, requires specific modifications to the user registration, user login, and form processing logic of the current system. In addition, it requires a modification for forwarding the enclave attestation quote to the user. For user registration, the Register ECALL replaces the used password hashing method.

Similarly, the Login ECALL replaces both the password hashing and verification steps in the login logic. Additionally, the signing key, assigned by the enclave, should be returned to the client as an HTTP response header. The form processing logic requires calling the Sign ECALL, adding the transaction log to the database, and returning the NRR evidence to the user in the response message. Last, the attestation quote should be returned by calling the Quote ECALL. Although these adjustments entail some development effort, they remain relatively minor compared with the alternative of constructing an entirely new system.

Table 1. The enclave APIs list.

API	**Description**
Initialize(*sealed_sk*$_s$: unsigned byte*, *sealed_k*: unsigned byte*): void	Initializes the TEE secrets, sk_s and k.
Quote(): unsigned byte[1300]	Generates the TEE attestation quote.
Register(c_p:unsigned byte*, B: unsigned byte[64]): unsigned byte[8], unsigned byte[64]	Assigns the UID and calculates the CMAC value of a new user.
Login(c_p:unsigned byte*, B: unsigned byte[64], UID: unsigned byte[8], *cmac*: unsigned byte[64]): unsigned byte[32])	Checks user credentials and generates a session signing key pair.
Sign(Request: unsigned byte*, S_T: unsigned byte[64], UID: unsigned byte[64], PK_i: unsigned byte[64]&): unsigned byte[64]	Verifies the client's signature and generates the TEE's signature.

6.2 Integration with Wordpress

WordPress [46] stands as a PHP-based content management system renowned for its ability to facilitate the creation of static web pages through a user-friendly, code-free interface, employing a drag-and-drop paradigm, primarily tailored for blogs. However, WordPress extends its functionality to accommodate dynamic page creation and form generation using plugins such as WPForms [48] and Forminator [11]. Normally, non-admin users cannot set their password. However, the "Theme My Login" plugin [35] enables users to set their passwords.

To utilize the functionality of TEE-Receipt in WordPress, we developed a relayer and a PHP module to interact with the enclave APIs. The relayer is an ASP.NET application serving as an intermediary interface for the enclave APIs. It furnishes a set of APIs mirroring the enclave's APIs. These APIs can be accessed via URLs without low-level socket coding. In parallel, the framework's PHP module interfaces with the relayer, facilitating the incorporation of enclave functionality into other PHP scripts. Furthermore, we applied the required modifications to integrate with the enclave APIs, as illustrated in Table 2. The total Lines of Code (LoC) affected by the modifications is 26 which is negligible compared to WordPress's 350k LoCs source code [45].

Table 2. Modifications to WordPress source code.

	Purpose	Details	Function	File	Directory	LoC
1	Forward the enclave attestation quote to the client.	Adding Wordpress action and defining a function that calls the enclave Qoute ECALL and returns the output as HTTP response header.	NA	functions.php	/wordpress/ wp-content/ themes/ twentytwentyone	8
2	Calculate the CMAC and UID for a new user	Calling the Register ECALL using the TEE-Receipt php module instead of calling the HashPassword function of the PasswordHash module.	wp_hash _password	pluggable.php	/wordpress/ wp-includes/	3
3	Calculate and match the CMAC of a user password.	Calling the login ECALL using the TEE-REceipt php module instead of calling the HashPassword function of the PasswordHash module and matching the password.	wp_check _password	pluggable.php	/wordpress/ wp-includes/	7
4	Verify the browser signature and generate the NRR evidence.	Concatenating all form fields, calling the Sign ECALL, and forwarding the NRR evidence to the client.	process	class-process.php	/wordpress/ wp-content/ plugins/ wpforms-lite/ includes/	8
			Total LoC			26

7 Security Analysis

This section evaluates the effectiveness of TEE-Receipt in meeting the requirements stated in Section 3.3.

7.1 Server Secrets Protection

The server secrets, including CMAC and signing keys, are initialized and managed within a TEE. Additionally, sealed copies of these secrets are exported to server storage. As a result, adversaries can only obtain these secrets by breaking the TEE's protection, such as by conducting side-channel attacks or breaking the encryption of the sealed copies. Assuming that all side-channel attacks of SGX are mitigated and the adversary cannot break the well-implemented cryptography used in secret sealing, the adversary has no advantage in obtaining server secrets.

7.2 Server Authentication

Before sending sensitive information, including passwords, users verify the authenticity of the server and the code running within a TEE. To bypass authentication, an adversary must break the enclave quote and identity verification. There are two possible directions the adversary thinks in. The first direction is generating a custom enclave impersonating the genuine enclave. Even though this trial passes the quote verification, it does not pass the enclave identity check. The second direction is forging a valid enclave quote with the same MRENCLAVE as the genuine enclave. This trial is also infeasible as the quote containing the

MRENCLAVE is authenticated via the hardware-secured key of the TEE. Given the formidable level of complexity involved in attempting to guess the TEE's signing key, and the inherent incapacity of adversaries to execute side-channel attacks for key retrieval, any endeavor to replicate the MRENCLAVE and the associated legitimate enclave signature remains implausible.

7.3 Password Protection

Assuming attack-free clients, TEE-Receipt protects users' passwords against dishonest operators by ensuring the confidentiality of the password when exchanged between the user and the TEE and eliminating online and offline attacks against the stored password hash.

Password Confidentiality The potential disclosure of the user's password hinges on the adversary's capacity to execute specific attacks. First, the adversary must orchestrate a Man-in-the-Middle (MITM) attack by impersonating the server's TEE. However, the impersonation of the server's TEE is infeasible, necessitating the presentation of a valid quote with a specific identity, MRENCLAVE —an endeavor rendered implausible as expounded earlier. Second, the adversary would need to successfully guess one of the session keys, encompassing the private keys of both the server's TEE and the client, alongside the session shared key. Yet, this undertaking also proves unattainable. Given the adversary's inability to either convincingly impersonate the server's TEE or procure a session key, the confidentiality of the transmitted data, including the password, remains safeguarded throughout its transmission and processing within the server. Consequently, attackers are prevented from gaining any advantage in acquiring the user's password.

Offline Guessing Attacks Offline guessing attacks are mitigated by employing the CMAC computation for password verification, thereby rendering offline attacks significantly more challenging, even in the case of weak passwords. An adversary attempting such an attack must contend with the requirement of simultaneously guessing both the password and the CMAC key. While the password's vulnerability to being easily guessed is acknowledged, the same cannot be said for the CMAC key, as its cryptographic strength precludes feasible guessing. In addition, it is not feasible to get the CMAC key from the sealed copy, as discussed above in Section 7.1.

Online Guessing Attacks In online guessing attacks, an adversary directly guesses the password and tests it against the operational TEE enclave. In such cases, the adversary is not required to guess the CMAC key and is solely focused on the password. Consequently, the proposed framework introduces a rate-limiting mechanism [29] designed to elongate the time intervals between successive authentication attempts, effectively heightening the complexity of password guessing for malicious actors. As an illustration, assuming an average password length of 20 bits and an established maximum authentication rate of 144 attempts per day, an attacker would, on average, need approximately 10 years to successfully guess the password [29].

7.4 User Authentication

Users are authenticated within the server's TEE based on their passwords. For an adversary to successfully impersonate a user, they first need to acquire the user's password. Assuming that the client side is free of malware, strong passwords are used, and thorough measures are taken to protect the confidentiality and strength of passwords against both offline and online attacks. This greatly reduces the chance of an adversary bypassing the user authentication process.

7.5 Integrity

The framework ensures the integrity of transaction requests by necessitating user signatures before transmission to the server. Moreover, the user's signing key is securely assigned by the TEE and transmitted to the user in encrypted form. To manipulate the content of a user's request, an adversary must first obtain the corresponding signing key. Under the conditions that the client side remains devoid of malware and that the adversary's ability to impersonate the client or the TEE of the server is negated, the adversary is left with two potential avenues to breach request integrity: either guessing the signing key or a session key. However, both of these avenues remain unfeasible for the adversary to exploit. Consequently, any attempt to modify the user's transaction request becomes unattainable for the adversary.

7.6 Non-repudiation

Considering only potential repudiations from the service provider, there are 2 possible repudiations: 1) a service provider denies receiving a legitimate transaction from an authorized user and 2) a service provider denies an illegitimate transaction conducted by a dishonest operator on behalf of a user. There are two scenarios for repudiation of type 2. In the first scenario, a dishonest server operator conducts an illegitimate transaction with a log record. In the other scenario, it may conduct an illegitimate transaction without a log record. The latter scenario is considered an illegitimate transaction without any verification. Given that, the dishonest operator cannot generate a valid signature for an illegitimate transaction without either getting the TEE signing key or impersonating a user and it is infeasible to either get the TEE signing key or impersonate a user, as discussed in Sections 4.3 and 7. Then, the repudiations of type 1 and type 2 scenario 2 are resolved by verifying the TEE's NRR evidence using the user request and UID and the server's TEE verification key, pk_s. TEE-Receipt offers a strong NRR due to the secure protection of the server signing key within the

TEE, as described in Section 7.1. Note that the NRR is deemed strong because of the strong threat model including dishonest operators, while the standard NRO offered by TEE-Receipt is based on the common Dolev-Yao threat model, e.g., not including insider threats.

8 Performance Evaluation

This section presents the performance measurement of TEE-Receipt involving the average processing rates and the latency overhead from the user perspective.

During the conducted experimental test, both the client and server components were deployed within the confines of the same local network. The hardware setup utilized an Intel Core i7-10700 2.90GHz CPU with 16GB of RAM on the Windows 10 operating system. Furthermore, the specific version of the Intel SGX Software Development Kit (SDK) employed for this test was version 2.14.1.

8.1 Processing Rate

The average processing rate, i.e., the number of operations performed per second, provides valuable insights into the system's capacity to handle growing workloads and accommodate the increase of users. It is calculated for the enclave Register, Login, and Sign ECALLs. The average processing rates of Register, Login, and Sign ECALLs are 2.6k, 1.5k, and 1.2k operations per second, respectively. It is worth mentioning that the average processing rate is computed over 1M iterations for each ECALL of the three ECALLS. Moreover, the Sign ECALL is evaluated with a request size of 32 bytes.

8.2 Latency

Considering user experience, it is crucial to consider the latency overhead introduced by the TEE-Receipt extensions on both the client and server sides when integrated into real systems. This section presents the latency overhead of TEE-Receipt integrated with WordPress for each of Enclave's quote verification, user registration, login, and form submission.

The Enclave's quote verification process through the Intel Attestation Service (IAS) takes 140 ms on average. This delay does not negatively impact the user's browsing experience since the verification process runs concurrently with page loading and runs when a user registers and logs in.

Table 3 presents the average processing overhead of TEE-Receipt for user registration, user login, and form submission on WordPress. The user registration overhead includes password encryption on the client side and TEE-Receipt's registration ECALL on the server side. Similarly, the user login overhead comprises password encryption on the client side and the utilization of TEE-Receipt's login ECALL. The form submission overhead involves NRO evidence generation on the client side and NRR evidence generation via the TEE-Receipt Sign ECALL on the server side. TEE-Receipt adds 5 ms, 10 ms, and 17 to 27 ms overheads to the user registration, login, and form submission (with 0.1k to 2k Bytes content) processes, respectively. Practically, this overhead is small compared to the average response time for web applications [34]. In conclusion, TEE-Receipt operations do not adversely impact the user experience.

Table 3. The average processing overhead in ms of TEE-Receipt utilized in WordPress over 500K iterations.

	Password Length/ Content Size	**Average Overhead (ms)**
User Registration	12 Bytes Password	5ms
User Login	12 Bytes Password	10ms
Form Submission	0.1k to 2K Bytes Content	17 to 27 ms

Table 4. Related Work Comparison.

Work	**Trustworthiness**	**Evidence Type**	**Sensitive Data Exclusion**	**TTP**	**Communications**	**Hardware**
TEE-Receipt	Server is semi-trusted	NRO and strong NRR	No	No TTP	2 messages per a transaction	A TEE for the server
Coffey [7]	Sender and receiver not trusted while TTP is trusted	NRO and NRR	No	Inline TTP	11 messages per a transaction	None
Resondry [37]	Sender and receiver not trusted while TTP is trusted	NRO and NRR	No	Inline TTP	6 messages per a transaction	None
Schiavone [41]	Server is trusted	NRO and NRR	No	Online TTP	2 messages to open and renew a session and 2 messages per a transaction	A biometric reader for each user
Hiltgen [19]	Server is trusted	NRO and NRR	No	No TTP	4 messages per a transaction	A smart card and reader for each user
TLS-N [39]	Server is trusted	NRC	Yes	No TTP	2 messages per a transaction	None
ROZEN [6]	Server is trusted	NRC	Yes	No TTP	2 messages per a transaction	None
Guann [18]	Server is semi-trusted (web context)	NRC	No	No TTP	2 messages per a transaction	TEE for a server (web context)

9 Related Work

Various works [30, 7, 37, 27, 52, 41, 19, 39, 6] have been presented to establish reliable and efficient non-repudiation protocols. Some of these protocols [7, 37, 27, 52, 41] rely on TTP to ensure non-repudiation, while others do not [19, 39, 6, 18]. The TTP-based protocols can be categorized into two groups based on how the TTP is involved in the communication. The first group employs inline TTPs in which the TTP is involved in each transaction between sender and recipient parties. An early example of this group is Coffey's fair non-repudiation protocol based on inline TTP [7], where a non-repudiation server (NRS) serves as the inline TTP. The NRS receives a sender's message and signature as proof of origin, then forwards a commitment of the message to the recipient, along with a signed acknowledgment. The sender's message and the recipient's acknowledgment include a trusted timestamp obtained from a time-stamping authority (TSA) to prevent replays. Similarly, subsequent works, such as the fair non-repudiation framework for web-based transactions using inline TTP presented by [37], have expanded upon these concepts. However, the client does not have to hold a private key where both key maintaining and cryptography operations including digital signature are delegated to the inline TTP.

Although inline TTP-based protocols offer robust non-repudiation considering untrustworthy sender and receiver, they can experience delays due to the extra communication overhead resulting from the three-way transmission between sender, TTP, and recipient parties. Furtheremore, the risk of potential privacy violation. To address this, Schiavone proposed a non-repudiation protocol based on an online TTP [41] in which the TTP is involved once in a session. The non-repudiation evidence is generated only when the session is initiated or extended. Unfortunately, this solution does not provide non-repudiation for each request. Consequently, a dishonest operator can pass illegitimate requests as it comes from authorized user during the session.

To eliminate the need for a TTP and its associated overhead, authors of [19] proposed a non-repudiation system for online banking using a JavaCard. A secure JavaApplet integrates with the browser, allowing the JavaCard to sign user-initiated transaction requests. The bank server responds with an encrypted, signed transaction, and user approval on the reader finalizes the process. This ensures robust non-repudiation in online banking. However, this solution assumes that the server is trusted and does not address the issue of dishonest operators.

Moreover, TLS-N [39] and ROZEN [6] were introduced as extensions to the TLS protocol to provide non-repudiation capabilities. These extensions enable non-repudiation of conversation and offer the flexibility to exclude sensitive data from the generated NRC evidence. However, both TLS-N and ROZEN assume that the server is trusted and do not address the protection issue of the server's secrets and users' credentials from dishonest operators. They rely on PKI for client and server authentication, necessitating clients to maintain their long-term private keys. Moreover, the NRC evidence generation for all exchanged messages between the client and server results in more computation overhead and requires substantial storage to back up this evidence.

Using TEE, the authors of [18] presented an attack on deniable communications [5]. The proposed attack enables parties to generate non-repudiable transcripts of the communications using the remote attestation feature of TEE. In this attack, the TEE attests to all messages exchanged between parties to enable a third-party verifier to validate these transcripts. It is more suitable for peer-to-peer messaging applications. However, in web applications, the attack method does not address the protection issue of users' credentials from dishonest operators since the data store is still exposed to operator access. Additionally, it requires the full messaging protocol to run inside the TEE.

In summary, our TEE-based TEE-Receipt represents a suitable solution for web-based transactions compared to the mentioned protocols because it achieves non-repudiation without dependence on a TTP with a small communication overhead, just two messages per transaction, as Table 4 shows. It addresses the potential presence of dishonest operators on a legitimate server and eliminates the need for long-term certified key pairs for users. Additionally, it can be extended to sensitive data exclusion from the collected evidence without complex computation, and system usability is not impacted.

10 Conclusion

The proposed framework, TEE-Receipt, leveraging a Trusted Execution Environment (TEE), effectively protects users from potential server repudiations. It ensures the exchange of Non-Repudiation of Origin (NRO) and Non-Repudiation of Receipt (NRR) evidence upon completion of a transaction without relying on a TTP, with a threat model shift considering dishonest operators on the server. It eliminates the need for long-term certified key pairs for users. Sensitive data can be excluded easily from the collected evidence without complex computation. Additionally, it introduces small changes to the system and maintains usability. Performance evaluations indicate that the framework does not adversely impact system latency, and from the user's perspective, the response time remains satisfactory. Overall, the proposed TEE-based framework offers a secure and efficient solution to address non-repudiation requirements in web-based transactions, enhancing the security and trustworthiness of the system.

References

1. Anati, I., Gueron, S., Johnson, S., Scarlata, V.: Innovative technology for cpu based attestation and sealing. In: Proceedings of the 2nd international workshop on hardware and architectural support for security and privacy. vol. 13. ACM New York, NY, USA (2013)
2. Barboza, D.: Online brokers fined millions in fraud case (2003), http://www.nytimes.com/2003/01/15/business/online-brokers-finedmillions-in-fraud-case.html
3. Bluehost: Web hosting (2024), https://www.bluehost.com/hosting/shared
4. Boeyen, S., Santesson, S., Polk, T., Housley, R., Farrell, S., Cooper, D.: Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile. RFC 5280 (May 2008). https://doi.org/10.17487/RFC5280, https://www.rfc-editor.org/info/rfc5280
5. Borisov, N., Goldberg, I., Brewer, E.: Off-the-record communication, or, why not to use pgp. In: Proceedings of the 2004 ACM Workshop on Privacy in the Electronic Society. p. 77–84. WPES '04, Association for Computing Machinery, New York, NY, USA (2004). https://doi.org/10.1145/1029179.1029200, https://doi.org/10.1145/1029179.1029200
6. Capkun, S., Ozturk, E., Tsudik, G., Wüst, K.: Rosen: Robust and selective non-repudiation (for tls). In: Proceedings of the 2021 on Cloud Computing Security Workshop. pp. 97–109 (2021)
7. Coffey, T., Saidha, P.: Non-repudiation with mandatory proof of receipt. ACM SIGCOMM Computer Communication Review **26**(1), 6–17 (1996)

8. Corporation, G.: Chrome extension development: Get started (2022), https://developer.chrome.com/docs/extensions/mv3/getstarted/
9. Corporation, I.: Intel software guard extensions (intel sgx) (2022), https://www.intel.com/content/www/us/en/developer/tools/software-guard-extensions/overview.html
10. for Cyber Security, C.C.: Security considerations when developing and managing your website (itsap.60.005) (2021), https://www.cyber.gc.ca/en/guidance/security-considerations-when-developing-and-managing-your-website-itsap60005
11. DEV, W.: Forminator (2023), https://wordpress.org/plugins/forminator/
12. Dietrich, C., Krombholz, K., Borgolte, K., Fiebig, T.: Investigating system operators' perspective on security misconfigurations. In: Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security. pp. 1272–1289 (2018)
13. Dworkin, M.J.: Sp 800-38b. recommendation for block cipher modes of operation: The cmac mode for authentication (2005)
14. Fortinet: 7 common web security threats for an enterprise (2023), https://www.fortinet.com/resources/cyberglossary/web-security-threats
15. Fortinet: What is an insider threat? (2023), https://www.fortinet.com/resources/cyberglossary/insider-threats
16. Greitzer, F.L.: Insider threats: It's the human, stupid! In: Proceedings of the Northwest Cybersecurity Symposium. pp. 1–8 (2019)
17. Gueron, S.: Memory encryption for general-purpose processors. IEEE Security & Privacy **14**(6), 54–62 (2016)
18. Gunn, L., Parra, R.V., Asokan, N.: Circumventing cryptographic deniability with remote attestation. In: Privacy Enhancing Technologies Symposium. pp. 350–369. De Gruyter (2019)
19. Hiltgen, A., Kramp, T., Weigold, T.: Secure internet banking authentication. IEEE security & privacy **4**(2), 21–29 (2006)
20. Homoliak, I., Toffalini, F., Guarnizo, J., Elovici, Y., Ochoa, M.: Insight into insiders and it: A survey of insider threat taxonomies, analysis, modeling, and countermeasures. ACM Computing Surveys (CSUR) **52**(2), 1–40 (2019)
21. Hostinger: Web hosting (2024), https://www.hostinger.com/web-hosting
22. Jahanshahi, R., Azad, B.A., Nikiforakis, N., Egele, M.: Minimalist: Semi-automated debloating of PHP web applications through static analysis. In: 32nd USENIX Security Symposium (USENIX Security 23). pp. 5557–5573. USENIX Association, Anaheim, CA (Aug 2023), https://www.usenix.org/conference/usenixsecurity23/presentation/jahanshahi
23. Jang, D., Jhala, R., Lerner, S., Shacham, H.: An empirical study of privacy-violating information flows in javascript web applications. In: Proceedings of the 17th ACM Conference on Computer and Communications Security. p. 270–283. CCS '10, Association for Computing Machinery, New York, NY, USA (2010). https://doi.org/10.1145/1866307.1866339, https://doi.org/10.1145/1866307.1866339
24. Jesus, V.: Towards an accountable web of personal information: The web-of-receipts. IEEE Access **8**, 25383–25394 (2020). https://doi.org/10.1109/ACCESS.2020.2970270
25. Johnson, D., Menezes, A., Vanstone, S.: The elliptic curve digital signature algorithm (ecdsa). International journal of information security **1**, 36–63 (2001)
26. Jotform: Jotform official website (2024), https://www.jotform.com/

27. Kamel, M., Boudaoud, K., Resondry, S., Riveill, M.: A low-energy consuming and user-centric security management architecture adapted to mobile environments. In: 12th IFIP/IEEE International Symposium on Integrated Network Management (IM 2011) and Workshops. pp. 722–725. IEEE (2011)
28. Kammueller, F., Kerber, M., Probst, C.W.: Towards formal analysis of insider threats for auctions. In: Proceedings of the 8th ACM CCS International Workshop on Managing Insider Security Threats. pp. 23–34 (2016)
29. Krawiecka, K., Kurnikov, A., Paverd, A., Mannan, M., Asokan, N.: Safekeeper: Protecting web passwords using trusted execution environments. In: Proceedings of the 2018 World Wide Web Conference. pp. 349–358 (2018)
30. Kremer, S., Markowitch, O., Zhou, J.: An intensive survey of fair non-repudiation protocols. Computer communications **25**(17), 1606–1621 (2002)
31. Kul, G., Upadhyaya, S.: A preliminary cyber ontology for insider threats in the financial sector. In: Proceedings of the 7th ACM CCS International Workshop on Managing Insider Security Threats. pp. 75–78 (2015)
32. Kurnikov, A., Paverd, A., Mannan, M., Asokan, N.: Keys in the clouds: auditable multi-device access to cryptographic credentials. In: Proceedings of the 13th International Conference on Availability, Reliability and Security. pp. 1–10 (2018)
33. Lekies, S., Kotowicz, K., Groß, S., Vela Nava, E.A., Johns, M.: Code-reuse attacks for the web: Breaking cross-site scripting mitigations via script gadgets. In: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security. p. 1709–1723. CCS '17, Association for Computing Machinery, New York, NY, USA (2017). https://doi.org/10.1145/3133956.3134091, https://doi.org/10.1145/3133956.3134091
34. LittleData: What is the average server response time? (2023), https://lp.littledata.io/average/server-response-time
35. Login, T.M.: Theme my login (2023), https://wordpress.org/plugins/theme-my-login/
36. Naderi-Afooshteh, A., Kwon, Y., Nguyen-Tuong, A., Razmjoo-Qalaei, A., Zamiri-Gourabi, M.R., Davidson, J.W.: Malmax: Multi-aspect execution for automated dynamic web server malware analysis. In: Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security. pp. 1849–1866 (2019)
37. Resondry, S., Boudaoud, K., Kamel, M., Bertrand, Y., Riveill, M.: An alternative version of https to provide non-repudiation security property. In: 2014 International Wireless Communications and Mobile Computing Conference (IWCMC). pp. 536–541. IEEE (2014)
38. Rijmen, V., Daemen, J.: Advanced encryption standard. Proceedings of federal information processing standards publications, national institute of standards and technology **19**, 22 (2001)
39. Ritzdorf, H., Wüst, K., Gervais, A., Felley, G., Capkun, S.: Tls-n: Non-repudiation over tls enabling-ubiquitous content signing for disintermediation. Cryptology ePrint Archive (2017)
40. Sanchez-Avila, C., Sanchez-Reillol, R.: The rijndael block cipher (aes proposal): a comparison with des. In: Proceedings IEEE 35th Annual 2001 international carnahan conference on security technology (Cat. No. 01CH37186). pp. 229–234. IEEE (2001)
41. Schiavone, E., Ceccarelli, A., Bondavalli, A.: Continuous biometric verification for non-repudiation of remote services. In: Proceedings of the 12th International Conference on Availability, Reliability and Security. pp. 1–10 (2017)
42. Schuldt, H.: Web Transactions, pp. 3523–3524. Springer US, Boston, MA (2009). https://doi.org/10.1007/978-0-387-39940-9_731

43. Shan, S., Bhagoji, A.N., Zheng, H., Zhao, B.Y.: Patch-based defenses against web fingerprinting attacks. In: Proceedings of the 14th ACM Workshop on Artificial Intelligence and Security. p. 97–109. AISec '21, Association for Computing Machinery, New York, NY, USA (2021). https://doi.org/10.1145/3474369.3486875, https://doi.org/10.1145/3474369.3486875
44. Technologies, A.: Trustzone for cortex-a (2022), https://www.arm.com/technologies/trustzone-for-cortex-a
45. Wordpress: Projects (2023), https://wordpressfoundation.org
46. WordPress: Wordpress official website (2023), https://wordpress.org
47. WordPress: Hosting (2024), https://wordpress.org/hosting/
48. WPForms: Wpforms plugin page (2023), https://wordpress.org/plugins/wpforms-lite/
49. Yang, Z., Allen, J., Landen, M., Perdisci, R., Lee, W.: TRIDENT: Towards detecting and mitigating web-based social engineering attacks. In: 32nd USENIX Security Symposium (USENIX Security 23). pp. 6701–6718. USENIX Association, Anaheim, CA (Aug 2023), https://www.usenix.org/conference/usenixsecurity23/presentation/yang-zheng
50. Yao, M., Fuller, J., Kasturi, R.P., Agarwal, S., Sikder, A.K., Saltaformaggio, B.: Hiding in plain sight: An empirical study of web application abuse in malware. In: 32nd USENIX Security Symposium (USENIX Security 23). pp. 6115–6132. USENIX Association, Anaheim, CA (Aug 2023), https://www.usenix.org/conference/usenixsecurity23/presentation/yao-mingxuan
51. Yasinsac, A.: Insider threats to voting systems. In: Proceedings of the 2010 Workshop on Governance of Technology, Information and Policies. pp. 1–8 (2010)
52. Zhang, N., Shi, Q.: Achieving non-repudiation of receipt. The Computer Journal **39**(10), 844–853 (1996)