expandZK: Securing Web2 Data Verification in Web3 Ecosystems with Zero-Knowledge TLS

January 4, 2025

Abstract

This paper introduces expandZK, a protocol designed to address the challenge of verifying Web2 data within Web3 ecosystems with a strong focus on security. By integrating **Zero-Knowledge Proofs (ZKPs)** into the **Transport Layer Security (TLS)** framework, expandZK ensures that sensitive data originating from Web2 environments can be securely verified in Web3 applications without exposing the underlying data. expandZK distinguishes between public data and **Personally Identifiable Information (PII)** and provides robust, cryptographic guarantees for secure data transmission and verification across decentralized platforms. Applications such as **decentralized finance (DeFi)**, digital identity, and cross-network data validation benefit from expandZK's ability to securely integrate Web2 data into Web3 infrastructures.

1 Introduction

1.1 The Web2-Web3 Data Security Challenge

As the internet evolves toward decentralized ecosystems, the need for secure and reliable methods of verifying Web2 data within Web3 platforms is becoming more urgent. In traditional Web2 environments, **Transport Layer Security (TLS)** has been the gold standard for securing communication channels, ensuring that data remains protected during transmission. However, TLS does not inherently provide mechanisms to prove the integrity or authenticity of data once it transitions from Web2 to Web3. This limitation presents a significant security challenge for decentralized applications (dApps), particularly those that rely on sensitive or mission-critical data such as financial records, identity credentials, or transaction details.

Web3, which operates on decentralized infrastructure, introduces an additional layer of complexity. Unlike Web2, where data is largely managed and verified by centralized authorities, Web3 relies on decentralized systems like blockchains, which require cryptographic guarantees to ensure data integrity and trustworthiness. In this context, ensuring that data originating from Web2 sources can be securely verified in Web3 environments is a key challenge for maintaining security in decentralized applications.

Without a reliable method to verify Web2 data in Web3 platforms, attackers could manipulate or forge information, leading to severe consequences such as fraudulent transactions, identity theft, or unauthorized access to secure systems. This creates a demand for a robust solution that can bridge the gap between Web2's reliance on TLS for secure data transmission and Web3's need for verifiable, immutable data integrity.

1.2 expandZK: Securing Data Verification Across Web2 and Web3

expandZK is designed to address these security concerns by extending the capabilities of **zkTLS**, a protocol that combines the encryption strength of TLS with the cryptographic guarantees of **Zero-Knowledge Proofs (ZKPs)**. While zkTLS focuses on securing communication between Web2 and Web3, expandZK specifically targets the secure verification of both public data and **Personally Identifiable Information**

(PII). By enabling Web3 platforms to securely validate Web2 data through cryptographic proofs, expandZK ensures that sensitive data can be securely verified without exposing it to potential manipulation.

At the heart of expandZK's security model is the integration of Zero-Knowledge Proofs (ZKPs), which allow a user (or prover) to demonstrate the validity of a claim without revealing the underlying data. This method provides a critical layer of security for decentralized applications that need to confirm the authenticity of Web2 data. For example, in decentralized finance (DeFi) applications, expandZK can enable users to prove their creditworthiness or transaction history without revealing sensitive financial details. Similarly, in identity verification systems, expandZK allows secure verification of credentials such as age or citizenship, ensuring that only the relevant data points are exposed for verification.

The expandZK protocol involves three main actors:

- **Web2 Server (Data Provider)**: The entity responsible for serving the data using standard TLS protocols.
- **User (Prover)**: The individual or system that generates zero-knowledge proofs to confirm the authenticity of Web2 data.
- **Web3 (Verifier)**: The decentralized system, such as a smart contract, that verifies the ZKP to confirm the authenticity and integrity of the data.
- **Proxy (Verifier)**: They proxy server for forwarding messages between the web2 server and the user.

expandZK's secure framework ensures that Web3 systems can trust Web2 data sources by allowing decentralized applications to confirm the authenticity and integrity of the data, without needing direct access to the data itself. This secure verification mechanism plays a pivotal role in enabling a wide range of use cases where data integrity is essential, such as cross-platform identity verification, financial auditing, and secure data portability between Web2 and Web3 platforms.

1.3 Securing Public Data and PII with expandZK

One of the key innovations of expandZK is its ability to differentiate between public data and **Personally Identifiable Information (PII)**. While public data, such as user activity or publicly available records, can be more easily verified, PII requires stricter security measures due to its sensitive nature. expandZK introduces a flexible protocol that handles both types of data with appropriate security guarantees, ensuring that public data can be verified efficiently, while PII is secured with more strict cryptographic techniques.

For public data, expandZK optimizes the proof generation process, enabling decentralized applications to verify the authenticity and integrity of these data with minimal computational overhead. This is particularly useful for decentralized content platforms, social networks, or public blockchains where public user data needs to be verified for authenticity without introducing significant security risks.

On the other hand, expandZK introduces advanced mechanisms for handling PII, ensuring that sensitive information such as identity credentials, financial records, or medical data is cryptographically protected throughout the verification process. By utilizing ZKPs, expandZK enables Web3 systems to verify critical attributes of PII (e.g., verifying a user's age without revealing their exact date of birth) without compromising the security of the underlying data.

This dual approach to securing both public data and PII makes expandZK a versatile protocol for a wide range of Web3 applications, where secure data verification is essential for maintaining trust and integrity within decentralized ecosystems.

1.4 Applications and Relevance of expandZK in Web3

The need for secure data verification is rapidly growing as more applications transition to decentralized infrastructures. expandZK is especially relevant in sectors such as:

- **Decentralized Finance (DeFi)**: expandZK can securely verify financial data such as credit history, income or proof of transactions, enabling secure lending, credit scoring, or investment processes in decentralized financial systems.
- **Digital Identity Verification**: expandZK allows for the secure verification of identity credentials, ensuring that only the necessary information is revealed to the verifier, while other sensitive information remains protected.
- **Cross-Platform Data Portability**: expandZK enables the secure transfer and verification of data from centralized Web2 platforms to decentralized Web3 systems, ensuring that data integrity is maintained throughout the process.

By offering a secure framework for verifying Web2 data in Web3 environments, expandZK ensures that decentralized applications can trust the integrity of the data they rely on, enabling secure integration of Web2 and Web3 ecosystems.

2 Protocol Overview

2.1 Web Server, User, and Verifier Relationship

In the expandZK framework, the key participants and their roles are as follows. Figure 1 provides a high-level overview of the interactions between these entities:

- **Web Server (Web2)**: The traditional data provider using **TLS** to securely deliver content.
- **User (Prover)**: The individual accessing the data and generating **Zero-Knowledge Proofs (ZKPs)** to attest to the data's authenticity.
- **Verifier (Web3)**: A Web3 entity (e.g., a smart contract) responsible for verifying the **ZKP**, ensuring the data's validity without exposing its full content.
- **Verifier (Proxy)**: A proxy server that facilitates message forwarding while attesting to the data's origin and integrity.

Figure 1 illustrates the high-level structure of these relationships, showing the flow of data, the generation of proofs, and the verification process. This diagram emphasizes how each component collaborates to establish secure and verifiable communication.

2.2 TLS and ZKP Interaction in expandZK

expandZK allows for seamless interaction between the **TLS** protocol and **Zero-Knowledge Proofs**. When the user accesses data over a **TLS** session, they generate a **ZKP** that can be verified by the **Web3** verifier. The **TLS** encryption ensures the confidentiality of data during transmission, while the **ZKP** ensures the integrity and authenticity of the data, even after the session ends.

- **Three-Party Handshake**: The client, verifier (web3) and the server participate in a **three-party handshake** to establish shared session keys, which are shared secret between the client and verifier (web3). This prevents forgery by ensuring that the verifier holds part of the cryptographic key.
- **Proof Generation**: The client generates **ZKPs** about the data exchanged, which are then verified without revealing the actual data.
- **Proxy-Assisted Attestation**: Utilizing verifier (proxy), the proxy model of expandZK enables proxies to attest to encrypted TLS data exchanged between the web server and the user.

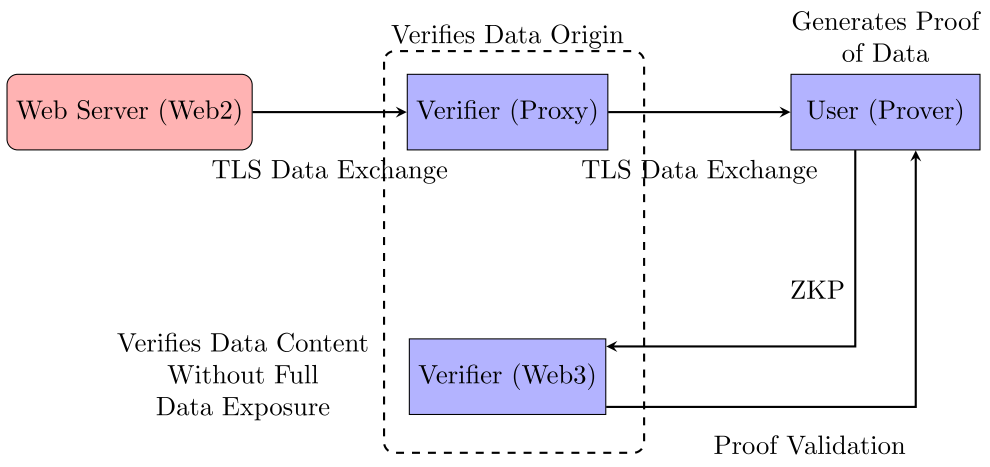

Figure 1: High-Level Structure of Relationships Between Web Server, User, and Verifier

3 Innovations in expandZK

3.1 Efficient Zero-Knowledge Proof Generation with Proxy Integration

Zero-Knowledge Proofs (ZKPs) have traditionally been associated with significant computational and networking overhead. In expandZK, we introduce a novel integration of a proxy server to optimize efficiency and enhance security. The proxy acts as both a facilitator of secure communication and an attester of data origin, streamlining the entire workflow. Figure 2 illustrates this enhanced protocol flow, highlighting the proxy's critical role.

- **Proving MAC Integrity with Proxy Attestation**: Instead of proving the integrity of the entire **TLS** session, expandZK focuses on verifying the integrity of the **MAC** (Message Authentication Code). The proxy, acting as a verifier, provides an attestation that confirms the origin and integrity of the data exchanged, allowing the system to bypass computationally expensive full-session proofs.
- **Precomputation Techniques with Proxy Optimization**: For **AES-GCM**, expandZK leverages precomputation to accelerate polynomial evaluations in **MAC** generation. The proxy further optimizes this process by caching intermediate results and facilitating efficient data forwarding.
- **Proxy-Assisted ZKP Generation and Validation**: The proxy not only forwards encrypted messages but also aids in generating attestations for the data. These attestations, combined with the **ZKP** generated by the user, are validated by the Web3 verifier, ensuring trust without exposing the full data.

The integration of the proxy enables expandZK to achieve a balance between computational efficiency and robust security. Figure 2 provides a high-level overview of the workflow, from the initiation of the **TLS** session to the proxy-assisted generation and validation of the **ZKP**. This architecture represents a significant advancement in minimizing resource usage while maintaining verifiable trust.

3.2 Selective Opening with Context Integrity

expandZK introduces the concept of **Selective Opening**, allowing the user to reveal only specific parts of a **TLS** session, rather than the entire session. This is particularly important in applications where only part of the information is relevant for verification.

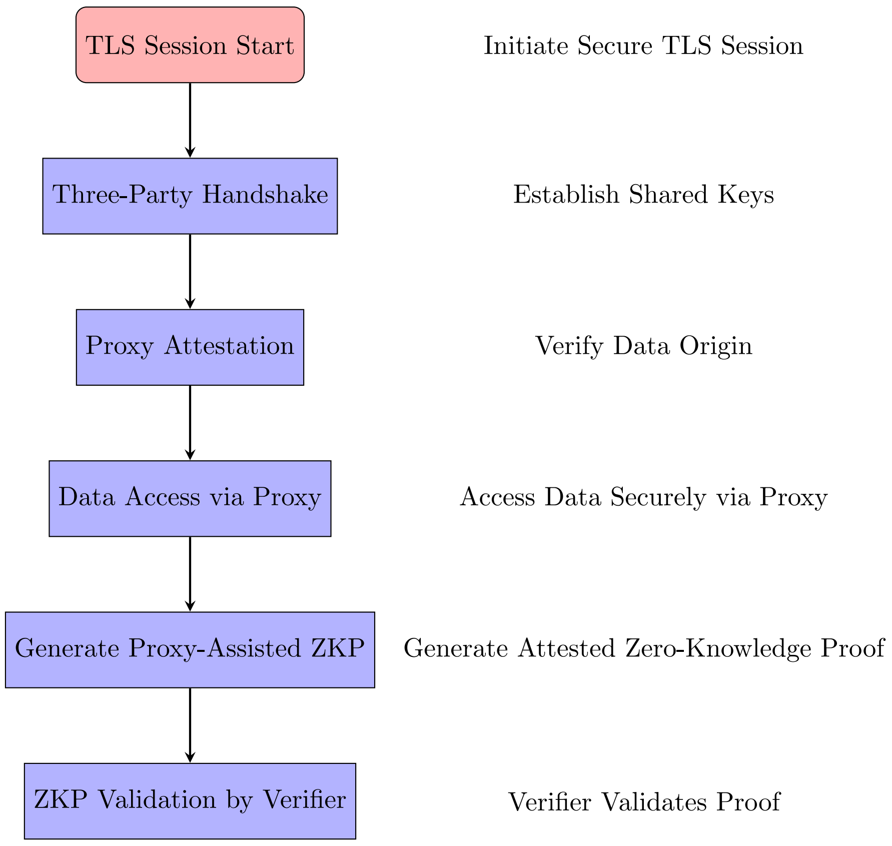
TLS Session Start
Initiate Secure TLS Session
Three-Party Handshake
Establish Shared Keys
Proxy Attestation
Verify Data Origin
Data Access via Proxy
Access Data Securely via Proxy
Generate Proxy-Assisted ZKP
Generate Attested Zero-Knowledge Proof
ZKP Validation by Verifier
Verifier Validates Proof

Figure 2: Enhanced Protocol Flow with Proxy Integration in expandZK

- **Example**: A user can reveal only their balance from a bank statement without exposing transaction history. The **ZKP** generated ensures that the balance data has not been altered.
- **Context Integrity**: expandZK ensures that the selectively opened data is verifiably part of the original session, preventing manipulation or out-of-context data revelation.

4 Implementation Details

Implementing distributed zero-knowledge proofs involves several key components and techniques. This section delves into the specifics of implementing these systems, focusing on the integration of distributed STARKs, MPC-in-the-Head, and the GKR protocol. We also discuss advanced techniques like STARKPack and modular split-and-pack, inspired by recent research advancements.

4.1 Distributed FFT and Merkle Tree Construction

To achieve efficient proof generation, we utilize distributed Fast Fourier Transform (FFT) and Merkle tree construction. These techniques are essential for handling large-scale computations and ensuring the integrity of the data.

Distributed FFT The Fast Fourier Transform (FFT) is a critical component in many cryptographic protocols, including zero-knowledge proofs. The FFT is used to perform polynomial evaluations and interpolations efficiently. In a distributed setting, the FFT algorithm is parallelized to handle large datasets by splitting the computation across multiple nodes.

Process:

1. **Data Partitioning:** The input data is divided into smaller chunks that can be processed independently. This is typically done by splitting the input size n into $\sqrt{n}$ batches.

2. **Local Computation:** Each node computes the FFT on its assigned chunk. This involves recursive decomposition of the input data into even and odd indexed elements, which are then transformed using the radix-2 Cooley-Tukey algorithm [CT65].

3. **Aggregation:** The results from all nodes are aggregated to obtain the final FFT result. This step involves combining the results from each node using appropriate weighting factors and summing them up to form the final polynomial evaluation.

$$FFT(a) = \begin{cases} \sum_{k=0}^{n-1} a_k \cdot \omega^{kj} & \text{if } n = 1 \\ FFT(a_{even}) + \omega^j FFT(a_{odd}) & \text{otherwise} \end{cases} \tag{1}$$

Benefits:

- **Scalability:** Distributed FFT allows for handling large datasets by distributing the computational load across multiple nodes, making it feasible to perform FFT on large-scale data efficiently [Sze11].

- **Efficiency:** Parallelizing the FFT computation reduces the overall processing time, making it faster than performing the computation on a single machine.

- **Reliability:** Distributed computation also improves fault tolerance, as the failure of a single node does not compromise the entire computation.

Distributed Merkle Tree Construction Merkle trees are fundamental to ensuring data integrity and providing efficient data verification. In a distributed setting, constructing a Merkle tree involves generating Merkle roots from data chunks processed by different nodes.

Process:

1. **Local Merkle Tree Generation:** Each node generates a Merkle tree for its portion of the data. This involves hashing the data elements and recursively hashing pairs of hash values until a single root hash is obtained.

2. **Root Sharing:** The root hash of each local Merkle tree is shared with other nodes. This ensures that each node has a consistent view of the overall data structure.

3. **Global Merkle Tree Construction:** The individual root hashes are combined to form a higher-level Merkle tree, resulting in a single Merkle root that represents the entire dataset.

$$\text{MerkleRoot} = \text{Hash}(\text{Hash}(leaf_1 || leaf_2) || \text{Hash}(leaf_3 || leaf_4)) \tag{2}$$

Benefits:

- **Integrity:** Merkle trees provide a way to verify the integrity of data efficiently. Any change in the data will result in a different root hash, making tampering easily detectable [Mer89].

- **Efficiency:** The hierarchical structure of Merkle trees allows for efficient verification of data subsets, as only a small portion of the tree needs to be recomputed to verify any given leaf.

- **Scalability:** Distributed Merkle tree construction allows for handling large datasets by distributing the workload, similar to the distributed FFT. This makes it feasible to construct Merkle trees for large-scale applications.

4.2 Distributed Folding

The commitment vector is folded by randomly combining paired entries using a field element r. Each machine performs this operation on its assigned entries and constructs a Merkle commitment. This step reduces the size of the commitment, making it more efficient to verify.

$$C_i = r \cdot v_{2i} + v_{2i+1} \tag{3}$$

4.3 MPC-in-the-Head

MPC-in-the-Head is a technique where a prover simulates a multi-party computation protocol internally and uses the simulation to construct a zero-knowledge proof. This method significantly reduces the communication complexity between the prover and verifier by embedding the MPC protocol within the prover's computation. The main steps include:

1. The prover simulates an MPC protocol among multiple virtual parties.
2. The prover commits to the initial state and random tapes of each virtual party.
3. Using the committed states, the prover generates a proof of correct computation by simulating the MPC protocol.
4. The verifier checks the commitments and verifies the proof, ensuring the integrity of the computation.

Benefits of MPC-in-the-Head

- **Efficiency**: MPC-in-the-Head reduces the overhead of traditional multi-party computation protocols by embedding the interaction within the prover. This results in lower communication costs and faster proof generation.
- **Scalability**: By distributing the computation among multiple virtual parties, MPC-in-the-Head can handle larger and more complex proofs, enhancing the scalability of the ZK system.
- **Security**: The technique ensures that the security and integrity of the data are maintained, as the prover does not need to reveal the intermediate states of the computation.

4.4 GKR Protocol

The GKR protocol provides a scalable method for verifying computations represented as layered arithmetic circuits. It offers logarithmic communication complexity in the size of the circuit, making it ideal for large-scale computations. The protocol involves the following steps:

1. The computation is represented as a layered arithmetic circuit.
2. The prover sends commitments to the values of the circuit at each layer.
3. The verifier performs random checks on these commitments to ensure the correctness of the computation.
4. The interaction is repeated for each layer, reducing the communication overhead.

Benefits of GKR Protocol

- **Efficiency**: The GKR protocol's logarithmic communication complexity allows for efficient verification of large-scale computations, making it suitable for complex smart contracts and AI models.
- **Flexibility**: The protocol can be applied to various types of arithmetic circuits, providing flexibility in the types of computations that can be verified.
- **Security**: By using interactive proofs, the GKR protocol ensures that the verifier can be confident in the correctness of the computation without needing to perform the computation itself.

4.5 STARKPack and Modular Split-and-Pack

Recent advancements in STARK-based proof systems have introduced techniques such as packing and modular split-and-pack, which significantly improve the efficiency of proof generation and verification.

STARKPack STARKPack enables the generation of a single proof for multiple instances by randomly combining constraints and polynomials. This reduces the workload for verifiers and minimizes proof sizes, particularly beneficial for large traces [GHJ+24]. The process involves the following steps:

1. The prover generates multiple proofs for different instances of the same statement.
2. These proofs are then combined into a single proof using random linear combinations of the constraints and polynomials.
3. The combined proof is verified by checking the combined constraints and polynomials, reducing the overall verification workload.

Modular Split-and-Pack Modular split-and-pack is a proof acceleration technique where the prover divides a witness into smaller sub-witnesses and uses packing to prove their simultaneous satisfiability. This method improves prover time and memory usage while increasing verifier time and proof size. The key steps include:

1. Splitting the execution trace of size $n \times r$ into smaller chunks of size $n/k \times r$.
2. Translating the constraints for the entries in the chunks into polynomial equations involving the polynomials interpolating the new, smaller columns.
3. Applying packing to the evaluations of the witness polynomials at the same point in the evaluation domain.
4. Using a single validity function for all batched-FRI tests, reducing the overall complexity.

Benefits of STARKPack and Modular Split-and-Pack

- **Reduced Proof Size**: By packing multiple instances into a single proof, the overall proof size is reduced, making it more efficient for verification.
- **Improved Prover Time**: Splitting the execution trace and using packing techniques improve the prover's time by reducing the complexity of the computations.
- **Enhanced Scalability**: These techniques enable the handling of larger execution traces and more complex computations, making them suitable for large-scale applications.
- **Lower Communication Costs**: By reducing the number of Merkle commitments and leveraging batched-FRI tests, the communication costs between the prover and verifier are minimized.

5 Use Cases for expandZK

5.1 Verifiable Credentials

expandZK can be applied to verifiable credentials, enabling security-preserving proofs of specific attributes, such as age or income. This is especially valuable in contexts where sensitive data should remain confidential while allowing specific details to be verified.

5.2 DeFi (Decentralized Finance)

expandZK can integrate with decentralized financial systems to provide proof of income or credit history without revealing detailed financial records. This enhances user security while maintaining the integrity of financial transactions in **Web3** applications.

5.3 Data Portability

expandZK allows users to export their data from **Web2** platforms, such as social media, into **Web3** applications while ensuring data security. Only relevant portions of the data are exposed, ensuring security throughout the process.

5.4 Proof of Payment

expandZK can be used in verifying **Web2** payment data for **Web3** applications, such as proving a payment was made to a specific service without revealing the full payment history. This is particularly useful for financial transactions that require security but also verifiability.

6 Handling Public User Data vs. PII in expandZK

As the internet evolves toward decentralized and security-first paradigms, it becomes increasingly important to differentiate between public user data and data, especially **Personally Identifiable Information (PII)**. expandZK provides a flexible mechanism to handle these two types of data with different security and protection guarantees.

6.1 Public User Data

Public user data refers to non-sensitive information that is intended to be shared openly and is often publicly accessible. Examples of such data include social media posts, public profiles, and forum contributions. While public user data does not require the same level of security protection as data, it still requires verifiability and integrity to ensure trustworthiness.

In expandZK, public user data is:

- **Accessible with Verifiability**: The focus for public user data in expandZK is on proving the authenticity and integrity of the data. Users can generate **Zero-Knowledge Proofs (ZKPs)** to demonstrate that this data originates from a trusted source without revealing unnecessary metadata.
- **Reduced Overhead**: Since security is less critical for public data, expandZK can optimize ZKP generation by simplifying proofs and reducing computational overhead. This ensures that public data can be verified efficiently, especially for applications like decentralized content platforms and public blockchains.

6.2 Personally Identifiable Information (PII)

Personally Identifiable Information (PII) includes sensitive data such as names, addresses, social security numbers, and financial details. PII requires a higher level of protection because unauthorized access or exposure could lead to identity theft, fraud, or other security violations.

expandZK provides the following protections for PII:

- **Strict Security Preservation**: For PII, expandZK employs advanced ZKP techniques to ensure that data is never exposed during the verification process. Users can generate ZKPs to prove that specific attributes (e.g., age, income level) meet certain criteria without revealing the underlying PII.
- **Selective Opening with Context Integrity**: expandZK's selective opening feature is crucial for handling PII. Users can choose to reveal only a subset of their data (e.g., verifying a person is over 18 without revealing the exact age) while ensuring the context of that data is maintained, preventing partial or out-of-context disclosures.
- **End-to-End Data Security**: Even when data moves between Web2 and Web3 systems, expandZK ensures that PII remains confidential. The cryptographic guarantees built into expandZK prevent third parties from intercepting or accessing sensitive data, even during data exports.

6.3 Differentiated Protocol Handling

The key innovation in expandZK for handling public vs. PII lies in its flexible protocol design:

- **Adaptable ZKP Complexity**: expandZK can adapt the complexity of ZKPs based on the data type. For public user data, simpler proofs focusing on data integrity can be used, while for PII, more complex proofs ensuring security and protection are applied.
- **Policy-Driven Disclosure**: Organizations can configure expandZK to enforce specific policies on public data vs. PII. For example, an institution may allow users to prove certain public credentials while keeping PII credentials (e.g., income or health records) fully protected.

6.4 Applications

- **Public User Data**: Decentralized social platforms, online forums, and Web3 applications can use expandZK to verify user-generated content and ensure its integrity while maintaining transparency.
- **PII**: Applications like decentralized finance (DeFi) platforms, healthcare systems, and identity verification services can leverage expandZK to handle sensitive PII securely, allowing users to prove eligibility or credentials without exposing their entire identity.

6.5 Ensuring Compliance with Security Laws

expandZK's handling of PII aligns with major security regulations such as **GDPR (General Data Protection Regulation)** and **CCPA (California Consumer Security Act)**, ensuring that PII is handled appropriately while offering strong security guarantees. The ability to control what data is revealed and to whom, combined with cryptographic proof of compliance, makes expandZK a powerful tool for legal compliance.

7 Future Innovations

7.1 Post-Quantum Cryptography

As the threat of quantum computing looms, expandZK can integrate post-quantum cryptographic algorithms to ensure the long-term security of the protocol. Innovations in **ZKP** and **TLS** compatible with post-quantum systems will further enhance expandZK's resilience.

7.2 Security Preserving Smart Contracts

expandZK opens up possibilities for smart contracts that interact with external data without compromising security. Future iterations of expandZK could explore tighter integration with **ZK rollups** and other layer 2 scaling solutions to reduce the computational cost of proof generation.

8 Conclusion

expandZK represents a novel approach to bridging **Web2** and **Web3**, combining the security guarantees of **TLS** with the security-preserving nature of **ZKPs**. Through **selective opening**, **context integrity**, and efficient proof generation, expandZK opens up new possibilities for **decentralized finance**, verifiable credentials, and security-preserving data portability. By ensuring cryptographic proofs of data provenance without exposing the underlying data, expandZK ensures the seamless and secure integration of **Web2** data into the **Web3** ecosystem.

References

[CT65] J. W. Cooley and J. W. Tukey. An algorithm for the machine calculation of complex fourier series. *Mathematics of Computation*, 19(90):297–301, 1965.

[GHJ$^+$24] Albert Garreta, Hayk Hovhanissyan, Aram Jivanyan, Ignacio Manzur, Isaac Villalobos, and Michał Zając. On amortization techniques for fri-based snarks. Cryptology ePrint Archive, Paper 2024/661, 2024. `https://eprint.iacr.org/2024/661`.

[Mer89] R. C. Merkle. A digital signature based on a conventional encryption function. In *Advances in Cryptology — CRYPTO '87*. Springer, 1989.

[Sze11] Tsz-Wo Sze. The schönhage-strassen algorithm with mapreduce for multiplying terabyte size integers. *arXiv preprint arXiv:1111.5139*, 2011.