TLShare: Private Authenticated MPC and FHE Inputs Over TLS

Manuel B. Santos[1], Dimitris Mouris[1], Xiang Xie[2], Miguel de Vega[1], and Andrei Lapets[1]

[1] Nillion Labs
manuel.batalha.santos@gmail.com, {dimitris, miguel, andrei.lapets}@nillion.com

[2] Primus Labs
xiexiangiscas@gmail.com

Abstract. Transport Layer Security (TLS) is the backbone of the web, allowing clients to establish secure and private channels with servers. DECO (CCS'20) and follow-up works proposed protocols that enable proving the provenance of a TLS response, i.e., that a payload came from a particular server, without needing server-side modifications. Unfortunately, these works are limited to proving Boolean statements over the payload (e.g., age ≥ 18) and cannot combine payloads from multiple clients.

We introduce TLShare, a framework that extracts authenticated data from a TLS connection and imports it into secure multiparty computation (MPC) or fully homomorphic encryption (FHE), without requiring server-side changes or exposing client credentials. Unlike prior work, TLShare allows the payload itself, not just a predicate about it, to serve as private input to secure downstream computation. TLShare supports combining verifiable inputs across multiple clients and servers, enabling new applications such as privacy-preserving financial risk assessment and collaborative analytics. We design three protocols for TLShare: one for MPC using verifiable secret sharing, and two for FHE using interactive and non-interactive zero-knowledge proofs, each ensuring input authenticity, integrity, and end-to-end privacy. We evaluate all three protocols of TLShare over both LAN and WAN settings, comparing their trade-offs and demonstrating their practicality.

Keywords: Fully Homomorphic Encryption, Multiparty Computation, TLS, Zero-Knowledge Proofs

1 Introduction

Transport Layer Security (TLS) is the de facto standard for securing communication over the Internet, providing confidentiality, integrity, and authentication guarantees between clients and servers [28,76]. However, reusing data transmitted over a TLS connection in a separate application context (e.g., proving a property about the payload to a third party) poses significant challenges [97]. The TLS payload is cryptographically bound to the session between the client and the server, and its authenticity is not easily verifiable outside that context. One obvious solution is to modify the TLS server (e.g., a bank) to support cryptographic proofs or attestation mechanisms; however, this is highly impractical, as organizations are unlikely to implement and maintain custom TLS extensions for arbitrary downstream applications requested by their clients [16]. Conversely, simply handing over client credentials to a third-party application breaks the fundamental trust model and compromises both security and privacy [60]. These limitations highlight the need for mechanisms that allow clients to extract verifiable private data from TLS sessions without requiring server cooperation or revealing sensitive credentials.

DECO [98] introduced the concept of a decentralized oracle, which partially addresses the problem of reusing TLS payload in external applications. It enables a client to prove statements about data received over a TLS connection in zero knowledge, without requiring any server modifications. DECO extracts authenticated data from the TLS session and converts it into a zero-knowledge proof (ZKP) friendly form, allowing clients to prove predicates on the payload (e.g., threshold conditions) [33]. However, DECO ultimately outputs only a Boolean value (i.e., whether the predicate holds), which limits expressiveness. Additionally, the proofs are often published on-chain [15], revealing information about the data they certify (e.g., age ≥ 18). Finally, decentralized oracles [98,33,77,20,90,18,53] do not allow combining data from multiple clients as the prover must possess all underlying secrets in the clear.

This motivates the need to extend beyond existing oracle paradigms and explore a new class of architectures that preserve both privacy and authenticity in a more expressive and composable way. We broadly categorize existing oracles into *public-to-public* and *private-to-public* models. The former type exposes both input and output data, limiting their applicability to non-sensitive domains [15,56]. Private-to-public oracles [98,77,20,53], allow clients to keep their input data private while releasing a public zero-knowledge proof of a predicate over that data. However, these oracles suffer from two severe limitations: a) they cannot combine data across users without violating privacy, and b) they rely on simple ZKPs that reveal a single bit of information instead of offering support for any private downstream computation.

Private-to-private oracle. To address these limitations, we introduce a new oracle model, called *private-to-private*, which extracts private data from a TLS connection and imports it into private computation servers for secure multiparty computation (MPC) or fully homomorphic computation (FHE) while keeping the data private end-to-end and maintaining data provenance. This opens the door to new use cases such as credit scoring for collateralized lending [45], or aggregate statistical analysis across individuals and institutions [24,52,62,66,65] that maintain strict cryptographic trust without exposing any sensitive information. To that end, we introduce a novel framework called TLShare,[3] which is the first private-to-private oracle for both MPC and FHE downstream computation. Contrary to approaches that require institutions (e.g., banks, hospitals) to engage in private computation with each other, TLShare lowers the barrier by introducing a user-driven approach. TLShare allows users to ensure which data points are used in what computation, instead of a direct inter-institutional MPC/FHE where clients have little to no control over their data [42,64,23,32]. Additionally, secure computation between institutions requires significant trust and coordination, often hindered by regulatory or competitive concerns (e.g., competitive dynamics, GDPR constraints, operational overhead). In contrast, with TLShare, clients connect to TLS endpoints that serve as authentic and verifiable data sources without having to participate in any secure computation. Finally, in contrast to private-to-public oracles, where a ZKP proves a statement about only a single client's data, TLShare allows data from multiple clients to be combined under secure computation while proving provenance.

1.1 Our Contribution

We propose TLShare, the first private-to-private oracle that allows private computation over authenticated and private TLS payloads. The client facilitates the data-sharing process by extracting authenticated data (e.g., financial information) from trusted sources (e.g., banks, credit bureaus) and inputting it into MPC or FHE servers for downstream computation. For instance, after importing financial data from multiple institutions, the downstream MPC/FHE can perform a financial risk assessment for the user without revealing the raw data to lenders. TLShare minimizes adoption barriers for private computation, as institutions simply handle a standard TLS connection (e.g., a client accessing their bank account) and remain completely agnostic to the downstream MPC/FHE. This TLS connection is the foundation for clients to input their response payload with provenance and MPC/FHE servers to verify its authenticity. To that end, in Section 6 we elaborate on two motivating applications for TLShare. Below, we outline the three novel protocols of TLShare.

1.1.1 MPC mode. This protocol (Section 4) focuses on private downstream computation within an MPC network. At a high level, the user and an attestor first perform a two-party computation and connect to a TLS server. This 2PC is secure against malicious adversaries and ends up with commitments on each bit of the TLS server's response. Next, we use these commitments as the base commitments for verifiable secret sharing, associating them with the randomness and the commitments produced during the TLS phase. Finally, each MPC node verifies the correctness of its share against the original TLS commitment using efficient consistency checks.

1.1.2 FHE mode with Non-Interactive ZK Proofs. Our second protocol (Section 5.1) begins similarly to our first one, in that we start with commitments for each bit of the TLS response. Next, the client encrypts the TLS response using a public key version of an FHE scheme based on the learning with errors problem. We

[3] Pronounced "TLS Share".

focus on the CGGI scheme [22], but our techniques can be generalized to other schemes. To prove that the ciphertext is a valid encryption of the same message committed in the TLS phase, the client and the MPC nodes run a non-interactive ZK (NIZK) proof. The client starts by constructing a binary vector encoding the message bits, encryption randomness and noise, and commits to this vector using a pairing-based vector commitment scheme (see Section 2.4). Finally, we introduce an NIZK argument of vector membership to show that the value committed in each commitment corresponds to the message bits contained within the vector commitment. This ensures that the ciphertext is well-formed and tied to the authenticated TLS response.

1.1.3 FHE mode with Interactive ZK Proofs. Last but not least, our third protocol (Section 5.2) relies on information-theoretic message authentication codes (IT-MACs) using an interactive zero-knowledge (IZK) [91] protocol to prove the validity of Regev [74] FHE encryptions. More specifically, the client proves in zero knowledge to the attestor that the ciphertext is computed correctly using the response plaintext, a binary randomness vector, and bounded noise terms. The attestor then signs the ciphertext, and finally, an FHE server verifies that it corresponds to the original TLS response, without seeing the plaintext.

1.2 Related Work

Proving various properties of data obtained from TLS connections has been an increasingly popular area of research. We classify these works as private-to-public oracles since private data from a TLS connection (e.g., age) results in a public proof (e.g., age ≥ 18), usually used in smart contracts. Another crucial distinction between private-to-public oracles and our work is that the ZKP in the former is quite limited (True/False statements) compared to the capabilities of TLShare – see Section 6. This line of work is also referred to as designated-commitment TLS (DCTLS) protocols and focuses on ensuring data authenticity and integrity while providing privacy guarantees.

Town Crier (TC) [97] was an early pioneer, relying on a trusted execution environment (TEE), specifically Intel Software Guard Extensions (SGX), to act as a trusted bridge between smart contracts and external websites. However, its reliance on SGX comes with a multitude of attacks [95,84,21]. Unlike TC, our approach does not depend on hardware-based trust assumptions. The notable work of DECO [98] enables clients to demonstrate data provenance and prove statements about it selectively or in zero-knowledge, all without requiring any protocol modifications on the TLS server side. DECO's design is tailored for TLS 1.2, which remains widely supported across many real-world systems.[4] Despite its utility, DECO is susceptible to attacks during the multiplicative-to-additive (MtA) protocol [83,61,18]. Dido [20] builds upon DECO by introducing a round-optimal three-party key exchange along with enhanced X25519-based protocols. TLSNotary [72] further extends this model by separating the verifier into two roles: a verifier and a notary who co-signs the client's transcript, enabling auditability and trust separation. Recently, Garble-then-Prove (GtP) [90] demonstrated substantial performance improvements over DECO by optimizing several two-party protocols and transforming authenticated garbling bits into zkSNARK-friendly commitments. While all these protocols focus largely on TLS 1.2, Janus [53] introduces performance enhancements and extends support to both 1.2 and 1.3. However, Janus requires clients to pre-select cipher suites, reducing negotiation flexibility and increasing implementation complexity. Origo [33] focuses on TLS 1.3 and follows the GtP approach by converting the commitment into a ZK-friendly one. It introduces a two-party computation (2PC)-free oracle solution, effectively reducing 2PC overheads, but at the cost of requiring additional security assumptions, such as resilience to a proxy that allows the client to perform man-in-the-middle attacks, a notably strong assumption.

Several works explore private features over TLS connections, often requiring modifications on the server side: TLS-N [77] introduces a mechanism for non-repudiation of origin, Oblivious TLS [2] enables a client running a standard TLS endpoint to communicate securely with a set of MPC servers. An orthogonal approach, Compute but Verify [31], presents a method to execute an MPC protocol on authenticated inputs. As these works rely on server-side extensions, the cooperation of servers in adopting their protocols is essential,

[4] DECO has explored TLS 1.3 but has introduced vulnerabilities [18], largely attributable to the technical complexities introduced by TLS 1.3's revised handshake protocol and key derivation mechanisms.

and something we see as a significant lift. In contrast, TLShare offers a novel approach to authenticate MPC inputs without requiring any changes to the server infrastructure.

PECO [80] and DiStefano [18] focus on providing an additional property: the client's privacy. PECO offers two modes: one that provides ZKP of valid signatures for authentication among a set of k verifier-approved servers (k-anonymity), and another that verifies the validity of the server's signature in zero-knowledge. However, PECO uses a broken primitive [19] and lacks an implementation. DiStefano, on the contrary, generates secure private commitments over TLS 1.3 data (relying on modified AES) and introduces a new primitive called "zero-knowledge proofs of knowledge of valid signatures (ZKPVS)" that proves in zero-knowledge that a signature is valid in relation to a set of public keys. This property essentially allows a client using DiStefano to prove that they are connecting with the intended server without revealing the server's identity (which will reveal the client's browsing history over time). ZKPVS is orthogonal to TLShare and compatible with our approach.

All the aforementioned works implement private-to-public oracles, focusing on proving a property of the TLS traffic in zero knowledge. In contrast, TLShare advances introduces the first *private-to-private oracle*, supporting three distinct modes for secure downstream computation on private data.

MPCAuth [82] is a related work that enables a client to authenticate to multiple TLS servers independently with the work of only one authentication. Although MPCAuth uses a similar technique of encrypting the TLS channel within MPC, it solves a different problem than private-to-public oracles, that of authentication. Likewise, zk-creds [78] focus on anonymous credentials such as proving properties from a passport to anonymously access age-restricted videos. Lastly, zkLogin [4] leverages identity tokens to authenticate blockchain transactions and ensure that the off-chain and on-chain identities of users remain hidden.

Finally, there exists a special class of ZKP systems over secret shared data where the servers can verify some property of their shares without learning the underlying data [23,12,13,27,65]. These ZKPs have been increasingly interesting in the realm of privacy-preserving statistics, such as histograms and heavy-hitters, but they do not support any form of input authenticity from a third party.

2 Preliminaries

2.1 Multiparty Computation (MPC)

MPC enables multiple parties to compute a function $f(x_1, \ldots, x_n)$ over their inputs $x_1, \ldots, x_n$ without revealing their private data [93,94]. MPC protocols provide different security guarantees: semi-honest security assumes participants follow the protocol honestly but may try to infer additional information, while malicious security accounts for participants who deviate from the protocol. We view these protocols in two categories: garbled circuits [94,41,51,88,92] and secret-sharing protocols [26,50,49]. Garbled circuits focus on Boolean circuits such as AES and SHA-256, comprising `AND`, `XOR`, etc. gates and encode the function f as the circuit and evaluate an encrypted variant of it. Secret-sharing techniques focus on arithmetic circuits (consisting of additions and multiplications) and split the inputs into seemingly random values, which are then distributed between the parties.

One popular scheme is Shamir's secret sharing [81] that constructs a random polynomial $F(x) = \mu + a_1 x + a_2 x^2 + \cdots + a_{t-1} x^{\mathsf{T}-1}$ over a finite field $\mathbb{F}_q$, where μ is the secret message and T is the reconstruction threshold. The share for party i is a pair $(i, F(i))$, and any T of shares can recover μ via Lagrange interpolation. We use $[\![x]\!]$ to refer to shares of x and $[\![x]\!]_i$ to the share of x owned by party i. In TLShare, we use Pedersen verifiable secret sharing (PVSS) [70], where each share is also accompanied by a proof verifying its validity. In PVSS, the dealer commits to the secret using Pedersen commitments [70] (i.e., $g^{\mu} \cdot h^{r}$ for generators g and h and random r) and distributes Shamir shares ($[\![\mu]\!]_i, [\![r]\!]_i$) along with Pedersen commitments to the coefficients of the corresponding Shamir polynomials. We provide more details about PVSS and its integration with TLShare in Section 4.

2.2 Fully homomorphic encryption (FHE)

Fully homomorphic encryption (FHE) enables computation directly on encrypted data, supporting arbitrary encrypted additions and multiplications without the need to decrypt intermediate results [40,85,43].

Modern FHE constructions rely on the hardness of the Learning With Errors (LWE) problem [74] or its ring-based variant, RLWE [59], which encrypt messages using noisy inner products. The LWE problem, introduced by Regev [74], is based on the difficulty of recovering a secret $\boldsymbol{s}$ given samples of the form $\boldsymbol{b} = \boldsymbol{A} \cdot \boldsymbol{s} + \boldsymbol{e}$, where $\boldsymbol{A} \in \mathbb{Z}_q^{m \times n}$ is a public matrix, $\boldsymbol{s} \in \mathbb{Z}_q^n$ is a secret vector, and $\boldsymbol{e} \in \mathbb{Z}^m$ is a small noise vector. From the LWE assumption, one can construct a public-key encryption scheme consisting of three algorithms $(\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ such that $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}$ and $\mathsf{Dec}(\mathsf{sk}, \mathsf{Enc}(\mathsf{pk}, \mu)) = \mu$. Furthermore, an LWE-based FHE scheme supports both addition and multiplication over encrypted data. Specifically, $\mathsf{Enc}(\mathsf{pk}, \mu_1) \oplus \mathsf{Enc}(\mathsf{pk}, \mu_2) = \mathsf{Enc}(\mathsf{pk}, \mu_1 + \mu_2), \quad \mathsf{Enc}(\mathsf{pk}, \mu_1) \otimes \mathsf{Enc}(\mathsf{pk}, \mu_2) = \mathsf{Enc}(\mathsf{pk}, \mu_1 \cdot \mu_2)$, where the operations $\oplus$ and $\otimes$ on ciphertexts correspond to plaintext addition and multiplication, respectively.

Regev Scheme. Choose the dimension of the secret-key n, the ciphertext modulus q, and a noise distribution χ (e.g., discrete Gaussian) based on a security parameter λ. The Regev encryption scheme [74] is defined as follows. The key generation algorithm KeyGen samples a secret key uniformly $\boldsymbol{s} \leftarrow \mathbb{Z}_q^n$, a matrix $\boldsymbol{A} \leftarrow \mathbb{Z}_q^{n \times m}$, and noise $\boldsymbol{e} \leftarrow \chi^m$, then sets $\boldsymbol{y}^\top = \boldsymbol{s}^\top \boldsymbol{A} + \boldsymbol{e}^\top$. The public key is $(\boldsymbol{A}, \boldsymbol{y})$ and the secret key is $\boldsymbol{s}$. Here, $m = \Omega(n \log q)$. To encrypt a bit $\mu \in \{0, 1\}$, sample $\boldsymbol{r} \leftarrow \{0, 1\}^m$ and compute $(\boldsymbol{a}, b) = (\boldsymbol{A}\boldsymbol{r},\ \boldsymbol{y}^\top \boldsymbol{r} + \lfloor \mu \cdot q/2 \rfloor)$ mod q. To decrypt, compute $b - \boldsymbol{s}^\top a \mod q$ and output 0 if the result is closer to 0 than to $q/2$, and 1 otherwise.

Subsequent work improved on Regev's initial scheme by decreasing the size of the public key [48,59,57]. The LPR [59] scheme decreases the matrix $\boldsymbol{A}$ to an $n \times n$ matrix. This is achieved by adding noise to the ciphertext components, i.e. $(\boldsymbol{a}, b) = (\boldsymbol{A}\boldsymbol{r} + \boldsymbol{e},\ \boldsymbol{y}^\top \boldsymbol{r} + e_0 + \lfloor \mu \cdot q/2 \rfloor) \bmod q$, where $\boldsymbol{r}, \boldsymbol{e} \leftarrow \chi^n$ and $e_0 \leftarrow \chi$.

CGGI Scheme. Encrypted operations result in noise growth, which eventually prevents decryption. To allow for unlimited encrypted operations, a technique called bootstrapping resets ciphertext noise down to minimal levels [30,22]. The schemes by Ducas–Micciancio (DM) [30] and Chillotti–Gama–Gentry–Izabachene (CGGI) [22] (also known as TFHE) allow evaluating arbitrary univariate functions during bootstrapping, further enhancing the capabilities of these schemes as it can be used to replace a series of additions and multiplications [58,44]. Currently, CGGI [22] has the most efficient bootstrapping in terms of latency and is thus the cryptosystem we use as one of our back-ends.

Let us now recall the CGGI scheme in more detail. For a modulus $q \in \mathbb{N}$, define the discretized torus as $\mathbb{T}_q = q^{-1}\mathbb{Z}/\mathbb{Z}$ [47]. The plaintext space is $\mathbb{Z}_p$, and the secret key space is $\{0, 1\}^n \subset \mathbb{Z}^n$, where $p \mid q$ and $\Delta = q/p$. The CGGI secret key is defined as $(s_1, \ldots, s_n) \in \{0, 1\}^n$. To encrypt a message $\mu \in \mathbb{Z}_p$ under the secret key, one samples a random vector $(a_1, \ldots, a_n) \in \mathbb{T}_q^n$ uniformly and samples an error term $e \in \mathbb{Z}$ from a suitable distribution. The ciphertext is defined as $(a_1, \ldots, a_n, b) \in \mathbb{T}_q^{n+1}$, where: $b = \sum_{j=1}^{n} s_j \cdot a_j + \mu^*$ and $\mu^* = (\Delta \cdot \mu + e)/q \in \mathbb{T}_q$. This encryption is additively homomorphic and supports bootstrapping to maintain decryptability even after many homomorphic operations.

The public-key version presented in [46] follows the method proposed by Rothblum [79], where the public key encodes encryptions of the zero message under different randomness. In summary, it constructs a public key matrix $\boldsymbol{A} \in \mathbb{T}_q^{m \times (n+1)}$ by sampling m independent encryptions of 0 under $\boldsymbol{s}$. The public key is $\boldsymbol{A}$, and the secret key is $\boldsymbol{s}$. To encrypt a message $\mu \in \mathbb{T}_p$, sample a random vector $\boldsymbol{r} \leftarrow \{0, 1\}^m$ and compute $(\boldsymbol{a}, b) = \boldsymbol{r}^\top \boldsymbol{A} + (0, \ldots, 0, \mu) \in \mathbb{T}_q^{n+1}$. Joye [48] uses the LPR [59] approach to reduce the size of the public key by adding small noise to the ciphertext.

2.3 Zero-Knowledge Proofs (ZKP)

A ZKP allows a prover $\mathcal{P}$ to convince a verifier $\mathcal{V}$ that some given statement is true, without revealing any information beyond the fact that the statement is true [63,10,7,39,6,67,11]. Informally, ZKPs must be: *complete*, meaning that valid proofs generated honestly are always accepted; *sound*, meaning that it is infeasible to produce a convincing proof for a false statement; and *zero-knowledge*, meaning that the proof reveals nothing about the underlying witness [63,54]. In the context of this work, we focus on two specific ZKPs: a non-interactive zero-knowledge (NIZK) for proving correct encryptions, and an interactive zero-knowledge (IZK) system called QuickSilver [91].

NIZK. We consider NIZK arguments as a tool for generating verifiable CGGI ciphertexts (Section 5.1). Let $\{R_\lambda\}_\lambda$ be a family of NP relations parameterized by the security parameter λ [54]. A NIZK argument for this family consists of the following three algorithms:

- $\mathsf{NIZK.GenCRS}(1^\lambda, \mathsf{lpp}) \to (pp, \tau)$: generates a common reference string pp and a simulation trapdoor τ, possibly using language-dependent parameters lpp. The relation to be proven becomes $\mathcal{R}_{\mathsf{pp}}$.
- $\mathsf{NIZK.Prove}_{\mathsf{pp}}(x, w) \to \pi$: given a statement x and witness w such that $(x, w) \in \mathcal{R}_{\mathsf{pp}}$, returns a proof π.
- $\mathsf{NIZK.Verify}_{\mathsf{pp}}(x, \pi) \to \{0, 1\}$: checks whether π is a valid proof for x under pp.

In addition to the aforementioned ZKP algorithms, we include a commitment algorithm $\mathsf{NIZK.Com}_{\mathsf{pp}}(w) \to V$ allowing $\mathcal{P}$ to commit to the witness such that it can be linked to the proof.

IZK with QuickSilver. QuickSilver [91] is an IZK protocol [89,5] and leverages Information-Theoretic Message Authentication Codes (IT-MACs) to authenticate Boolean and arithmetic circuits. IT-MACs are also widely used in MPC [8,87,29]. QuickSilver involves a prover and a verifier where $\mathcal{V}$ holds a private key $\Delta \in \mathbb{F}_{2^\lambda}$ and a local key $K[x] \in \mathbb{F}_{2^\lambda}$, while $\mathcal{P}$, holds a bit $x \in \{0, 1\}$ and a MAC tag of the form $M[x] = K[x] + x \cdot \Delta \in \mathbb{F}_{2^\lambda}$. In QuickSilver, the prover commits to a witness and to all wire values of the circuit using IT-MACs and then evaluates the circuit interactively with $\mathcal{V}$ to obtain IT-MACs on output values. We use $\langle\langle x \rangle\rangle$ to refer to such verifiable shares with IT-MACs, where $\mathcal{P}$ holds x and $M[x]$, and $\mathcal{V}$ holds $K[x]$ and Δ. QuickSilver proves circuit satisfiability and verifies correctness with batched checks [68,89] and achieves constant-size communication.

2.4 Vector Commitments over Pairings

Vector commitments allow a sender to commit to an ordered sequence $\boldsymbol{w} = (w_1, \ldots, w_n) \in \mathbb{Z}_p^n$ and later open the value at a specific position i along with a proof that w_i is indeed the committed message at that index [17]. One of the key advantages of vector commitments is that the size of the commitment remains constant, regardless of the size of the vector. In this work, we focus on the vector commitment scheme from [54], which is based on bilinear pairings and includes the additional functionality of proving inner product relations. Specifically, it enables the prover to demonstrate that a committed private vector $\boldsymbol{w}$ satisfies a linear relation $\langle \boldsymbol{w}, \boldsymbol{t} \rangle = x$, where $\boldsymbol{t}$ and x are public.

A bilinear map, or asymmetric pairing, is a function $e : \mathbb{G} \times \hat{\mathbb{G}} \to \mathbb{G}_T$ defined over two source groups $\mathbb{G}$ and $\hat{\mathbb{G}}$. We assume a structured common reference string (CRS) that defines the group elements $(g, \{g_i = g^{\alpha^i}\}_{i \in [1,2n] \setminus \{n+1\}})$ and $(\hat{g}, \{\hat{g}_i = \hat{g}^{\alpha^i}\}_{i=1}^n)$, for a hidden scalar $\alpha \in \mathbb{Z}_p$ [55,54]. This structure relies on the assumed hardness of the $(2n, n)$-Discrete Logarithm problem, which assumes computing α from the CRS is hard [54, Definition 1].

To commit to a vector $\boldsymbol{w} = (w_1, \ldots, w_n) \in \mathbb{Z}_p^n$, the sender samples $\gamma \xleftarrow{R} \mathbb{Z}_p$ and computes the vector commitment:

$$V = g^\gamma \cdot \prod_{j=1}^n g_j^{w_j} = g^{\gamma + \sum_{j=1}^n w_j \cdot \alpha^j}. \quad (1)$$

To open a specific position i in the committed vector, the sender provides a proof $\pi_i = g_{n+1-i}^\gamma \cdot \prod_{j=1, j \neq i}^n g_{n+1-i+j}^{w_j} = (V / g^{w_i \cdot \alpha^i})^{\alpha^{n+1-i}}$, which can be verified by checking the pairing equation $e(V, \hat{g}_{n+1-i}) = e(\pi_i, \hat{g}) \cdot e(g_1, \hat{g}_n)^{w_i}$, confirming that w_i is the message committed at index i (with $i \in [1, n]$).

The use of pairings also enables proving that the committed vector satisfies a linear relation of the form $\langle \boldsymbol{m}, \boldsymbol{t} \rangle = x$, for public $\boldsymbol{t} \in \mathbb{Z}_p^n$ and $x \in \mathbb{Z}_p$. This is done by raising the verification equation to $t_i \in \mathbb{Z}_p$ and multiplying over all $i \in [1, n]$ as $e\left(V, \prod_{i=1}^n \hat{g}_{n+1-i}^{t_i}\right)$, which equals $e\left(\prod_{i=1}^n \pi_i^{t_i}, \hat{g}\right) \cdot e(g_1, \hat{g}_n)^{\sum_{i=1}^n w_i \cdot t_i}$.

3 Overview of TLShare

3.1 Functionality & Threat Model

In this section, we present an overview of TLShare, a framework that enables the extraction of data from a TLS session into secure computation techniques. TLShare operates as a private-to-private oracle, allowing

Ideal Functionality $\mathcal{F}_{\text{TLShare}}$

This functionality interacts with a server $\mathcal{S}$, a client, an attestor, $n > 1$ Nodes $\mathcal{N}_i$, and an adversary.

- **Public Parameters:** A mode parameter $\mathsf{mode} \in \{\mathsf{fhe}, \mathsf{mpc}\}$. A query template Query for the server $\mathcal{S}$ (e.g., a `GET` request) and a threshold T for secret sharing.
- **Server $\mathcal{S}$:** has no input ($\perp$) to the functionality but hosts and operates a public website.
- **Client:** has a private input (e.g., a password) Inp.
- **Attestor:** has no input ($\perp$).
- **Node $\mathcal{N}_i$ (for $i \in [1, n]$):** In case of mpc mode, the nodes have no input ($\perp$). In case of fhe mode, the nodes have shares of the secret key of an FHE key pair ($[\![\mathsf{sk}_{\mathsf{fhe}}]\!]_i, \mathsf{pk}_{\mathsf{fhe}}$).

TLS Phase (Handshake, Query, and Response):

1. On input ($\mathsf{Init}, \mathsf{sid}, \mathcal{S}, \mathsf{Inp}, \mathsf{Query}$) from the client and ($\mathsf{Init}, \mathsf{sid}$) from the attestor:
   (a) Compute $\mathsf{Q} \leftarrow \mathsf{Query}(\mathsf{Inp})$.
   (b) Send ($\mathsf{Init}, \mathsf{sid}, \mathsf{Q}$) to $\mathcal{S}$.
2. Receive a response ($\mathsf{Response}, \mathsf{sid}, \mathsf{R}$) from $\mathcal{S}$:
   (a) Store the tuple ($\mathsf{sid}, \mathsf{Q}, \mathsf{R}$) and send ($\mathsf{sid}, \mathsf{R}$) to the client.
   (b) Send ($\mathsf{sid}, |\mathsf{Q}|, |\mathsf{R}|$) to the adversary.

Share/Encryption Phase:

1. On input ($\mathsf{Share}, \mathsf{sid}, \mathsf{mode}$) from the client, retrieve ($\mathsf{sid}, \mathsf{Q}, \mathsf{R}$) from memory.
2. If $\mathsf{mode} = \mathsf{fhe}$, then:
   (a) Encrypt each bit R_j of the response R to a ciphertext $\mathsf{Enc}_{\mathsf{pk}_{\mathsf{fhe}}}(\mathsf{R}_j)$.
   (b) Send ($\mathsf{sid}, \mathsf{Enc}_{\mathsf{pk}_{\mathsf{fhe}}}(\mathsf{R}_j)$) to $\mathcal{N}_1$. Also, send ($\mathsf{sid}, \mathsf{Enc}_{\mathsf{pk}_{\mathsf{fhe}}}(\mathsf{R}_j)$) to the client.
3. If $\mathsf{mode} = \mathsf{mpc}$, then:
   (a) For each bit R_j of the response R, generate secret shares $[\![\mathsf{R}_j]\!]_i$ with threshold T for node $\mathcal{N}_i$, where $i \in [1, n]$.
   (b) Send ($\mathsf{sid}, [\![\mathsf{R}_j]\!]_i$) to $\mathcal{N}_i$ for $i \in [1, n]$.

Fig. 1: Ideal Functionality $\mathcal{F}_{\text{TLShare}}$ between a server $\mathcal{S}$, a client, an attestor, and nodes $\mathcal{N}_i$ for $i \geq n$.

data transmitted over standard TLS connections to be consumed by secure multiparty computation (MPC) or fully homomorphic encryption (FHE) systems without any changes from the TLS server. This enables authenticated inputs in secure computation.

We present the ideal functionality of TLShare $\mathcal{F}_{\text{TLShare}}$ in Fig. 1, which defines the interactions between a client, an attestor, a TLS server, and a set of private computation nodes $\mathcal{N}_1, \ldots, \mathcal{N}_n$. We adopt the standard assumption used in prior work [98,18,90], namely that the attestor does not collude with the client. The functionality $\mathcal{F}_{\text{TLShare}}$ has two phases: 1) the *TLS Phase* and 2) the *Share/Encryption Phase*. The former involves the TLS handshake, the client's query to the TLS server, and the server's response, while the latter transforms the server's response either to secret shares or to FHE ciphertexts along with a proof of provenance. We introduce three novel protocols for TLShare and support two modes of operation (i.e., MPC and FHE), with one protocol for MPC (Section 4) and two different protocols for FHE (Section 5). We target two scenarios that combine data from multiple TLS servers, which is not feasible with any related work [98,33,77,20,90,18,53]. We outline the two scenarios below and motivating applications in Section 6.

Single-client multiple-sources. In this case, a client aggregates data from various institutions (e.g., banks, tax authorities) through TLS into secure computation (such as assessing their financial risk). We demonstrate an overview of this scenario in Fig. 2.

Multiple-clients multiple-sources. In this case, multiple users contribute data from different sources into a joint secure computation. Our motivating application for this scenario is the Boston Women's Workforce Council (BWWC) study that measured wage gaps across demographic groups (e.g., race and gender, or inequality of opportunity for women and minority-owned small businesses) [24,52,62]. More details in Section 6.2.

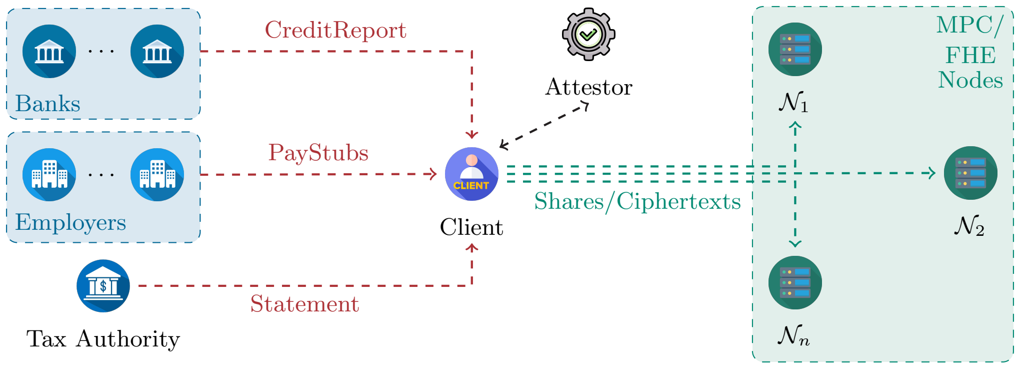

Fig. 2: **Overview.** The client wants to: a) import their data from multiple TLS connections into MPC/FHE nodes, b) prove to the nodes that the imported data is authentic and valid, and c) keep the data private. In the depicted example, the MPC/FHE nodes assess the client's financial risk by combining bank statements, credit reports, pay stubs, tax reports, citizenship cards, etc, all from different sources while keeping all the data private. Some applications may further require multiple provers to import their data into the same MPC/FHE session, enabling computation over the joint data.

3.2 Modes of Operation

TLShare supports both MPC and FHE secure computation techniques through three cryptographic protocols. Namely, TLShare supports MPC mode, FHE mode with non-interactive zero-knowledge proofs (NIZK), and FHE mode with interactive zero-knowledge proofs (IZK). The different modes introduce different performance tradeoffs, such as the presence or absence of preprocessing, which we explore in Section 7.4.

MPC Mode. In MPC mode, the client secret shares their authenticated TLS response among a network of MPC nodes. In the TLS phase, the client obtains an authenticated response R by initiating a TLS session with both the server and the attestor. To securely distribute R across a network of MPC nodes, the client transforms it into an additively homomorphic commitment. We instantiate this using Pedersen commitments, which are compatible with Pedersen verifiable secret sharing (PVSS). The Pedersen commitments of R serve as the base commitments for PVSS, allowing the response to be secret-shared among the MPC nodes. The attestor also signs the Pedersen commitment, enabling each MPC node to verify the authenticity of the committed message. This ensures that the value under commitment is exactly the authenticated TLS response generated in the protocol's initial phase.

FHE Mode with Non-Interactive ZK Proofs (NIZK). In this mode, the client encrypts the authenticated TLS response under a public FHE key and transmits the resulting ciphertext to a designated FHE node $\mathcal{N}_1$. The corresponding secret key is secret-shared among a set of nodes $\mathcal{N}_i$, enabling threshold decryption. Without loss of generality, the secret key may also be held by an external analyst interested in learning the final result of the computation. As in the MPC mode, the client first initiates a TLS session with the server and an attestor. This process yields signed commitments to the server's private response, which are shared between the client and the attestor. Using these commitments, the client encrypts the TLS response under the FHE public key and generates a non-interactive zero-knowledge (NIZK) argument that attests to the validity of the ciphertext. Specifically, the proof establishes that the ciphertext is well-formed and that it encrypts a value consistent with the previously committed TLS data. To achieve this, the client runs a novel NIZK argument of vector membership, which links the encrypted plaintext to the committed value. This proof guarantees that the ciphertext is a valid encryption of the committed response, thereby ensuring the consistency and correctness of the encryption.

FHE Mode with Interactive ZK Proofs (IZK). This variant supports FHE encryption based on LWE schemes over a prime modulus, with a correctness proof implemented using an interactive zero-knowledge (IZK) protocol built on IT-MACs [91]. Compared with the previous two approaches, the TLS phase is modified to produce IT-MACs derived from the server's private response. These act as authenticated witnesses in the zero-knowledge circuit that proves the correctness and validity of the LWE-based encryption. The proof circuit encodes the encryption process and links it directly to the TLS response obtained earlier. Finally, the resulting ciphertext is signed by the attestor, enabling the FHE node to confirm that it encodes the original TLS response.

Table 1: Feature comparison between the three proposed protocols during the Share/Encryption phase.

Protocol	**No Preprocessing**	**No CRS**	**Modulus Type**
MPC	✓	✓	Field
FHE with NIZK	✓	✗	Field/Ring
FHE with IZK	✗	✓	Field

Discussion. Table 1 shows a qualitative comparison between the proposed protocols. In MPC mode, the client does not require preprocessing or trusted setup, and there is only a one-shot communication with the MPC nodes during the share phase. The attestor is not involved after the TLS phase, leading to very low latency and bandwidth overhead. Meanwhile, both MPC and FHE with NIZK require the client and attestor to run a conversion protocol during the TLS phase to generate commitments of the response. In contrast, the FHE with IZK avoids this step by performing an interactive ZKP of valid encryption directly on the TLS-derived IT-MACs. During the share/encryption phase, the MPC and FHE with NIZK modes communicate only once with the secure computation nodes $\mathcal{N}_i$. At the same time, the FHE IZK-based protocol requires additional interaction with the attestor to generate the proof that ties the ciphertext to the TLS response.

Comparing the FHE mode protocols, the NIZK version protocol requires a single trusted setup and enables a fully non-interactive execution between the client and the FHE node $\mathcal{N}_1$, shifting proof verification entirely onto the FHE node and not involving the attestor after the TLS phase. In contrast, the FHE with IZK protocol eliminates trusted setup but requires preprocessing between the client and attestor during the encryption phase. Overall, the FHE with IZK achieves faster computation time during the encryption phase but is limited to prime-field LWE schemes, whereas the FHE with NIZK method supports ring-based LWE schemes.

4 MPC Mode Protocol

In TLShare's MPC mode, our goal is to ensure that the authenticated response obtained from a TLS session is not merely shared between the client and attestor but rather securely distributed as verifiable shares to a network of MPC nodes. This distinction is important: while the client and attestor both share verified shares of the TLS response, private computation over that data must be carried out by the MPC nodes, who must receive a different set of shares of the same underlying TLS response.

Garble-then-Prove (GtP) [90] produces authenticated Pedersen commitments to the TLS response, which are verified by the attestor and returned to the client. To enable the client to convince other parties of the authenticity of a TLS payload, our attestor produces signed commitments. However, this is not enough to allow a distributed MPC network to verify and consume the data securely for private computation. We unlock this functionality in TLShare with a construction that transfers the commitments and the corresponding secret shares directly to the MPC nodes in a way that maintains authenticity. We build on this insight by composing two cryptographic tools that share a common foundation: 1) Pedersen commitments, which provide authenticated commitments to the TLS response; and 2) Pedersen verifiable secret sharing (PVSS), which supports verifiability of shares under Pedersen commitments. The full protocol is described in Fig. 3.

The protocol is divided into two phases: *TLS phase* and *share/encryption phase*. In the TLS phase, the client and the attestor jointly execute the TLS handshake with the server and submit the query, resulting in the server's response committed with an additive homomorphic commitment scheme. These steps follow the Π_{AuthData} protocol from GtP, instantiated with Pedersen commitments. Both the client and attestor hold commitments to each bit of the TLS response R and, as shown in step 2 of Fig. 3, the attestor signs these commitments, thereby certifying their provenance and binding them to the TLS payload. The only

TLShare: MPC mode using PVSS

This protocol runs between a server $\mathcal{S}$, a client, an attestor, and an MPC network of n nodes $\mathcal{N}_1, \ldots, \mathcal{N}_n$. The server $\mathcal{S}$ behaves the same as in a standard TLS connection. $C = g^{\mu} \cdot h^{r}$ is a Pedersen commitment to μ for generators g and h and random r [70].

– **Public Parameters:** a query template (Query) for the server $\mathcal{S}$ (e.g., a `GET` request) and a threshold T for Shamir secret sharing.
– **Server $\mathcal{S}$:** has a secret key ($sk_{\mathcal{S}}$) for a signature scheme (e.g., ECDSA), a certificate $\mathsf{cert}_{\mathcal{S}}$, and hosts and operates a public website.
– **Client:** has a private input Inp (e.g., a password).
– **Attestor:** owns a key pair $(pk_{\mathcal{A}}, sk_{\mathcal{A}})$ to a signature scheme (e.g., for ECDSA).
– **MPC Node $\mathcal{N}_i$ (for $i \in [1, n]$):** has no input.

TLS Phase (Handshake, Query, and Response):

1. The client, $\mathcal{S}$, and the attestor run $\Pi_{\mathsf{AuthData}}(sk_{\mathcal{S}}, \mathsf{cert}_{\mathcal{S}}, \mathsf{Inp}, \mathsf{Query})$ instantiated with Pedersen commitments over group $\mathbb{G}$, i.e., converting the IT-MACs of $[\![\mathsf{Q}]\!]$ and $[\![\mathsf{R}]\!]$ to commitments over them. Let C_j be a commitment to the j^{th} bit of R with randomness r_j. The client outputs $(\mathsf{sid}, [C_1, \ldots, C_\ell], \mathsf{Q}, \mathsf{R})$ and the attestor outputs $(\mathsf{sid}, [C_1, \ldots, C_\ell])$, where $\ell = |\mathsf{R}|$.
2. The attestor signs the committed response as $\sigma = \mathsf{Sign}(sk_{\mathcal{A}}, [C_1, \ldots, C_\ell])$ and sends it to the client.

Share/Encryption Phase:

3. The client broadcasts $(\sigma, [C_1, \ldots, C_\ell])$ to all nodes $\mathcal{N}_i$.
4. The nodes verify the signature $\mathsf{Verify}(pk_{\mathcal{A}}, [C_1, \ldots, C_\ell], \sigma)$ and abort in case it fails.
5. For $j \in [1, \ell]$, take each bit of the response R_j, its corresponding commitment C_j with randomness r_j, and run the following:
   (a) The client selects two random polynomials $F(x)$ and $G(x)$ over $\mathbb{Z}_q$ of degree at most $\mathsf{T} - 1$ such that:
   $$F(x) = \mathsf{R}_j + \alpha_1 x + \alpha_2 x^2 + \cdots + \alpha_{\mathsf{T}-1} x^{\mathsf{T}-1} \quad \text{and} \quad G(x) = r_j + \beta_1 x + \beta_2 x^2 + \cdots + \beta_{\mathsf{T}-1} x^{\mathsf{T}-1},$$
   where $\mathsf{R}_j = F(0)$ and $r_j = G(0)$ are the secrets. Recall that R_j is the message and r_j is the randomness used in the commitment C_j. For each node $\mathcal{N}_i$ ($i \in [1, n]$), the client computes the shares $[\![\mathsf{R}_j]\!]_i = F(i)$ and $[\![r_j]\!]_i = G(i)$.
   (b) The client computes commitments to the coefficients as $g^{\alpha_l} \cdot h^{\beta_l}$ for $l \in \{1, \ldots, \mathsf{T} - 1\}$, and broadcasts them to all nodes.
   (c) The client sends the pair $([\![\mathsf{R}_j]\!]_i, [\![r_j]\!]_i)$ privately to each node $\mathcal{N}_i$.
   (d) Each $\mathcal{N}_i$ verifies the correctness of its share as $g^{[\![\mathsf{R}_j]\!]_i} \cdot h^{[\![r_j]\!]_i} = C_j \times \prod_{l=1}^{\mathsf{T}-1} \left(g^{\alpha_l} \cdot h^{\beta_l}\right)^{(i^l)}$. Abort in case of failure.

Fig. 3: TLShare protocol for $\mathsf{mode} = \mathsf{mpc}$.

communication in the share phase is between the client and the MPC nodes, as the attestor is no longer involved. The client starts (Fig. 3 step 3) by distributing the signed commitments to all MPC nodes, allowing them to verify the provenance of the committed data (step 4). Then, the Pedersen commitments generated during the TLS phase are used as the commitments of the secrets in the PVSS scheme. We refer to these as the *base commitments*. Specifically, treating each TLS bit commitment $C_j = g^{\mathsf{R}_j} \cdot h^{r_j}$ as the base commitment, the client creates Shamir secret-sharing polynomials $F(\cdot)$ and $G(\cdot)$ for both R_j and its randomness r_j, respectively in steps 5.*a*) of Fig. 3. Then, at steps 5.*b*) and 5.*c*), the client generates and sends corresponding commitments of the polynomial coefficients $g^{\alpha_l} \cdot h^{\beta_l}$ and shares $([\![\mathsf{R}_j]\!]_i, [\![r_j]\!]_i)$ to the nodes. Finally, using these commitments, each MPC node is able to locally verify its share by computing:

$$g^{[\![\mathsf{R}_j]\!]_i} \cdot h^{[\![r_j]\!]_i} = C_j \times \prod_{l=1}^{\mathsf{T}-1} \left(g^{\alpha_l} \cdot h^{\beta_l}\right)^{(i^l)},$$

where α_l and β_l are the coefficients of the Shamir polynomials. This expression ensures that each share is consistent with the attested commitment C_j. Notably, authenticated broadcast channels are used to ensure global consistency across nodes.

TLShare: FHE mode with Non-Interactive Zero-Knowledge (NIZK) Proofs

This protocol runs between $\mathcal{S}$, a client, an attestor, and FHE node $\mathcal{N}_1$. $\mathcal{S}$ behaves the same as in a standard TLS connection. The protocol requires a common reference string (CRS) computed in a setup phase by a trusted third party and accessible to both the client and $\mathcal{N}_1$. $\hat{V}$ is a vector commitment as in Eq. (1), and $\hat{C}$ is a Pedersen commitment [70], over the second source group $\hat{\mathbb{G}}$ of pairing $e : \mathbb{G} \times \hat{\mathbb{G}} \rightarrow \mathbb{G}_T$.

– **Public Parameters:** a query template (Query) for the server $\mathcal{S}$ (e.g., a `GET` request).
– **Server $\mathcal{S}$:** has a secret key ($sk_{\mathcal{S}}$) for a signature scheme (e.g., ECDSA), a certificate $\mathsf{cert}_{\mathcal{S}}$, and hosts and operates a public website.
– **Client:** has a private input Inp (e.g., a password).
– **Attestor:** owns a key pair $(pk_{\mathcal{A}}, sk_{\mathcal{A}})$ to a signature scheme (e.g., for ECDSA).
– **FHE Node** $\mathcal{N}_1$ has no input.

- -

TLS Phase (Handshake, Query, and Response):

1. The client, $\mathcal{S}$, and attestor run $\Pi_{\mathsf{AuthData}}(sk_{\mathcal{S}}, \mathsf{cert}_{\mathcal{S}}, \mathsf{Inp}, \mathsf{Query})$ instantiated with Pedersen commitments over group $\hat{\mathbb{G}}$, i.e., converting the IT-MACs of $[\![\mathsf{Q}]\!]$ and $[\![\mathsf{R}]\!]$ to commitments over them. Let $\hat{C}_j$ be a commitment to the j^{th} bit of R with randomness r_j. The client outputs $(\mathsf{sid}, [\hat{C}_1, \ldots, \hat{C}_\ell], \mathsf{Q}, \mathsf{R})$ and the attestor outputs $(\mathsf{sid}, [\hat{C}_1, \ldots, \hat{C}_\ell])$, where $\ell := |\mathsf{R}|$.
2. The attestor signs the committed response as $\sigma := \mathsf{Sign}(sk_{\mathcal{A}}, [\hat{C}_1, \ldots, \hat{C}_\ell])$ and sends it to the client.

Share/Encryption Phase:

3. The client sends $(\sigma, [\hat{C}_1, \ldots, \hat{C}_\ell])$ to $\mathcal{N}_1$.
4. $\mathcal{N}_1$ verifies the signature $\mathsf{Verify}(pk_{\mathcal{A}}, [\hat{C}_1, \ldots, \hat{C}_\ell], \sigma)$ and aborts in case it fails.
5. For each bit of the response R_j, $j \in [1, \ell]$:
   (a) The client locally runs the CGGI proving algorithm to obtain $(\pi_{\mathsf{CGGI}}, \hat{V}_j, \gamma_j) \leftarrow \mathsf{CGGI.Prove}_{\mathsf{pp}}(\mathsf{R}_j)$.
   (b) Then, the client locally sets $\mathsf{com} = (\hat{C}_j, \hat{V}_j) \in \hat{\mathbb{G}}^2$ and the opening information $\mathsf{aux} = (r_j, \gamma_j) \in \mathbb{Z}_p^2$. Finally, the client locally runs $\pi_{\mathsf{VM}} \leftarrow \mathsf{VM.Prove}_{\mathsf{pp}}(\mathsf{com}, k, l, (\mathsf{R}_j, \boldsymbol{w}_j, \mathsf{aux}))$ for index k, bitlength l, binary vector $\boldsymbol{w}_j$ containing the response R_j, encryption randomness and noise, and π_{VM} given by Eq. (2).
   (c) The client sends the tuple $(\hat{V}_j, \pi_{\mathsf{VM}}, \pi_{\mathsf{CGGI}}, k, l)$ to the FHE node $\mathcal{N}_1$ for verification.
   (d) $\mathcal{N}_1$ runs the verification algorithm $\mathsf{CGGI.Verify}_{\mathsf{pp}}$ to check the validity of π_{CGGI}; and finally runs the verification algorithm $\mathsf{VM.Verify}_{\mathsf{pp}}(\mathsf{com}, k, l, \pi_{\mathsf{VM}})$ by checking Eq. (3). $\mathcal{N}_1$ aborts if any of the above checks fail.

Fig. 4: TLShare protocol with $\mathsf{mode} = \mathsf{fhe}$ with NIZK for the CGGI scheme.

Theorem 1. *Under the decisional Diffie–Hellman (DDH) assumption, the MPC mode protocol from Fig. 3 securely realizes the functionality $\mathcal{F}_{TLShare}$ (Fig. 1) in the stand-alone model in the $\mathcal{F}_{\mathsf{AuthData}}$-hybrid world.*

Given a functionality $\mathcal{F}_{\mathsf{AuthData}}$ for secure authenticated data [90], we show that the MPC mode protocol realizes $\mathcal{F}_{\text{TLShare}}$ for malicious adversaries as stated in Theorem 1 in Appendix A.

5 FHE Mode Protocols

In this section, we describe how to generate an FHE ciphertext of a payload originating from a TLS connection in a verifiable and privacy-preserving manner. We present two solutions that can be instantiated with two popular types of FHE schemes: CGGI [22] and the LPR-style variant of Regev's scheme [74]. As in the MPC mode, we leverage tools from the Garble-then-Prove (GtP) [90] to produce authenticated shares of the TLS payload.

5.1 FHE Mode with Non-Interactive ZK Proofs (NIZK)

We present a protocol that generates an FHE ciphertext of a payload extracted from a TLS connection, instantiated using the CGGI scheme [22]. We design the protocol to ensure two key security guarantees: 1) the validity of the CGGI resulting ciphertext, and 2) the consistency between the encrypted message and the corresponding payload from the TLS response R, as authenticated by the TLS phase.

To achieve these goals, our TLShare FHE with NIZK protocol integrates pairing-based vector commitments (see Section 2.4) as these can be used to prove that a tuple $(a_1, \ldots, a_n, b)$ is a valid CGGI encryption of a plaintext μ under an encryption key [55,35,14]. This construction exploits the algebraic structure of vector commitments to prove inner product relations between a secret vector (containing the message, encryption randomness, and noise) and a public vector (containing the public key and output ciphertext). Beyond proving the correctness of encryption, we also establish the consistency between the encrypted value and the TLS payload R. To this end, both the client and the attestor receive Pedersen commitments to each bit of R. However, this design introduces a technical challenge: the commitments used to encode the TLS payload are not directly compatible with the vector commitment constructions (Eq. (1)) required to prove encryption validity.

To bridge this gap, we introduce a novel cryptographic component that we call *proof of vector membership*. This technique enables us to use Pedersen committed values with the NIZK proof of valid encryption, proving end-to-end provenance for FHE. Specifically, we allow the client to prove that the binary plaintext μ of a CGGI ciphertext is the same as the response R committed using Pedersen commitments, without revealing R or μ. Next, we describe the novel construction of our vector membership proof.

NIZK Proof of Vector Membership (VM). To enable composability between our NIZK for vector membership and the NIZK for CGGI encryption, the generators of Pedersen commitments are $\hat{g}$ and $\hat{g}_1$, derived by the CRS, see Section 2.4. Consider a Pedersen commitment $\hat{C} = \hat{g}^r \cdot \hat{g}_1^{\mu}$ of a message μ in the second source group $\hat{\mathbb{G}}$ of a pairing, where μ has a binary expansion given by $\mu = (\mu_1, \ldots, \mu_\ell)$. Additionally, let $\boldsymbol{w} = (w_1, \ldots, w_n)$ be a binary vector containing μ as a contiguous segment. More formally, for $i \in [1, \ell]$, $\mu_i = w_{k+i}$, for some index k. Finally, let $\hat{V}$ be a vector commitment to the binary vector $\boldsymbol{w}$ as $\hat{g}^{\gamma} \cdot \prod_{j=1}^{n} \hat{g}_j^{w_j}$ from Eq. (1).

In our NIZK proof, the prover aims to convince the verifier that the binary representation of μ (i.e., $\mu_1, \ldots, \mu_\ell$) corresponds to a contiguous segment within the binary vector $\boldsymbol{w}$ committed in the vector commitment $\hat{V}$, while keeping μ and $\boldsymbol{w}$ private. In other words, the prover wants to convince the verifier that μ is a member of $\boldsymbol{w}$, with $\hat{C}$ and $\hat{V}$ being their public commitments, respectively. More formally, we define the vector membership (VM) relation as

$$\mathcal{R}_{\mathsf{pp}} = \Big\{ \big((\hat{C}, \hat{V}, k, l), (r, \gamma, \mu, \boldsymbol{w})\big) \;\Big|\; \hat{C} = \hat{g}^r \cdot \hat{g}_1^{\mu} \quad \wedge \quad \hat{V} = \hat{g}^{\gamma} \cdot \textstyle\prod_{j=1}^{n} \hat{g}_j^{w_j} \quad \wedge \quad \forall i \in [1, l],\ \mu_i = w_{i+k} \quad \wedge \quad \mu = \sum_{i=1}^{l} 2^{i-1} \cdot \mu_i \Big\}$$

where the public parameters pp denotes the CRS containing the commitment key $(\hat{g}, \{\hat{g}_i\}_{i=1}^{n})$ and the pairing groups $(\mathbb{G}, \hat{\mathbb{G}}, \mathbb{G}_T)$. The NIZK argument for the above VM relation $\mathcal{R}_{\mathsf{pp}}$ consists of the following algorithms:

– $\mathsf{VM.GenCRS}(1^\lambda, 1^n)$: on input a security parameter λ and the maximal bit length n, proceed as follows. Choose asymmetric bilinear groups $(\mathbb{G}, \hat{\mathbb{G}}, \mathbb{G}_T)$ of prime order $p > 2^{\ell(\lambda)}$, for some polynomial function $\ell : \mathbb{N} \to \mathbb{N}$, and sample $g \xleftarrow{\$} \mathbb{G}$, $\hat{g} \xleftarrow{\$} \hat{\mathbb{G}}$. The public parameters are defined as: $\mathsf{pp} = ((\mathbb{G}, \hat{\mathbb{G}}, \mathbb{G}_T),\ g,\ \hat{g},\ \{g_i\}_{i \in [1,2n] \setminus \{n+1\}},\ \{\hat{g}_i\}_{i \in [1,n]})$.

– $\mathsf{VM.Com}_{\mathsf{pp}}(\mu, (\hat{V}, \gamma))$: on input $\mu \in \mathbb{Z}_p$ and $\hat{V} \in \mathbb{G}$, the prover chooses a random $r \xleftarrow{\$} \mathbb{Z}_p$ and computes the commitment $\hat{C} = \hat{g}^r \cdot \hat{g}_1^{\mu} \in \hat{\mathbb{G}}$. Return $\mathsf{com} = (\hat{C}, \hat{V}) \in \hat{\mathbb{G}}^2$ and the opening information $\mathsf{aux} = (r, \gamma) \in \mathbb{Z}_p^2$.

– $\mathsf{VM.Prove}_{\mathsf{pp}}((\mathsf{com}, k, l), (\mu, \boldsymbol{w}, \mathsf{aux}))$: given $\mathsf{com} = (\hat{C}, \hat{V}) \in \hat{\mathbb{G}}^2$, indexes k and l, and witness information (μ, w, aux), where $\mathsf{aux} = (r, \gamma)$ and μ has binary expansion $(\mu_1, \ldots, \mu_l)$ and $\boldsymbol{w}$ is a binary vector, compute the following proof π:
$$g_n^{-r} \cdot \textstyle\prod_{i \in L} \big(g_{n+1-i}^{\gamma} \cdot \prod_{j \in [1,n] \setminus \{i\}} g_{n+1+j-i}^{w_j}\big)^{2^{i-1-k}}, \tag{2}$$
where $L = \{k+1, \ldots, k+l\}$.

– $\mathsf{VM.Verify}_{\mathsf{pp}}((\mathsf{com}, k, l), \pi)$: given $\mathsf{com} = (\hat{C}, \hat{V}) \in \hat{\mathbb{G}}^2$ and proof π, return True if and only if
$$e\big(\textstyle\prod_{i \in L} g_{n+1-i}^{2^{i-1-k}}, \hat{V}\big) / e(g_n, \hat{C}) = e(\pi, \hat{g})\,. \tag{3}$$

TLShare: FHE mode with Interactive Zero-Knowledge (IZK) Proofs

This protocol runs between a server $\mathcal{S}$, a client, an attestor, and FHE node $\mathcal{N}_1$. $\mathcal{S}$ behaves the same as in a standard TLS connection. We use $\langle\langle x \rangle\rangle$ to denote authenticated shares (i.e., IT-MACs) of x, as defined in Section 2.3.

- **Public Parameters:** a query template (Query) for the server $\mathcal{S}$ (e.g., a `GET` request). An LPR-style Regev public key $(\boldsymbol{A}, \boldsymbol{y})$ and a plaintext modulus p as defined in Section 2.2.
- **Server $\mathcal{S}$:** has a secret key ($sk_{\mathcal{S}}$) for a signature scheme (e.g., ECDSA), a certificate $\mathsf{cert}_{\mathcal{S}}$, and hosts and operates a public website.
- **Client:** has a private input Inp (e.g., a password).
- **Attestor:** owns a key pair $(pk_{\mathcal{A}}, sk_{\mathcal{A}})$ to a signature scheme (e.g., for ECDSA).
- **FHE Node** $\mathcal{N}_1$ has no input.

TLS Phase (Handshake, Query, and Response):

1. The client, $\mathcal{S}$, and attestor run $\Pi_{\mathsf{AuthData}}(sk_{\mathcal{S}}, \mathsf{cert}_{\mathcal{S}}, \mathsf{Inp}, \mathsf{Query})$ and retrieve the IT-MACs of the query and response as $\langle\langle \mathsf{Q} \rangle\rangle$ and $\langle\langle \mathsf{R} \rangle\rangle = [\langle\langle \mathsf{R}_1 \rangle\rangle, \ldots, \langle\langle \mathsf{R}_\ell \rangle\rangle]$. We denote by ℓ, the length of the response, i.e., $\ell := |\mathsf{R}|$.

Share/Encryption Phase:

2. For each bit of the response R_j, $j \in [1, \ell]$:
   (a) The client locally generates a binary randomness vector, $\boldsymbol{r} \xleftarrow{\$} \{0,1\}^n$; and noise samples $\boldsymbol{e} \xleftarrow{\$} \chi^n$, $e_0 \xleftarrow{\$} \chi$, where χ is a centered discrete Gaussian with standard deviation stdev over $\mathbb{Z}_q$.
   (b) The client and the attestor generate IT-MACs of the secret inputs of the client as $\langle\langle \boldsymbol{r} \rangle\rangle, \langle\langle \boldsymbol{e} \rangle\rangle, \langle\langle e_0 \rangle\rangle$.
   (c) The client and the attestor run an IZK using QuickSilver with witness $(\langle\langle \boldsymbol{r} \rangle\rangle, \langle\langle \mathsf{R}_j \rangle\rangle, \langle\langle \boldsymbol{e} \rangle\rangle, \langle\langle e_0 \rangle\rangle)$ and the LWE public key $(\boldsymbol{A}, \boldsymbol{y})$ as the ZKP public statement on the following circuit:
      i. Check $0 \le \langle\langle \mathsf{R}_j \rangle\rangle < p$ and abort in case of failure.
      ii. $\forall i \in [1, n]$ compute $\langle\langle r_i \rangle\rangle \cdot (1 - \langle\langle r_i \rangle\rangle)$, reveal its value, and abort if not zero.
      iii. $\forall i \in [0, n]$ check that $|\langle\langle e_i \rangle\rangle| \le 6 \cdot \mathsf{stdev}$ and abort in case of failure.
      iv. Compute the ciphertext $(\boldsymbol{a}, b)$ as $(\boldsymbol{A} \cdot \langle\langle \boldsymbol{r} \rangle\rangle + \langle\langle \boldsymbol{e} \rangle\rangle, \boldsymbol{y}^\top \cdot \langle\langle \boldsymbol{r} \rangle\rangle + \langle\langle e_0 \rangle\rangle + \lfloor \langle\langle \mathsf{R}_j \rangle\rangle \cdot \Delta \rceil) \bmod q$, following Eq. (4).
      v. Reveal the ciphertext $(\boldsymbol{a}, b)$ to both client and attestor.
   (d) The attestor signs the ciphertext $(\boldsymbol{a}, b)$, i.e., $\mathsf{Sign}(sk_{\mathcal{A}}, (\boldsymbol{a}, b))$ and sends the signature to the client.
   (e) The client verifies the signature and sends it along with the ciphertext to node $\mathcal{N}_1$.
   (f) The node $\mathcal{N}_1$ verifies the signature using the attestor's public key, $pk_{\mathcal{A}}$.

Fig. 5: TLShare protocol with $\mathsf{mode} = \mathsf{fhe}$ with IZK.

A detailed correctness proof of the VM NIZK definition can be found in Appendix B, and a security argument for the vector membership construction is given in Appendix C.

TLShare for CGGI encryption. We now combine the NIZK proof of vector membership with the NIZK proof of valid CGGI encryption [35,54,14], and the Pedersen commitments from the TLShare TLS phase. As a result, we perform a verifiable FHE encryption of a plaintext that can be provably linked to a TLS payload. We adopt the public-key NIZK argument for valid CGGI encryption as described by Bootland et al. [14, Section 7.6.3]. Let $(\mathsf{CGGI.Prove}_{\mathsf{pp}}, \mathsf{CGGI.Verify}_{\mathsf{pp}})$ denote the proving and verification algorithms of this NIZK proof. Our full TLShare protocol combining the VM and the CGGI NIZK proofs is summarized in Fig. 4.

In the TLS phase, the server, client, and attestor jointly run Π_{AuthData} from GtP [90] instantiated with Pedersen commitments in step 1 of Fig. 4. This results in both the client and the attestor receiving Pedersen commitments to the bits of the TLS payload R. To enable external verification of the CGGI encryption, the attestor signs the committed response in a signature σ and forwards it to the client in step 2. In step 3, the client sends σ along with the commitments $[\hat{C}_1, \ldots, \hat{C}_\ell]$ to the node, who, in turn, verifies it in step 4.

The encryption phase involves only one-way communication from the client to the FHE node. At this stage, the client locally generates the required proofs: the NIZK argument for valid CGGI encryption (step 5.a) and the NIZK argument of vector membership (step 5.b). Specifically, for each bit j of the payload R_j, the client performs the following steps. It runs the CGGI proving algorithm to obtain

$(\pi_{\mathsf{CGGI}}, \hat{V}_j, \gamma_j) \leftarrow \mathsf{CGGI.Prove}_{\mathsf{pp}}(\mathsf{R}_j)$ and takes the commitment $\hat{C}_j$ corresponding to R_j and the vector commitment $\hat{V}_j$ from the previous step. Then, the client uses them as the commitments $\mathsf{com} = (\hat{C}_j, \hat{V}_j)$ from VM.Com and the randomness of the commitments as $\mathsf{aux} = (r_j, \gamma_j)$. In other words, the NIZK argument for valid CGGI encryption [14] internally commits to the vector $\boldsymbol{w}$ through the value $\hat{V}$ during the proof generation. Therefore, we reuse the vector commitment $\hat{V}$ as part of the NIZK argument for vector membership. The client continues in step 5.b of Fig. 4 by generating the proof of vector membership via $\pi_{\mathsf{VM}} \leftarrow \mathsf{VM.Prove}_{\mathsf{pp}}(\mathsf{com}, k, l, (\mathsf{R}_j, \boldsymbol{w}_j, \mathsf{aux}))$. In step 5.$c$), the client sends the tuple $(\hat{V}_j, \pi_{\mathsf{VM}}, \pi_{\mathsf{CGGI}}, k, l)$ to the FHE node for verification. Finally, in step 5.d), the FHE node locally runs the verification algorithms (i.e., CGGI.Verify and VM.Verify) for the proofs sent by the client.

We provide a security argument for the TLShare protocol with $\mathsf{mode} = \mathsf{fhe}$ with NIZK for the CGGI scheme in Appendix D.

5.2 FHE Mode with Interactive ZK Proofs (IZK)

In this section, we describe an interactive zero-knowledge (IZK) approach to securely encrypt a TLS payload message R into an FHE ciphertext. We focus on an LPR-style variant of Regev's LWE scheme [71] with ciphertext and plaintext moduli q and p, and public key encryption of a message μ:

$$(\boldsymbol{a}, b) = (\boldsymbol{A}\boldsymbol{r} + \boldsymbol{e}, \boldsymbol{y}^\top \boldsymbol{r} + e_0 + \lfloor \mu \cdot \Delta \rceil) \bmod q, \tag{4}$$

under public key $(\boldsymbol{A}, \boldsymbol{y})$, where $\Delta = q/p$ is the scaling factor between plaintext and ciphertext spaces, $\boldsymbol{r} \overset{\$}{\leftarrow} \{0,1\}^n$ is a binary randomness vector, $\boldsymbol{e} \overset{\$}{\leftarrow} \chi^n$, $e_0 \overset{\$}{\leftarrow} \chi$ are noise samples, and χ is a centered discrete Gaussian with standard deviation stdev over $\mathbb{Z}_q$.

At a high level, our FHE mode with IZK protocol derives authenticated shares (i.e., IT-MACs) of the server's private response from the TLS phase. Then, we compose it with an IT-MAC-based IZK protocol to prove the validity of an LWE encryption circuit. We use the efficient QuickSilver protocol [91] together with the GtP framework to achieve a constant number of rounds and a communication overhead near one field element per multiplication gate [91]. Our full IZK protocol is summarized in Fig. 5.

Recall, in the MPC mode and the FHE mode with NIZK, the Π_{AuthData} [90, Figs. 17–18] in the TLS phase outputs Pedersen commitments of the TLS payload R. These Pedersen commitments are computed over IT-MAC shares of R through a conversion protocol [90, Fig. 19]. In our protocol, we avoid this conversion and directly extract the IT-MACs of R (i.e., $\langle\!\langle \mathsf{R} \rangle\!\rangle$) from GtP, as outlined in step 1 of Fig. 5. Next, we use $\langle\!\langle \mathsf{R} \rangle\!\rangle$ as part of the private witness of a circuit that proves the validity of LWE encryption in zero-knowledge.

For each bit of the response message R, we proceed as follows. In step 2.a, the client locally generates any randomness needed for Eq. (4), specifically, $\boldsymbol{r}$, $\boldsymbol{e}$, and e_0. Next, in 2.b, the client and the attestor generate authenticated shares [91] of the aforementioned elements $\langle\!\langle \boldsymbol{r} \rangle\!\rangle$, $\langle\!\langle \boldsymbol{e} \rangle\!\rangle$, $\langle\!\langle e_0 \rangle\!\rangle$. To prove the validity of encryption $(\boldsymbol{a}, b)$ from Eq. (4), our circuit takes $(\langle\!\langle \boldsymbol{r} \rangle\!\rangle, \langle\!\langle \mathsf{R}_j \rangle\!\rangle, \langle\!\langle \boldsymbol{e} \rangle\!\rangle, \langle\!\langle e_0 \rangle\!\rangle)$ (generated in the two previous steps) as witness and checks the following in step 2.c:

- $\langle\!\langle \mathsf{R}_j \rangle\!\rangle$ lies in the valid plaintext range $[0, p)$.
- Each coordinate of $\langle\!\langle \boldsymbol{r} \rangle\!\rangle$ is binary. Notably, the expression $\langle\!\langle r_i \rangle\!\rangle \cdot (1 - \langle\!\langle r_i \rangle\!\rangle)$ yields 0 if and only if $r \in \{0,1\}$.
- All noise terms e_i satisfy $|\langle\!\langle e_i \rangle\!\rangle| \le 6 \cdot \mathsf{stdev}$ for $i = [0, n]$.
- The ciphertext $(\boldsymbol{a}, b)$ matches Eq. (4).

More formally, we show membership of $(\boldsymbol{a}, b)$ in the relation:

$$\mathcal{R} = \Big\{ \big((\boldsymbol{a}, \boldsymbol{b}), (\boldsymbol{r}, \mathsf{R}_j, \boldsymbol{e}, e_0)\big) \;\Big|\; \mathsf{R}_j \in \mathbb{Z}_p \quad \wedge \quad \boldsymbol{r} \in \{0,1\}^n \quad \wedge \\ \forall i \in [0, n],\ |e_i| \le 6 \cdot \mathsf{stdev} \quad \wedge \quad \boldsymbol{a} = \boldsymbol{A}\boldsymbol{r} + \boldsymbol{e} \bmod q \qquad \wedge \quad b = \boldsymbol{y}^\top \boldsymbol{r} + e_0 + \lfloor \mathsf{R}_j \cdot \Delta \rceil \bmod q \Big\}.$$

Then, after verifying the validity of $(\boldsymbol{a}, b)$, the attestor signs the ciphertext and sends it to the client (step 2.d), which subsequently forwards it to the node $\mathcal{N}_1$ (step 2.e). Finally, the node verifies the signature, which guarantees that the value under the ciphertext comes from a TLS connection that the attestor verified.

We provide a security argument for the TLShare protocol with $\mathsf{mode} = \mathsf{fhe}$ with IZK in Appendix E.

6 Applications

6.1 Single-client, multiple-sources: Financial Risk Assessments

We consider a use case in which an individual client wishes to utilize a third-party risk assessment service to evaluate their financial circumstances by jointly analyzing data housed within multiple institutions, such as banks, employers, and tax authorities. One major implementation challenge associated with secure computation workflows for assessing financial risk [1] is determining which parties are responsible for coordinating interactions between the multiple institutions contributing data and the risk assessment service provider. The institutions may find it prohibitively expensive (or in conflict with the fact that they are competitors) to agree in advance to collaborate on such workflows. Even if these obstacles are not present, it is impractical for each participating institution to integrate with the centralized risk assessment service provider.

As illustrated in Fig. 2, our approach addresses this obstacle by enabling the client to perform this coordination independently, utilizing existing TLS servers from various institutions. With the help of an attestor, the downstream computation can be assured of the accuracy of the client-provided data. The analysis is then performed under MPC/FHE by combining the different encrypted inputs, comparing them to thresholds, and even evaluating them with machine learning models [37,69].

6.2 Multiple-clients, multiple-sources: Wage Equity Reporting

We consider the Boston Women's Workforce Council (BWWC) study that measures wage disparities across demographic groups (including race, gender, and level of seniority) within a metropolitan region [52,62]. This study is accomplished by executing an aggregation workflow on input data (provided directly by a large number of individual employer organizations) that contains wage information broken down by demographic attributes; the aggregated output is then disclosed in public reports [24]. Traditional approaches, such as centralized collection of the plaintext input data by a trusted third party, are not acceptable to data contributors due to the inherent risks (to both employees and employers) of data exposure. Employing MPC to minimize data exposure risks is viable [52] but involves manual data entry that may introduce errors without careful interface design, costly user training [73], and does not inherently provide mechanisms for verifying the accuracy of the data. These drawbacks limit the scale of such studies and can invite skepticism about data accuracy.

Our approach can mitigate these drawbacks by leveraging multiple clients, each independently collecting authenticated data in an automated and scalable manner from authoritative TLS endpoints (e.g., payroll APIs, employer HR systems, municipal databases, and so on) by working with an attestor. Finally, the sensitive data (e.g., reported compensation amounts, employment statuses, and demographic attributes) is verifiably imported into a downstream secure computation. The downstream analysis can compute statistics (e.g., histograms of compensation ranges broken down by demographic attributes or wage ratios between demographic groups) [52], perform regressions, or even train federated models [36,38], all without revealing any party's individual records.

7 Experimental Evaluation

We implemented TLShare in C++ and Rust, choosing libraries tailored to each protocol's cryptographic requirements.We modified the conversion protocol of GtP [90] and the RELIC [75] library to integrate both the Ed25519 curve and the pairing-friendly BLS12-446 curve, required for both the MPC mode and FHE mode with NIZK, respectively. We extended the TFHE-rs library [96] to include our vector membership NIZK proofs for valid ciphertexts. For the FHE mode with IZK, we relied on the QuickSilver protocol [91] and the EMP toolkit [86] to implement the circuit for proving the correctness of Regev encryption. Lastly, our generated ciphertexts are compatible with the Primus FHE library [71] for the downstream FHE computation, while for the downstream MPC, we used MP-SPDZ [49].

The experiments involve the following set of parties: the client, the attestor, and the secure computation nodes. The client and the attestor are placed in two different `c5.xlarge` instances, each with 4 vCPUs and

8 GBs of RAM. For the secure computation nodes, we used a `c5.9xlarge` instance with a total of 36 vCPUs and 72 GBs of RAM, and dedicated 4 vCPUs per node. We conducted experiments over two settings: a LAN with 1 Gbps bandwidth and near-zero latency, and a WAN with 100 Mbps bandwidth and 50 ms latency.

Table 2: Latency comparisons (in ms) between TLShare in MPC mode and Garble-Then-Prove over LAN and WAN for different payload sizes (bytes). We focus on the conversion protocols of the TLS Phase for both works.

Protocol	**Network**	**LAN**				**WAN**			
	Payload (B)	2^8	2^9	2^{10}	2^{11}	2^8	2^9	2^{10}	2^{11}
GtP [90]	**Client**	10.1	18.5	35.2	69.7	317.2	330.9	357.3	414.9
	Attestor	9.41	17.8	34.1	67.9	215.9	229.4	255.3	312.3
TLShare	**Client**	12.0	20.5	37.5	71.6	318.5	332.8	358.8	413.7
	Attestor	11.5	19.9	36.7	70.2	317.9	331.7	357.8	412.8

7.1 MPC Mode

In the MPC mode, the original GtP implementation uses OpenSSL's NIST P-256 curve for Pedersen commitments and lacks support for signing those commitments. We reimplemented the conversion protocol Π_{Conv} [90, Fig. 20] on the Ed25519 curve, using the RELIC library. Ed25519 provides comparable 128-bit security to P-256 while offering significantly faster operations and smaller point sizes [9]. We further integrated ECDSA signatures directly into the conversion process, enabling authenticated commitments.

In Table 2, we compare the latency in milliseconds of the TLS phase of GtP and TLShare in the MPC mode for different payload sizes, ranging from 256 to 2048 bytes. We batch the IT-MACs of the payload bits, resulting in 9 and 66 batches for the 256 and 2048 byte payloads, respectively. Observe that under LAN, for both frameworks, both the client and the attestor take almost the same time. Under WAN, however, the client in GtP takes about 100 ms more than the attestor, while in TLShare, their runtime is the same. The clients of both GtP and TLShare are also equivalent. This is because the additional cost of the attestor in the TLShare over GtP (i.e., around 100ms under WAN) is introduced by signing the commitments. However, this cost is negligible when viewed in the context of the full TLS exchange, as it unlocks the ability to perform secure computation on authenticated data.

Table 3: Latency evaluations (in ms) for the share phase of TLShare in the MPC mode under LAN and WAN for different numbers of parties n, thresholds T, and one batch.

Party	**Network**	**LAN**			**WAN**		
	(T, n)	$(2,3)$	$(3,5)$	$(4,7)$	$(2,3)$	$(3,5)$	$(4,7)$
Client	**Preprocessing**	0.182	0.271	0.337	0.187	0.254	0.336
	Secret Share	0.00052	0.00061	0.00079	0.00052	0.00049	0.0005
MPC Nodes	**Signature Ver.**	0.432	0.442	0.494	0.538	1.012	0.880
	Shares Ver.	0.212	0.248	0.325	0.215	0.267	0.451
All	**Computation**	0.826	0.961	1.155	0.940	1.533	1.667
	Total	1.245	1.501	1.730	101.62	102.57	102.47

During the Share phase, we use Pedersen verifiable secret sharing (PVSS) to distribute one secret at a time across different configurations of Shamir shares [25,3]. Importantly, this phase excludes the attestor: once the client has the signed commitments, it executes a single interaction with the MPC nodes. The client first generates PVSS parameters, then computes the shares for each MPC node. On the MPC nodes side, each party verifies the attestor's signature on the base Pedersen commitments and checks its consistency.

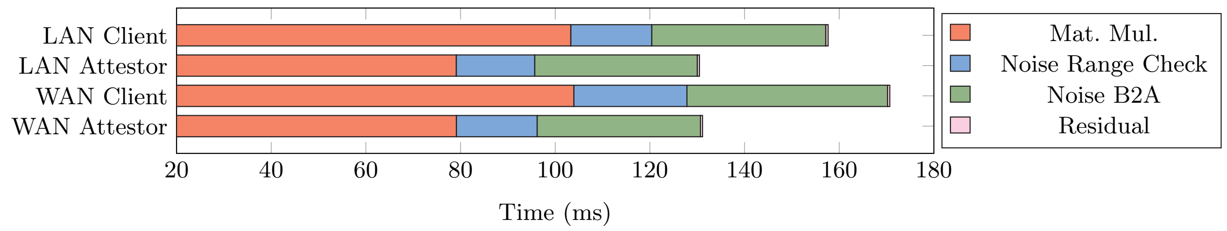

Fig. 6: Latency breakdown of total time (ms) per party (client and attestor) in LAN and WAN settings for the Encryption phase of the FHE mode with IZK. "Residual" includes a check verifying that the randomness of the encryption is a binary vector, an input range check asserting that the message μ is within the plaintext modulus, and the input conversion from binary to arithmetic (B2A).

We present a breakdown of these costs in Table 3, including preprocessing (PVSS parameter generation), share generation time, and node-side verification for different settings of threshold T and number of nodes n. The last row of Table 3 presents the end-to-end time, including network latency, while the row above it shows the local timings combined (i.e., without the network latency). The latter leads to a share phase faster than 2 ms and a bandwidth of 0.2 KB across the board. While the running time under LAN is dominated by local computation, the WAN results are primarily constrained by the network latency. Finally, the computation timing is linearly dependent on the number of commitments generated and verified, which is directly related to the threshold value T and not the number of nodes n.

Table 4: Latency evaluations (in ms) of the FHE mode of TLShare with NIZK over LAN and WAN between the client ($\mathcal{C}$), the attestor ($\mathcal{A}$), and the FHE node ($\mathcal{N}_1$) for a single batch.

Phase	**Network**	**LAN**			**WAN**		
	Party	$\mathcal{C}$	$\mathcal{A}$	$\mathcal{N}_1$	$\mathcal{C}$	$\mathcal{A}$	$\mathcal{N}_1$
TLS	**Conversion**	5.06	5.23	-	307.3	307.9	-
Enc.	**CGGI Proof**	609	-	137	607	-	98.0
	VM Proof	18	-	13	18	-	12
	Sig. Verification	-	-	0.56	-	-	0.56
	Total	800	-	151	1103	-	157

7.2 FHE Mode with NIZK

The vector commitments in TFHE-rs's NIZK proof of correct encryption [14, Section 7.6.3] are over the second source group $\hat{\mathbb{G}}$ of the BLS12-446 curve. For our vector membership proof, we reimplemented the GtP conversion protocol over $\hat{\mathbb{G}}$ to link the Pedersen commitments of the payload with these vector commitments using the RELIC library.

In Table 4, we report the timings for converting and encrypting a batch under the FHE mode with NIZK. Note that the NIZK-based protocol only interacts with the attestor during the TLS phase and with the FHE node during the encryption phase. In the latter, the bulk of the computation occurs within the client-side generation of the encryption correctness proof (i.e., $\mathsf{CGGI.Prove}$), which dominates the overall cost, while the NIZK proof of vector membership (i.e., $\mathsf{VM.Prove}$) adds minimal additional time. Indeed, the NIZK VM proof is 33$\times$ faster than the NIZK proof of valid encryption for the client (i.e., the prover) and over 8$\times$ faster for $\mathcal{N}_1$ (i.e., the verifier). The total bandwidth transferred between the client and the node is 92.4 KB. Finally, the last row in Table 4 shows the end-to-end time for the encryption phase between all parties, including the network latency.

7.3 FHE Mode with IZK

Last but not least, the FHE mode with IZK uses the Primus FHE parameter set [71]: dimension $n = 2048$, ciphertext modulus $q = 2^{61} - 1$, message modulus $p = 4$, Gaussian noise with $\mathtt{stdev} = 3.19 \times 2^7$, and a ternary secret key. The resulting circuit comprises approximately 148 million `AND` gates for binary checks (including message and noise range validations) and an arithmetic part that performs scalar matrix–vector multiplication and randomness checks. These checks require verifying, for each random component $\langle\langle r_i \rangle\rangle$ that $\langle\langle r_i \rangle\rangle \cdot (1 - \langle\langle r_i \rangle\rangle) = 0$ to confirm $\langle\langle r_i \rangle\rangle$ is binary.

The preprocessing in QuickSilver needed for our encryption circuit requires approximately 131 thousand edaBits [34]. We measured the performance of a single batch of 1.6 million edaBits, which takes about 40 seconds over a LAN and 118 seconds over a WAN, transferring approximately 122 MB of data. This step fully accounts for the offline work required by the IZK approach for the encryption phase. The performance of the online part of the encryption phase (see Fig. 6) is dominated by the matrix-scalar multiplication (orange bar), noise range checks (blue), and conversions from binary to arithmetic form (green). This multiplication is between the public key (a matrix) and the secret randomness binary vector $\boldsymbol{r}$. As a result, it does not require any communication, as experimentally verified in Fig. 6 since the LAN and WAN orange bars are the same. The total communication is around 204 KB, resulting in a slight discrepancy between the LAN and WAN end-to-end timings.

7.4 Discussion & Applications

The three TLShare protocols presented above provide different trade-offs across performance, interaction, and trust assumptions. Notably, all protocols begin by generating IT-MACs of the TLS response, taking roughly 0.46 seconds under LAN and 3.19 seconds under WAN for a payload of 256 bytes. Next, during the TLS phase, the client and the attestor in the MPC and FHE with NIZK protocols, run our conversion protocol based on Pedersen commitments with the Ed25519 and BLS12-446 curves, respectively. In contrast, the IZK protocol bypasses this conversion with an interactive ZKP performed over IT-MACs with the attestor directly.

Table 5: TLShare total end-to-end time (sec) containing both the TLS phase and the share/encryption phase under LAN and WAN for a payload of 2^8 bytes.

Network	**MPC Mode**	**FHE with NIZK**	**FHE with IZK**
LAN	0.477	1.127	0.618
WAN	3.615	4.601	3.361

An important difference between the protocols is in the bandwidth and the elliptic curve group representation. Our MPC mode uses Ed25519 curve elements at 256 bits during the conversion step in the TLS phase, while the FHE with NIZK operates with elements in the second source group $\hat{\mathbb{G}}$ of the pairing-friendly BLS12-446 of size 892 bits. This results in roughly a 4× increase in bandwidth during the conversion protocol. This cost is also reflected empirically: in the WAN setting, the conversion of a single batch under FHE with NIZK (307 ms from Table 4) takes around as much as a conversion of a payload of size 2^8 (i.e., 9 batches) under MPC (318 ms from Table 2). In contrast, the FHE with IZK avoids the TLS conversion cost altogether, yet still requires interaction during encryption proof generation between the client and the attestor.

When comparing FHE with NIZK and FHE with IZK in terms of end-to-end time, IZK outperforms NIZK as shown in Table 5. More specifically, under LAN, IZK takes 0.6 seconds compared to 1.1 seconds, while in the WAN setting, IZK takes 3.4 seconds as compared to 4.6 seconds for NIZK. This is attributed to the expensive CGGI NIZK. However, this advantage comes at the cost of a required preprocessing phase to generate approximately 131 thousand edaBits, taking around 3.3 seconds in LAN and 9.6 seconds in WAN. Furthermore, the FHE mode with IZK is limited to prime-field LWE schemes and lacks support for a ring structure, not making it compatible with TFHE-rs [96]. Conversely, the FHE mode with NIZK relies on a trusted setup, requiring stronger trust assumptions.

Finally, to present the capabilities of TLShare, we evaluated the application presented in Section 6.2 using the malicious Shamir secret sharing protocol [25,3]. We grouped records over five industries (e.g., healthcare, finance, etc.) and five job categories (e.g., executive, administrative, etc.), and computed the wage gaps between genders for different demographics. We evaluated this application for 10^3 up to 10^6 records under both LAN and WAN, with the LAN experiments taking 0.01, 0.1, 1.3, 13.1 seconds, respectively, and WAN experiments requiring 0.6, 2.07, 15.4, 146.7 seconds, respectively. The total communication scaled linearly from 0.4 MB for 1000 records, all the way up to approximately 400 MB for 1 million records.

8 Concluding Remarks

We presented TLShare, a framework that bridges the gap between authenticated web data and privacy-preserving computation. TLShare enables clients to verifiably extract TLS payloads and input them into secure downstream MPC or FHE computations, without requiring any server-side modifications. Unlike prior work restricted to Boolean predicates and single client proofs, TLShare supports arbitrary secure computation over verifiable payloads from multiple clients and servers, enabling new applications such as privacy-preserving financial risk assessment and collaborative analytics. We introduced three protocols for TLShare: one for MPC using verifiable secret sharing, and two for FHE using interactive and non-interactive zero-knowledge proofs, each ensuring input authenticity, integrity, and end-to-end privacy.

Acknowledgment

The authors would like to give special thanks to Xiao Wang for his help with QuickSilver and the EMP toolkit. We would also like to thank Sofía Celi for the fruitful discussions when we were kicking off this project.

References

1. Emmanuel A. Abbe, Amir E. Khandani, and Andrew W. Lo. Privacy-preserving methods for sharing financial risk exposures. *American Economic Review*, 102(3):65–70, May 2012.
2. Damiano Abram, Ivan Damgård, Peter Scholl, and Sven Trieflinger. Oblivious TLS via multi-party computation. In Kenneth G. Paterson, editor, *CT-RSA 2021*, volume 12704 of *LNCS*, pages 51–74, Virtual Event, May 17–20, 2021. Springer, Cham, Switzerland.
3. Toshinori Araki, Jun Furukawa, Yehuda Lindell, Ariel Nof, and Kazuma Ohara. High-throughput semi-honest secure three-party computation with an honest majority. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *ACM CCS 2016*, pages 805–817, Vienna, Austria, October 24–28, 2016. ACM Press.
4. Foteini Baldimtsi, Konstantinos Kryptos Chalkias, Yan Ji, Jonas Lindstrøm, Deepak Maram, Ben Riva, Arnab Roy, Mahdi Sedaghat, and Joy Wang. zkLogin: Privacy-preserving blockchain authentication with existing credentials. In Bo Luo, Xiaojing Liao, Jun Xu, Engin Kirda, and David Lie, editors, *ACM CCS 2024*, pages 3182–3196, Salt Lake City, UT, USA, October 14–18, 2024. ACM Press.
5. Carsten Baum, Alex J. Malozemoff, Marc B. Rosen, and Peter Scholl. Mac'n'cheese: Zero-knowledge proofs for boolean and arithmetic circuits with nested disjunctions. In Tal Malkin and Chris Peikert, editors, *CRYPTO 2021, Part IV*, volume 12828 of *LNCS*, pages 92–122, Virtual Event, August 16–20, 2021. Springer, Cham, Switzerland.
6. Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. Scalable zero knowledge with no trusted setup. In Alexandra Boldyreva and Daniele Micciancio, editors, *CRYPTO 2019, Part III*, volume 11694 of *LNCS*, pages 701–732, Santa Barbara, CA, USA, August 18–22, 2019. Springer, Cham, Switzerland.
7. Eli Ben-Sasson, Alessandro Chiesa, Daniel Genkin, Eran Tromer, and Madars Virza. SNARKs for C: Verifying program executions succinctly and in zero knowledge. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part II*, volume 8043 of *LNCS*, pages 90–108, Santa Barbara, CA, USA, August 18–22, 2013. Springer Berlin Heidelberg, Germany.

8. Rikke Bendlin, Ivan Damgård, Claudio Orlandi, and Sarah Zakarias. Semi-homomorphic encryption and multiparty computation. In Kenneth G. Paterson, editor, *EUROCRYPT 2011*, volume 6632 of *LNCS*, pages 169–188, Tallinn, Estonia, May 15–19, 2011. Springer Berlin Heidelberg, Germany.
9. Daniel J. Bernstein. Curve25519: New Diffie-Hellman speed records. In Moti Yung, Yevgeniy Dodis, Aggelos Kiayias, and Tal Malkin, editors, *PKC 2006*, volume 3958 of *LNCS*, pages 207–228, New York, NY, USA, April 24–26, 2006. Springer Berlin Heidelberg, Germany.
10. Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. Recursive composition and bootstrapping for SNARKS and proof-carrying data. In Dan Boneh, Tim Roughgarden, and Joan Feigenbaum, editors, *45th ACM STOC*, pages 111–120, Palo Alto, CA, USA, June 1–4, 2013. ACM Press.
11. Nir Bitansky, Alessandro Chiesa, Yuval Ishai, Rafail Ostrovsky, and Omer Paneth. Succinct non-interactive arguments via linear interactive proofs. *Journal of Cryptology*, 35(3):15, July 2022.
12. Dan Boneh, Elette Boyle, Henry Corrigan-Gibbs, Niv Gilboa, and Yuval Ishai. Zero-knowledge proofs on secret-shared data via fully linear PCPs. In Alexandra Boldyreva and Daniele Micciancio, editors, *CRYPTO 2019, Part III*, volume 11694 of *LNCS*, pages 67–97, Santa Barbara, CA, USA, August 18–22, 2019. Springer, Cham, Switzerland.
13. Dan Boneh, Elette Boyle, Henry Corrigan-Gibbs, Niv Gilboa, and Yuval Ishai. Lightweight techniques for private heavy hitters. In *2021 IEEE Symposium on Security and Privacy*, pages 762–776, San Francisco, CA, USA, May 24–27, 2021. IEEE Computer Society Press.
14. Carl Bootland, Kelong Cong, Daniel Demmler, Tore Kasper Frederiksen, Benoit Libert, Jean-Baptiste Orfila, Dragos Rotaru, Nigel P. Smart, Titouan Tanguy, Samuel Tap, and Michael Walter. Threshold (fully) homomorphic encryption. "Cryptology ePrint Archive, Report 2025/699", 2025.
15. Lorenz Breidenbach et al. Chainlink 2.0: Next steps in the evolution of decentralized oracle networks. *Chainlink Labs*, 1:1–136, 2021.
16. Marcus Brinkmann, Christian Dresen, Robert Merget, Damian Poddebniak, Jens Müller, Juraj Somorovsky, Jörg Schwenk, and Sebastian Schinzel. ALPACA: Application layer protocol confusion - analyzing and mitigating cracks in TLS authentication. In Michael Bailey and Rachel Greenstadt, editors, *USENIX Security 2021*, pages 4293–4310. USENIX Association, August 11–13, 2021.
17. Dario Catalano and Dario Fiore. Vector commitments and their applications. In Kaoru Kurosawa and Goichiro Hanaoka, editors, *PKC 2013*, volume 7778 of *LNCS*, pages 55–72, Nara, Japan, February 26 – March 1, 2013. Springer Berlin Heidelberg, Germany.
18. Sofía Celi, Alex Davidson, Hamed Haddadi, Gonçalo Pestana, and Joe Rowell. DiStefano: Decentralized infrastructure for sharing trusted encrypted facts and nothing more. Cryptology ePrint Archive, Report 2023/1063, 2023.
19. Sofía Celi, Shai Levin, and Joe Rowell. CDLS: Proving knowledge of committed discrete logarithms with soundness. In Serge Vaudenay and Christophe Petit, editors, *AFRICACRYPT 24*, volume 14861 of *LNCS*, pages 69–93, Douala, Cameroon, July 10–12, 2024. Springer, Cham, Switzerland.
20. Kwan Yin Chan, Handong Cui, and Tsz Hon Yuen. DIDO: Data Provenance from Restricted TLS 1.3 Websites. In *International Conference on Information Security Practice and Experience*, pages 154–169. Springer, 2023.
21. Guoxing Chen, Sanchuan Chen, Yuan Xiao, Yinqian Zhang, Zhiqiang Lin, and Ten-Hwang Lai. SgxPectre: Stealing intel secrets from SGX enclaves via speculative execution. In *2019 IEEE European Symposium on Security and Privacy*, pages 142–157, Stockholm, Sweden, June 17–19, 2019. IEEE Computer Society Press.
22. Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. TFHE: Fast fully homomorphic encryption over the torus. *Journal of Cryptology*, 33(1):34–91, January 2020.
23. Henry Corrigan-Gibbs and Dan Boneh. Prio: Private, robust, and scalable computation of aggregate statistics. In *Proceedings of the 14th USENIX Conference on Networked Systems Design and Implementation*, NSDI'17, page 259–282, USA, 2017. USENIX Association.
24. Boston Women's Workforce Council. Boston women's workforce council 2017 wage gap report. Technical report, City of Boston, 2017.
25. Ronald Cramer, Ivan Damgård, and Ueli M. Maurer. General secure multi-party computation from any linear secret-sharing scheme. In Bart Preneel, editor, *EUROCRYPT 2000*, volume 1807 of *LNCS*, pages 316–334, Bruges, Belgium, May 14–18, 2000. Springer Berlin Heidelberg, Germany.
26. Ivan Damgård, Valerio Pastro, Nigel P. Smart, and Sarah Zakarias. Multiparty computation from somewhat homomorphic encryption. In Reihaneh Safavi-Naini and Ran Canetti, editors, *CRYPTO 2012*, volume 7417 of *LNCS*, pages 643–662, Santa Barbara, CA, USA, August 19–23, 2012. Springer Berlin Heidelberg, Germany.
27. Hannah Davis, Christopher Patton, Mike Rosulek, and Phillipp Schoppmann. Verifiable distributed aggregation functions. *PoPETs*, 2023(4):578–592, October 2023.

28. T. Dierks and E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.2. RFC 5246, August 2008. Obsoleted by RFC 8446.
29. Samuel Dittmer, Yuval Ishai, Steve Lu, and Rafail Ostrovsky. Authenticated garbling from simple correlations. In Yevgeniy Dodis and Thomas Shrimpton, editors, *CRYPTO 2022, Part IV*, volume 13510 of *LNCS*, pages 57–87, Santa Barbara, CA, USA, August 15–18, 2022. Springer, Cham, Switzerland.
30. Léo Ducas and Daniele Micciancio. FHEW: Bootstrapping homomorphic encryption in less than a second. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT 2015, Part I*, volume 9056 of *LNCS*, pages 617–640, Sofia, Bulgaria, April 26–30, 2015. Springer Berlin Heidelberg, Germany.
31. Moumita Dutta, Chaya Ganesh, Sikhar Patranabis, and Nitin Singh. Compute, but verify: Efficient multiparty computation over authenticated inputs. In Kai-Min Chung and Yu Sasaki, editors, *ASIACRYPT 2024, Part VI*, volume 15489 of *LNCS*, pages 133–166, Kolkata, India, December 9–13, 2024. Springer, Singapore, Singapore.
32. Úlfar Erlingsson, Vitaly Feldman, Ilya Mironov, Ananth Raghunathan, Kunal Talwar, and Abhradeep Thakurta. Amplification by shuffling: From local to central differential privacy via anonymity. In Timothy M. Chan, editor, *30th SODA*, pages 2468–2479, San Diego, CA, USA, January 6–9, 2019. ACM-SIAM.
33. Jens Ernstberger, Jan Lauinger, Yinnan Wu, Arthur Gervais, and Sebastian Steinhorst. ORIGO: Proving Provenance of Sensitive Data with Constant Communication. *PoPETs*, 2025(2):416–433, July 2025.
34. Daniel Escudero, Satrajit Ghosh, Marcel Keller, Rahul Rachuri, and Peter Scholl. Improved primitives for MPC over mixed arithmetic-binary circuits. In Daniele Micciancio and Thomas Ristenpart, editors, *CRYPTO 2020, Part II*, volume 12171 of *LNCS*, pages 823–852, Santa Barbara, CA, USA, August 17–21, 2020. Springer, Cham, Switzerland.
35. Thibauld Feneuil, Jules Maire, Matthieu Rivain, and Damien Vergnaud. Zero-knowledge protocols for the subset sum problem from MPC-in-the-head with rejection. In Shweta Agrawal and Dongdai Lin, editors, *ASIACRYPT 2022, Part II*, volume 13792 of *LNCS*, pages 371–402, Taipei, Taiwan, December 5–9, 2022. Springer, Cham, Switzerland.
36. Hossein Fereidooni, Samuel Marchal, Markus Miettinen, Azalia Mirhoseini, Helen Möllering, Thien Duc Nguyen, Phillip Rieger, Ahmad-Reza Sadeghi, Thomas Schneider, Hossein Yalame, and Shaza Zeitouni. SAFELearn: Secure Aggregation for private FEderated Learning. In *2021 IEEE Security and Privacy Workshops (SPW)*, pages 56–62, 2021.
37. J. Galindo and P. Tamayo. Credit Risk Assessment Using Statistical and Machine Learning: Basic Methodology and Risk Modeling Applications. *Computational Economics*, 15(1):107–143, Apr 2000.
38. Till Gehlhar, Felix Marx, Thomas Schneider, Ajith Suresh, Tobias Wehrle, and Hossein Yalame. SafeFL: MPC-friendly Framework for Private and Robust Federated Learning. In *2023 IEEE Security and Privacy Workshops (SPW)*, pages 69–76, Los Alamitos, CA, USA, May 2023. IEEE Computer Society.
39. Rosario Gennaro, Craig Gentry, Bryan Parno, and Mariana Raykova. Quadratic span programs and succinct NIZKs without PCPs. In Thomas Johansson and Phong Q. Nguyen, editors, *EUROCRYPT 2013*, volume 7881 of *LNCS*, pages 626–645, Athens, Greece, May 26–30, 2013. Springer Berlin Heidelberg, Germany.
40. Craig Gentry. *A fully homomorphic encryption scheme*. PhD thesis, Stanford University, 2009.
41. Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. In Alfred Aho, editor, *19th ACM STOC*, pages 218–229, New York City, NY, USA, May 25–27, 1987. ACM Press.
42. Google/Apple. Exposure notification, cryptography specification. https://covid19-static.cdn-apple.com/applications/covid19/current/static/contact-tracing/pdf/ExposureNotification-CryptographySpecificationv1.2.pdf, 2020.
43. Charles Gouert, Dimitris Mouris, and Nektarios Georgios Tsoutsos. SoK: New insights into fully homomorphic encryption libraries via standardized benchmarks. *PoPETs*, 2023(3):154–172, July 2023.
44. Charles Gouert, Mehmet Ugurbil, Dimitris Mouris, Miguel de Vega, and Nektarios Georgios Tsoutsos. Ripple: Accelerating programmable bootstraps for FHE with wavelet approximations. In Nicky Mouha and Nick Nikiforakis, editors, *ISC 2024, Part I*, volume 15257 of *LNCS*, pages 273–293, Arlington, VA, USA, October 23–25, 2024. Springer, Cham, Switzerland.
45. Kose John, Anthony W Lynch, and Manju Puri. Credit ratings, collateral, and loan characteristics: Implications for yield. *The Journal of Business*, 76(3):371–409, 2003.
46. Marc Joye. Balanced non-adjacent forms. In Mehdi Tibouchi and Huaxiong Wang, editors, *ASIACRYPT 2021, Part III*, volume 13092 of *LNCS*, pages 553–576, Singapore, December 6–10, 2021. Springer, Cham, Switzerland.
47. Marc Joye. SoK: Fully homomorphic encryption over the [discretized] torus. *IACR TCHES*, 2022(4):661–692, 2022.
48. Marc Joye. TFHE public-key encryption revisited. In Elisabeth Oswald, editor, *CT-RSA 2024*, volume 14643 of *LNCS*, pages 277–291, San Francisco, CA, USA, May 6–9, 2024. Springer, Cham, Switzerland.

49. Marcel Keller. MP-SPDZ: A versatile framework for multi-party computation. In Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna, editors, *ACM CCS 2020*, pages 1575–1590, Virtual Event, USA, November 9–13, 2020. ACM Press.
50. Marcel Keller, Emmanuela Orsini, and Peter Scholl. MASCOT: Faster malicious arithmetic secure computation with oblivious transfer. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *ACM CCS 2016*, pages 830–842, Vienna, Austria, October 24–28, 2016. ACM Press.
51. Vladimir Kolesnikov and Thomas Schneider. Improved garbled circuit: Free XOR gates and applications. In Luca Aceto, Ivan Damgård, Leslie Ann Goldberg, Magnús M. Halldórsson, Anna Ingólfsdóttir, and Igor Walukiewicz, editors, *ICALP 2008, Part II*, volume 5126 of *LNCS*, pages 486–498, Reykjavik, Iceland, July 7–11, 2008. Springer Berlin Heidelberg, Germany.
52. Andrei Lapets, Frederick Jansen, Kinan Dak Albab, Rawane Issa, Lucy Qin, Mayank Varia, and Azer Bestavros. Accessible privacy-preserving web-based data analysis for assessing and addressing economic inequalities. In *Proceedings of the 1st ACM SIGCAS Conference on Computing and Sustainable Societies*, COMPASS '18, New York, NY, USA, 2018. Association for Computing Machinery.
53. Jan Lauinger, Jens Ernstberger, Andreas Finkenzeller, and Sebastian Steinhorst. Janus: Fast Privacy-Preserving Data Provenance For TLS 1.3. *PoPETs*, 2025(1):511–530, July 2025.
54. Benoît Libert. Vector commitments with proofs of smallness: Short range proofs and more. In Qiang Tang and Vanessa Teague, editors, *PKC 2024, Part II*, volume 14602 of *LNCS*, pages 36–67, Sydney, NSW, Australia, April 15–17, 2024. Springer, Cham, Switzerland.
55. Benoît Libert and Moti Yung. Concise mercurial vector commitments and independent zero-knowledge sets with short proofs. In Daniele Micciancio, editor, *TCC 2010*, volume 5978 of *LNCS*, pages 499–517, Zurich, Switzerland, February 9–11, 2010. Springer Berlin Heidelberg, Germany.
56. Iuon-Chang Lin and Chun-Wei Kuo. Trustworthy blockchain oracles for smart contracts. In George A. Tsihrintzis, Shiuh-Jeng Wang, and Iuon-Chang Lin, editors, *2021 International Conference on Security and Information Technologies with AI, Internet Computing and Big-data Applications*, pages 379–389, Cham, 2023. Springer International Publishing.
57. Richard Lindner and Chris Peikert. Better key sizes (and attacks) for LWE-based encryption. In Aggelos Kiayias, editor, *CT-RSA 2011*, volume 6558 of *LNCS*, pages 319–339, San Francisco, CA, USA, February 14–18, 2011. Springer Berlin Heidelberg, Germany.
58. Zeyu Liu, Daniele Micciancio, and Yuriy Polyakov. Large-precision homomorphic sign evaluation using FHEW/TFHE bootstrapping. In Shweta Agrawal and Dongdai Lin, editors, *ASIACRYPT 2022, Part II*, volume 13792 of *LNCS*, pages 130–160, Taipei, Taiwan, December 5–9, 2022. Springer, Cham, Switzerland.
59. Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On ideal lattices and learning with errors over rings. In Henri Gilbert, editor, *EUROCRYPT 2010*, volume 6110 of *LNCS*, pages 1–23, French Riviera, May 30 – June 3, 2010. Springer Berlin Heidelberg, Germany.
60. Christian Mainka, Vladislav Mladenov, and Jörg Schwenk. Do not trust me: Using malicious IdPs for analyzing and attacking single sign-on. In *2016 IEEE European Symposium on Security and Privacy*, pages 321–336, Saarbrücken, Germany, March 21–24, 2016. IEEE Computer Society Press.
61. Nikolaos Makriyannis and Udi Peled. A note on the security of GG18. https://info.fireblocks.com/hubfs/A_Note_on_the_Security_of_GG.pdf, 2024.
62. Gina Mantica. Using Data Science to Address the Gender and Racial Wage Gap. https://www.bu.edu/articles/2021/using-data-science-to-address-the-gender-and-racial-wage-gap, 2021.
63. Silvio Micali. Computationally sound proofs. *SIAM Journal on Computing*, 30(4):1253–1298, 2000.
64. Dimitris Mouris, Daniel Masny, Ni Trieu, Shubho Sengupta, Prasad Buddhavarapu, and Benjamin M. Case. Delegated private matching for compute. *PoPETs*, 2024(2):49–72, April 2024.
65. Dimitris Mouris, Christopher Patton, Hannah Davis, Pratik Sarkar, and Nektarios Georgios Tsoutsos. Mastic: Private Weighted Heavy-Hitters and Attribute-Based Metrics. *PoPETs*, 2025(1):290–319, July 2025.
66. Dimitris Mouris, Pratik Sarkar, and Nektarios Georgios Tsoutsos. PLASMA: Private, lightweight aggregated statistics against malicious adversaries. *PoPETs*, 2024(3):4–24, July 2024.
67. Dimitris Mouris and Nektarios Georgios Tsoutsos. Zilch: A Framework for Deploying Transparent Zero-Knowledge Proofs. *IEEE Transactions on Information Forensics and Security*, 16:3269–3284, 2021.
68. Jesper Buus Nielsen, Peter Sebastian Nordholt, Claudio Orlandi, and Sai Sheshank Burra. A new approach to practical active-secure two-party computation. In Reihaneh Safavi-Naini and Ran Canetti, editors, *CRYPTO 2012*, volume 7417 of *LNCS*, pages 681–700, Santa Barbara, CA, USA, August 19–23, 2012. Springer Berlin Heidelberg, Germany.
69. Nicola Paltrinieri, Louise Comfort, and Genserik Reniers. Learning about risk: Machine learning for risk assessment. *Safety Science*, 118:475–486, October 2019.

70. Torben P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In Joan Feigenbaum, editor, *CRYPTO'91*, volume 576 of *LNCS*, pages 129–140, Santa Barbara, CA, USA, August 11–15, 1992. Springer Berlin Heidelberg, Germany.
71. Primus Labs. Primus FHE. https://github.com/primus-labs/primus-fhe, 2025.
72. Privacy and Ethereum Foundation Scaling Explorations. TLSNotary, 2022. Accessed: 2014.
73. Lucy Qin, Peter Flockhart, Andrei Lapets, Kinan Dak Albab, Mayank Varia, Shannon Roberts, and Ira Globus-Harris. From usability to secure computing and back again. In *Proceedings of the Fifteenth USENIX Conference on Usable Privacy and Security*, SOUPS '19, pages 191–210, USA, August 2019. USENIX Association.
74. Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. In Harold N. Gabow and Ronald Fagin, editors, *37th ACM STOC*, pages 84–93, Baltimore, MA, USA, May 22–24, 2005. ACM Press.
75. Relic-Toolkit. RELIC. https://github.com/relic-toolkit/relic, 2025.
76. E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.3. RFC 8446, August 2018.
77. Hubert Ritzdorf, Karl Wüst, Arthur Gervais, Guillaume Felley, and Srdjan Capkun. TLS-N: Non-repudiation over TLS enablign ubiquitous content signing. In *NDSS 2018*, San Diego, CA, USA, February 18–21, 2018. The Internet Society.
78. Michael Rosenberg, Jacob D. White, Christina Garman, and Ian Miers. zk-creds: Flexible anonymous credentials from zkSNARKs and existing identity infrastructure. In *2023 IEEE Symposium on Security and Privacy*, pages 790–808, San Francisco, CA, USA, May 21–25, 2023. IEEE Computer Society Press.
79. Ron Rothblum. Homomorphic encryption: From private-key to public-key. In Yuval Ishai, editor, *TCC 2011*, volume 6597 of *LNCS*, pages 219–234, Providence, RI, USA, March 28–30, 2011. Springer Berlin Heidelberg, Germany.
80. Manuel B. Santos. PECO: methods to enhance the privacy of DECO protocol. Cryptology ePrint Archive, Report 2022/1774, 2022.
81. Adi Shamir. How to share a secret. *Communications of the Association for Computing Machinery*, 22(11):612–613, November 1979.
82. Sijun Tan, Weikeng Chen, Ryan Deng, and Raluca Ada Popa. MPCAuth: Multi-factor authentication for distributed-trust systems. In *2023 IEEE Symposium on Security and Privacy*, pages 829–847, San Francisco, CA, USA, May 21–25, 2023. IEEE Computer Society Press.
83. Dmytro Tymokhanov and Omer Shlomovits. Alpha-rays: Key extraction attacks on threshold ECDSA implementations. Cryptology ePrint Archive, Report 2021/1621, 2021.
84. Jo Van Bulck, Marina Minkin, Ofir Weisse, Daniel Genkin, Baris Kasikci, Frank Piessens, Mark Silberstein, Thomas F. Wenisch, Yuval Yarom, and Raoul Strackx. Foreshadow: Extracting the keys to the intel SGX kingdom with transient out-of-order execution. In William Enck and Adrienne Porter Felt, editors, *USENIX Security 2018*, pages 991–1008, Baltimore, MD, USA, August 15–17, 2018. USENIX Association.
85. Alexander Viand, Patrick Jattke, and Anwar Hithnawi. SoK: Fully homomorphic encryption compilers. In *2021 IEEE Symposium on Security and Privacy*, pages 1092–1108, San Francisco, CA, USA, May 24–27, 2021. IEEE Computer Society Press.
86. Xiao Wang, Alex J. Malozemoff, and Jonathan Katz. EMP-toolkit: Efficient MultiParty computation toolkit. https://github.com/emp-toolkit, 2016.
87. Xiao Wang, Samuel Ranellucci, and Jonathan Katz. Authenticated garbling and efficient maliciously secure two-party computation. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 2017*, pages 21–37, Dallas, TX, USA, October 31 – November 2, 2017. ACM Press.
88. Xiao Wang, Samuel Ranellucci, and Jonathan Katz. Global-scale secure multiparty computation. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 2017*, pages 39–56, Dallas, TX, USA, October 31 – November 2, 2017. ACM Press.
89. Chenkai Weng, Kang Yang, Jonathan Katz, and Xiao Wang. Wolverine: Fast, scalable, and communication-efficient zero-knowledge proofs for boolean and arithmetic circuits. In *2021 IEEE Symposium on Security and Privacy*, pages 1074–1091, San Francisco, CA, USA, May 24–27, 2021. IEEE Computer Society Press.
90. Xiang Xie, Kang Yang, Xiao Wang, and Yu Yu. Lightweight authentication of web data via garble-then-prove. In Davide Balzarotti and Wenyuan Xu, editors, *USENIX Security 2024*, Philadelphia, PA, USA, August 14–16, 2024. USENIX Association.
91. Kang Yang, Pratik Sarkar, Chenkai Weng, and Xiao Wang. QuickSilver: Efficient and affordable zero-knowledge proofs for circuits and polynomials over any field. In Giovanni Vigna and Elaine Shi, editors, *ACM CCS 2021*, pages 2986–3001, Virtual Event, Republic of Korea, November 15–19, 2021. ACM Press.
92. Kang Yang, Xiao Wang, and Jiang Zhang. More efficient MPC from improved triple generation and authenticated garbling. In Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna, editors, *ACM CCS 2020*, pages 1627–1646, Virtual Event, USA, November 9–13, 2020. ACM Press.

93. Andrew Chi-Chih Yao. Protocols for secure computations (extended abstract). In *23rd FOCS*, pages 160–164, Chicago, Illinois, November 3–5, 1982. IEEE Computer Society Press.
94. Andrew Chi-Chih Yao. How to generate and exchange secrets (extended abstract). In *27th FOCS*, pages 162–167, Toronto, Ontario, Canada, October 27–29, 1986. IEEE Computer Society Press.
95. Yuval Yarom and Katrina Falkner. FLUSH+RELOAD: A high resolution, low noise, L3 cache side-channel attack. In Kevin Fu and Jaeyeon Jung, editors, *USENIX Security 2014*, pages 719–732, San Diego, CA, USA, August 20–22, 2014. USENIX Association.
96. Zama. TFHE-rs. https://github.com/zama-ai/tfhe-rs, 2025.
97. Fan Zhang, Ethan Cecchetti, Kyle Croman, Ari Juels, and Elaine Shi. Town crier: An authenticated data feed for smart contracts. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *ACM CCS 2016*, pages 270–282, Vienna, Austria, October 24–28, 2016. ACM Press.
98. Fan Zhang, Deepak Maram, Harjasleen Malvai, Steven Goldfeder, and Ari Juels. DECO: Liberating web data using decentralized oracles for TLS. In Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna, editors, *ACM CCS 2020*, pages 1919–1938, Virtual Event, USA, November 9–13, 2020. ACM Press.

A Security of MPC mode protocol

Recall Theorem 1. We now prove that the protocol in Fig. 3 securely realizes $\mathcal{F}_{\mathrm{TLShare}}$.

Proof. Assuming up to $\mathsf{T}-1$ MPC nodes are corrupted by the adversary, PVSS [70] ensures perfect secrecy of the shared secret (i.e., response R). Therefore, the adversary's view is independent of the secret, so a simulator can draw a random "dummy" secret.

When the client is corrupted, it may attempt to convince nodes that the submitted shares correspond to the value committed in the signed base commitment, when a different is provided. However, because MPC nodes verify the consistency of the polynomial coefficient commitments, PVSS scheme [70] guarantees that any subset of T MPC nodes finds a pair (R_j, r_j) such that $C_j = g^{\mathsf{R}_j} \cdot h^{r_j}$. This property enforces binding: the client cannot distribute shares $([\![x]\!], [\![r_j]\!])$ that do not correspond to the signed base commitment C_j. Since no interaction is needed from nodes back to the client, the simulator simply runs the protocol to produce an indistinguishable view.

B Correctness of Proof of Membership

Recall from [54] that the commitments $\hat{C}$ and $\hat{V}$ satisfy the following:

$$\begin{aligned} e(g_{n+1-i}, \hat{C}) &= e(g^{\gamma}_{n+1-i} \cdot \textstyle\prod_{j=1}^{n} g^{w_j}_{n+1+j-i}, \hat{g}) \cdot e(g_1, \hat{g}_n)^{w_i} \\ e(g_n, \hat{V}) &= e(g^{r}_{n}, \hat{g}) \cdot e(g_1, \hat{g}_n)^{w} \end{aligned} \tag{5}$$

Proof. To check the correctness of the proof, we need to guarantee that Eq. (3) holds. Observe that:

$$\begin{aligned} & e(\textstyle\prod_{i\in L} g^{2^{i-1-k}}_{n+1-i}, \hat{C}) \\ &= \textstyle\prod_{i\in L} e(g_{n+1-i}, \hat{g}^{\gamma} \cdot \prod_{j=1}^{n} \hat{g}_j^{w_j})^{2^{i-1-k}} \\ &= \textstyle\prod_{i\in L} e(g^{\gamma} \cdot \prod_{j=1}^{n} g_j^{w_j}, \hat{g}_{n+1-i})^{2^{i-1-k}} \\ &= \textstyle\prod_{i\in L} e(g^{\gamma}_{n+1-i} \cdot \prod_{j=1}^{n} g^{w_j}_{n+1+j-i}, \hat{g})^{2^{i-1-k}} \\ &= \textstyle\prod_{i\in L} e(g^{\gamma}_{n+1-i} \cdot g^{w_i}_{n+1} \cdot \prod_{j\in[1,n]\setminus\{i\}}^{n} g^{w_j}_{n+1+j-i}, \hat{g})^{2^{i-1-k}} \\ &= \textstyle\prod_{i\in L} e(g_{n+1}, \hat{g})^{2^{i-1-k}\cdot w_i} \\ &\quad \cdot \textstyle\prod_{i\in L} e(g^{\gamma}_{n+1-i} \cdot \prod_{j\in[1,n]\setminus\{i\}}^{n} g^{w_j}_{n+1+j-i}, \hat{g})^{2^{i-1-k}}, \end{aligned}$$

where $\prod_{i\in L} e\left(g_{n+1}, \hat{g}\right)^{2^{i-1-k}\cdot w_i} = e\left(g_1, \hat{g}_n\right)^{w}$. In addition, from Eq. (5) it is known that $e(g_n, \hat{V}) = e\left(g_1, \hat{g}_n\right)^{w} \cdot e(g_n^r, \hat{g})$. So, dividing out the two equations $e(\prod_{i\in L} g_{n+1-i}^{2^{i-1-k}}, \hat{C})/e(g_n, \hat{V})$ yields the following,

$$
\begin{aligned}
&\frac{e(g_1,\hat{g}_n)^w \cdot \prod_{i\in L} e(g_{n+1-i}^{\gamma} \cdot \prod_{j\in[1,n]\setminus\{i\}}^{n} g_{n+1+j-i}^{w_j}, \hat{g})^{2^{i-1-k}}}{e(g_1,\hat{g}_n)^w \cdot e(g_n^r, \hat{g})} \\
&= e(\textstyle\prod_{i\in L}[g_{n+1-i}^{\gamma} \cdot \prod_{j\in[1,n]\setminus\{i\}}^{n} g_{n+1+j-i}^{w_j}]^{2^{i-1-k}}, \hat{g}) \cdot e(g_n^{-r}, \hat{g}) \\
&= e(g_n^{-r} \cdot \textstyle\prod_{i\in L}[g_{n+1-i}^{\gamma} \cdot \prod_{j\in[1,n]\setminus\{i\}}^{n} g_{n+1+j-i}^{w_j}]^{2^{i-1-k}}, \hat{g}) \\
&= e(\pi, \hat{g}).
\end{aligned}
$$

C Security of NIZK of Vector Membership

Recall that, as described in Section 5.1, the verifier only receives $\hat{V}$, $\hat{C}$ and π which do not reveal any information about the witness $\boldsymbol{w}$ as its value is hidden in the exponent and masked by randomness r and γ. This guarantees the zero-knowledge property.

Let us explore why π guarantees the soundness property of the proof. Assume that $\hat{C}$ is a commitment $\hat{C} = \hat{g}^r \cdot \hat{g}_j^z$ to some other element $\boldsymbol{z}$ different from $\boldsymbol{w}$. Then, the relation $e(g_n, \hat{V}) = e(g_n^r, \hat{g}) \cdot e(g_1, \hat{g}_n)^{\boldsymbol{w}}$ becomes $e(g_n, \hat{C}) = e(g_n^r, \hat{g}) \cdot e(g_1, \hat{g}_n)^{\boldsymbol{z}}$. Now, the verification step of the verifier yields,

$$
e(\textstyle\prod_{i\in L} g_{n+1-i}^{2^{i-1-k}}, \hat{V})/e(g_n, \hat{C}) = e(\pi, \hat{g}) \cdot e(g_1, \hat{g}_n)^{w-z}.
$$

Therefore, the membership verification Eq. (3) is only valid in the case $w = z$.

D Security of TLShare FHE Mode with NIZK Proofs

We begin by observing that the vector-membership proof $\mathsf{Prove}_{\mathsf{pp}}$ presupposes that the vector commitment $\hat{V}$ indeed binds to a *binary* vector $\boldsymbol{w}$. This is ensured by the $\mathsf{CGGI.Prove}_{\mathsf{pp}}$ algorithm, which outputs both $\hat{V}$ and a proof component within π_{CGGI} that certifies each coordinate of $\boldsymbol{w}$ is in $\{0, 1\}$.

The security of our TLShare protocol in $\mathsf{mode} = \mathsf{fhe}$, follows from the composable security of its individual components: Π_{AuthData} (GtP) is UC-secure [90]; vector membership NIZK ($\mathsf{VM.Prove}_{\mathsf{pp}}, \mathsf{VM.Verify}_{\mathsf{pp}}$) ensures correctness and consistency between the committed $\boldsymbol{w}$ and the committed value μ; and the public-key CGGI encryption NIZK ($\mathsf{CGGI.Prove}_{\mathsf{pp}}, \mathsf{CGGI.Verify}_{\mathsf{pp}}$) guarantees the validity of the ciphertext of the same $\boldsymbol{w}$ used in the membership proof [14].

E Security of TLShare FHE Mode with IZK Proofs

As outlined in the protocol, after the GtP phase, data is secretly shared under the QuickSilver (QS) protocol. Specifically, the client and the attestor each hold the IT-MAC. The subsequent proof of a valid FHE ciphertext statement is a natural extension of QuickSilver. The security of this phase directly follows from the security guarantees of both the GtP and QS protocols. In other words, TLShare securely realizes the intended ideal functionality if the relation

$$
\begin{aligned}
\mathcal{R} = \Big\{ \big((\boldsymbol{a}, \boldsymbol{b}), (\boldsymbol{r}, \mu, \boldsymbol{e}, e_0)\big) \mid\; & \mu \in \mathbb{Z}_p \quad \wedge \quad \boldsymbol{r} \in \{0,1\}^n \quad \wedge \\
& \forall i \in [0, n],\ |e_i| \le 6 \cdot \mathsf{stdev} \quad \wedge \quad \boldsymbol{a} = \boldsymbol{A}\boldsymbol{r} + \boldsymbol{e} \bmod q \\
& \wedge \quad b = \boldsymbol{y}^{\top}\boldsymbol{r} + e_0 + \lfloor \mu \cdot \Delta \rceil \bmod q \Big\}.
\end{aligned}
$$

defines a valid ciphertext.

Indeed, the relation characterizes a valid LPR-style variant of Regev's LWE encryption of plaintext μ encrypted under randomness r and noise elements $(\boldsymbol{e}, e_0)$, as it enforces the standard encryption equations ($\boldsymbol{a} = \boldsymbol{A}\boldsymbol{r} + \boldsymbol{e} \bmod q$ and $b = \boldsymbol{y}^\top \boldsymbol{r} + e_0 + \lfloor \mu \cdot \Delta \rceil \bmod q$) and bounds on errors ($\forall i \in [0, n],\ |e_i| \leq 6 \cdot \mathsf{stdev}$), thus proving encryption validity. This relation is enforced in zero-knowledge by QS through its corresponding circuit. Moreover, GtP uses information-theoretic MACs on the TLS response bits, which are carried forward into the QS protocol. By directly using these MACs inside QS, one guarantees that the plaintext μ encrypted under randomness r and noise elements $(\boldsymbol{e}, e_0)$ is exactly the same message present in the attested TLS connection.