Bootstrapping Trust across Web2 and Web3 Domains Using Publicly Verifiable Web Data

Yuan Lu[1], Qiang Tang[2]

[1] Institute of Software, Chinese Academy of Sciences, Beijing, China.
[2] School of Computer Science, University of Sydney, Australia.

Corresponding authors. E-mails: luyuan@iscas.ac.cn; qiang.tang@sydney.edu.au

Abstract

Through public key infrastructure (PKI) and the TLS protocol, users can securely connect to trusted web services such as banks, e-commerce platforms, and government portals. If such data could be made publicly verifiable, it could serve as a natural trust anchor for decentralized Web3 systems. Prior efforts, such as DECO (Zhang et al., CCS'20), introduced TLS web oracles that allow a prover to convince a designated verifier that specific data was obtained from a TLS server. However, these designs rely on the verifier's cooperation and allow strategic aborts that prevent the prover from demonstrating the authenticity of already-transmitted TLS data.

We present a new TLS oracle protocol that achieves public verifiability and a novel form of fairness using an "encrypt-TLS-keys-to-future" paradigm. The prover and verifier jointly emulate a TLS client via secure two-party computation (2PC), obtaining additive shares of the session key. They also set up a trapdoor verifiable delay function (tVDF), allowing them to encrypt the full TLS session key under a future-unlockable value, while the TLS ciphertext is recorded on-chain. After a sufficiently long delay, the prover can reconstruct the session key and use zero-knowledge proofs to selectively disclose committed TLS data.

As an application, we present the first decentralized protocol that enables fair exchange between blockchain-backed Web3 assets and TLS-protected Web2 assets, bridging the gap between the two domains by supporting asset swaps without relying on any trusted third-party mediators.

Keywords

TLS Oracle, Fair Exchange, P2P Swap, Web2-Web3 Interoperability.

DOI: 10.65701/t0z4p7m2j8

1 Introduction

Decentralized systems promise a "permissionless" world where individuals interact, transact, and collaborate without centralized supernodes. This vision fosters the Web3 movement, where centralized applications are built atop blockchain and operate under rules enforced by distributed consensus (Labs, 2017; Nakamoto, 2008; Wood et al., 2014). However, such a trend of decentralization introduces a new problem: the absence of native trust anchors. In particular, Web3 participants typically interact using cryptographic pseudonyms, which are long-lived public keys derived from wallet addresses or smart contract identities. While this design choice enhances privacy and censorship resistance, it also presents significant challenges. Many traditional forms of trade and transactions that are straightforward in Web2 (due to real-world identities, reputation systems, and legal frameworks) become much more challenging to realize in the permissionless environment of Web3, due to the lack of mechanisms linking malicious behaviors to real-world identities.

Nevertheless, the current Internet (commonly referred to as Web2) has established trust through its well-established infrastructure. For example, when browsing HTTPS-enabled web pages (Rescorla, 2000), the browser displays a lock icon to indicate that the user is securely connected to a verified service. This is enabled by the TLS protocol (Dierks and Rescorla, 2008; Rescorla, 2018), which is backed by digital certificates issued by certificate authorities (CAs) as part of the broader Public Key Infrastructure (PKI) (Cooper, Santesson, Farrell, Boeyen, Housley, and Polk, 2008; ISO, 2022). For example, when users log into digital banking platforms, government portals, or shopping websites, they rely on HTTPS running over TLS to authenticate the server and ensure data integrity and confidentiality; Moreover, many of these web services will authenticate users in return via passwords and other second-factor authenticators, after which they can display pages containing versatile identity-linked data that serve as reputation profile information, such as names, dates of birthdays, social security numbers, account balances, transaction histories, credit scores, and more.

The above discrepancy between Web2 and Web3 raises a natural question:

Can we leverage the existing trust infrastructure of the traditional Internet to bootstrap cross-domain trust in the decentralized systems? Specifically, can we ensure that web users extract verifiable claims from trusted Internet services and anchor them on blockchain platforms in a way that is privacy-preserving and publicly verifiable?

Prior Studies and Their Limitations. At first glance, it may seem straightforward to modify the server-side TLS protocol to support publicly verifiable extraction of TLS data. For example, a website could directly sign user-specific data, enabling smart contracts to verify authenticity without intermediaries (Backman, Richer, and Sporny, 2024). However, this approach is largely impractical, since most Web2 services, especially large organizations such as governments, banks, and major e-commerce platforms, have little incentive to alter their infrastructure to accommodate the demand of Web3 world. Such modifications could introduce compatibility challenges, operational costs, and regulatory risks, while offering no direct benefit or revenue to their existing business models.

Several standard-compatible efforts are also made to bridge Web2's trust infrastructure and Web3, proposing TLS web oracle systems, which extract authenticated web data for verification by a designated verifier (typically referred to as an oracle service provider). A pioneering example is Town Crier (Zhang, Cecchetti, Croman, Juels, and Shi, 2016), which leverages Intel SGX to run a trusted enclave that fetches HTTPS-protected data and relays the result accompanied by SGX remote attestation to smart contracts. The security of Town Crier critically depends on the non-breakability of SGX, assuming that the enclave operates correctly and remains secure against hardware-level attacks. However, this trust model could be elusive in practice: First, SGX and similar trusted execution environments (TEEs) have a growing list of known vulnerabilities, which undermines their long-term reliability; Second, the model implicitly requires trust in Intel, the centralized hardware vendor, for correct remote attestation. These concerns highlight the fragility of hardware-rooted trust in decentralized settings.

To eliminate the reliance on trusted hardware, DECO (Zhang, Maram, Malvai, Goldfeder, and Juels, 2020) introduced a cryptographic approach that leverages zero-knowledge proofs over standard TLS sessions. In DECO, a prover engages in a TLS connection with a web server (and a designated verifier), thus finally producing a zero-knowledge proof attesting that specific data was received from the authenticated server. While DECO eliminates the need for trusted hardware such as SGX, it still depends on a

designated verifier who has the exclusive capacity to validate the proof. If the goal is to interface with a public blockchain, the designated verifier finally needs to publish the authenticated data to the public. This publishing step is not publicly verifiable and reintroduces a centralized point of failure. Further improvements have emerged. For example, Janus (Lauinger, Ernstberger, Finkenzeller, and Steinhorst, 2025) offers privacy-preserving provenance for the latest version of TLS 1.3 with special optimizations to the cipher suites of TLS 1.3. Xie et al. propose the garble-then-prove framework (Xie, Yang, Wang, and Yu, 2024) to avoid malicious-secure garbling circuits. However, these approaches mostly follow the design framework from DECO, and therefore, they are still designed for a designated verifier and do not support public verifiability in the presence of a dishonest verifier.

While in principle, the aforementioned designs (Lauinger et al., 2025; Xie et al., 2024; Zhang et al., 2020) can be extended to support on-chain verifiability, for example, by having the prover commit ciphertexts to the blockchain and later reconstruct the TLS session key with the verifier's help. However, such adaptations remain fundamentally limited. In particular, they cannot guarantee that the prover can generate a valid proof for already-transmitted data if the verifier chooses to abort or becomes unresponsive after the transmission. This shortcoming is especially problematic in applications such as cross-domain exchange, where one party (e.g., the verifier) may have already received off-chain value (e.g., fiat payment) but can strategically withhold their key share to prevent the other party from completing a proof for the successful completion of off-chain transactions. In such cases, the lack of fairness in enabling verifiability opens the door to asymmetric outcomes and undermines mutual trust in such decentralized systems.

In summary, despite meaningful progress, existing designs (Backman et al., 2024; Lauinger et al., 2025; Xie et al., 2024; Zhang et al., 2016,2) fall short of meeting the core requirements of Web3: (1) compatibility with standard TLS servers, (2) avoidance of strong or impractical trust assumptions, and (3) support for proof and public verifiability of any already-transmitted TLS data. Our work aims at addressing these limitations by introducing a novel "encrypt-TLS-key-to-future" paradigm, in which jointly emulated TLS clients defer TLS key recovery at the prover side through a trapdoor verifiable delay function (tVDF), and later produce publicly verifiable zk-SNARKs attesting to the authenticity of the extracted web data.

2 Results

This paper addresses these limitations of previous Web oracle protocols (Backman et al., 2024; Lauinger et al., 2025; Xie et al., 2024; Zhang et al., 2016,2), by proposing a new approach that avoids reliance on trusted third parties for public access to the extracted authenticated web data. In particular, our contribution is three-fold:

- At the core of our design, we introduce a new technique paradigm called "encrypt-TLS-keys-to-future" for realizing a TLS web oracle. In this approach, a prover and a verifier collaboratively emulate a TLS client using secure two-party computation (2PC), then establish a connection with a TLS server to obtain additive shares of the resulting session keys, which is so far similar to prior designs such as DECO. The key innovation of the "encrypt-TLS-keys-to-future" paradigm lies in the following: the verifier and the prover distributedly set up a trapdoor verifiable delay function (tVDF) (Wesolowski, 2020) and distributedly encrypt the TLS session keys using the tVDF output obtained from the trapdoor evaluation of tVDF. After a predetermined and sufficiently large time delay, the prover can eventually unlock the tVDF to decrypt the TLS session keys. This enables the prover to be able to *solely* decrypt the ciphertext of TLS transcripts that are already committed to the blockchain.
- Another contribution is a new oracle protocol design, which is publicly verifiable and supports selective disclosure, based on the "encrypt-TLS-keys-to-future" paradigm.[91] An observation from our "encrypt-TLS-keys-to-future" paradigm is: the prover eventually can decrypt the TLS ciphertext committed to the blockchain, which means the prover can leverage generic zero-knowledge proof scheme, using the TLS session key as witness, to generate a zero-knowledge proof attesting selective disclosure of specific data fields within the encrypted transcript (in lieu of decrypting the whole

[91] Note that there may be alternative ways to modify existing TLS oracle protocols to support public on-chain verification. However, our design from the "encrypt-TLS-keys-to-future" paradigm offers a unique feature: if certain events such as fiat money transfers occur during the TLS session, the prover alone can later prove that these transmissions took place, even if the verifier takes no further action. This property is particularly important in fair exchange settings, where relying on the verifier's continued participation may lead to unfairness (since the proof for TLS data extraction needs the verifier's local secret to compute). We will present a concrete example demonstrating the usefulness of this property in Section 5.

ciphertext), thereby publicly demonstrating the authenticity of certain data retrieved during the TLS session without revealing anything else.

A unique feature of the resulting TLS oracle protocol is a form of "fairness": if some event indeed occurs within the TLS session (e.g., a certain web data was transmitted), the prover alone can generate a proof to enable the public blockchain to verify the authenticated data from the TLS session, even if the other verifier maliciously aborts or becomes unresponsive.

- Finally, to show the power of our TLS oracle design from the "encrypt-TLS-keys-to-future" paradigm, we demonstrate how this publicly verifiable oracle can be applied to enable decentralized fair exchange between blockchain-backed cryptocurrency (e.g., Ethereum tokens (Wood et al., 2014)) and TLS-backed digital assets (e.g., bank balances) on the Internet. To our knowledge, this is the first decentralized protocol enabling the fair exchange between off-chain and on-chain assets in the absence of a trusted third-party mediator (Chen, Liu, Lu, and Tang, 2025).

  This instantiation for on-chain and off-chain exchange showcases the viability of our publicly verifiable web oracle design in bridging the existing Web2 infrastructure to the decentralized Web3 world, in a secure and decentralized manner.

3 Cryptographic Primitives

In this section, we will briefly recall the key cryptographic components that are relevant to our public-verifiable Web oracle design.

3.1 Digital Certificate and Public Key Infrastructure

Public Key Infrastructure (PKI) (Cooper et al., 2008; ISO, 2022) is the foundational trust system that underpins secure communication across the Internet, particularly used throughout the Transport Layer Security (TLS) protocol to verify the authenticity of Web servers. Specifically, PKI enables the user clients to authenticate web servers using digital certificates that bind a domain name to a public key. This mechanism is essential for ensuring that users are communicating with the intended server rather than a phishing website. A digital certificate (Cooper et al., 2008) of some domain name (e.g., www.bankofamerica.com) is issued by a Certificate Authority (CA), which is a trusted entity verifying the domain owner's identity and subsequently issues the certificate containing, but not limited to, the following information:

- The server's public key;
- The server's identity (e.g., domain name);
- The expiration time of the certificate;
- The CA's identity and its digital signature over the certificate's contents.

When a client connects to a server via TLS, the server presents its certificate during the handshake phase. Therefore, the client can authenticate the server by verifying that the server's certificate was issued by a CA it trusts (e.g., signed by a root CA hardcoded in the client's operating system or browser, or by an intermediate CA that links back to a root CA through a valid certificate chain).

3.2 Transport Layer Security (TLS)

Given trust over the already established PKI system, clients can further leverage the TLS protocol to connect to their intended website (though only the clients themselves can verify the website's authenticity, and our goal is to extend such designated clients' verifiability to public verifiability). For a better understanding of the context, we briefly recall the workflow and components of TLS.

From a high level, TLS operates in two phases (Dierks and Rescorla, 2008; Rescorla, 2018): handshake and data transmission. In the handshake, the client and server perform an authenticated key exchange to establish a shared pre-master secret, which is used to derive the (symmetric) session keys for the subsequent data transfer. Then, application data is transmitted securely with confidentiality and integrity, using the session keys. Notably, the server-to-client key (i.e., server's key) and the client-to-server key (i.e., client's key) can be thought of being independently derived from the pre-master secret.

Note that TLS has various versions and offers multiple choices of cipher suits. Here in this work, we will instantiate our design for TLS 1.2 (Dierks and Rescorla, 2008; Nir, Josefsson, and Pégourié-Gonnard, 2018; Salowey, Choudhury, and McGrew, 2008), using (i) elliptic curve DH key exchange (ECDHE) for

establishing the pre-master secret (Nir et al., 2018), (ii) SHA256-based message authentication code (HMAC) for deriving session keys from the pre-master secret (Krawczyk, Bellare, and Canetti, 1997), and (iii) AES-GCM or other authenticated encryption schemes for secure transfer of payload data (Dworkin, 2007; Dworkin, Barker, Nechvatal, Foti, Bassham, Roback, Dray Jr et al., 2001; McGrew, 2008; Rogaway, 2002). At the time of writing, TLS 1.2 with ECDHE, AES128-GCM, and SHA256 remains one of the most widely supported TLS cipher suites in real-world deployments (Qualys, ????). Moreover, our implementation can be seamlessly extended to support TLS 1.3 (Rescorla, 2018) with AES256-GCM and SHA384 without fundamental barriers.

Below, we briefly review these TLS components.

Negotiating shared secret from ECDHE. In ECDHE (Nir et al., 2018), the client chooses a random $c \leftarrow_{\$} \mathbb{F}_q$ and sends $g^c \in \mathbb{G}_q$ to the server, and the server chooses a random $s \leftarrow_{\$} \mathbb{F}_q$ and sends $g^s \in \mathbb{G}_q$ to the client, such that both of them can compute the common group element $Y = g^{cs} \in \mathbb{G}_q$. Here $\mathbb{F}_q$ is a prime field and $\mathbb{G}_q$ is a cyclic group with order q. The first coordinate of Y, $Y[0]$, is then used as the pre-master secret. In TLS, ECDHE is also enhanced to be authenticated. In our setting, the server authenticates itself by showing its digital certificate and a digital signature.

Key derivation from HMAC. Both TLS 1.2 and 1.3 implement their key derivation processes based on HMAC (Krawczyk et al., 1997), with differences in concrete instantiations. In particular, to derive symmetric session keys, TLS 1.2 realizes a hash-based pseudo-random function (HPRF) from HMAC; it then uses the pre-master secret $Y[0]$ as the HPRF key, on input the sum of client's nonce and server's nonce with a label "master secret", to compute the so-called master secret ms; after that, it uses HPRF again, keyed by ms with a label "key expansion", to derive the symmetric session keys.

Authenticated encryption from AES-GCM. AES is a standardized block cipher scheme (Dworkin et al., 2001), and its Galois/Counter mode (GCM) (Dworkin, 2007) extends it to provide authenticated encryption with associated data (AEAD) (McGrew, 2008; Rogaway, 2002). Informally speaking, AEAD not only encrypts messages to ensure their confidentiality against an adversary not knowing the encrypting key, but also generates the tags of message authentication code (MAC) to ensure the integrity of messages, thus preventing an adversary from tampering messages without access to the AES-GCM key.

3.3 Trapdoor Verifiable Delay Function

Our protocol employs an advanced cryptographic primitive known as a verifiable delay function (VDF) (Boneh, Bonneau, Bünz, and Fisch, 2018; Pietrzak, 2019; Wesolowski, 2020), equipped with trapdoor evaluation (Wesolowski, 2020) (tVDF), to realize the "encrypt-TLS-keys-to-future" paradigm. At a high level, a VDF is a function whose output must take a minimum amount of sequential time to compute, no matter how much parallel computing power is applied and what arbitrary acceleration algorithms are adopted. However, once the output is computed, it is accompanied by a proof for correctness, such that anyone (including smart contracts) can verify its correctness efficiently. A trapdoor VDF (Wesolowski, 2020) further includes a trapdoor that allows its holder to bypass the sequential delay and compute the output almost instantly. Later in our protocol, we leverage this trapdoor to encrypt the TLS session keys in advance, before anyone (without the trapdoor) can feasibly compute the VDF output through the standard slow path. This ensures that the decryption of TLS traffic can happen and only happen after a designated delay, enforcing a form of time-lock and enabling delayed verifiability.

3.4 Zero-Knowledge Succinct Non-interactive ARgument of Knowledge (zk-SNARK)

To ensure that the data extracted from the TLS session can be validated publicly without revealing sensitive information, our approach will employ zero-knowledge Succinct Non-interactive ARguments of Knowledge (zk-SNARK) (Ben-Sasson, Chiesa, Tromer, and Virza, 2014; Gennaro, Gentry, Parno, and Raykova, 2013; Groth, 2016). For completeness, we will provide an informal definition of zk-SNARK below.

Informally, a zk-SNARK scheme is a cryptographic proof system (Goldwasser, Micali, and Rackoff, 1989) to prove that a public statement, given some private witness, can be decided to be true due to some polynomial-time computable binary relation R. Note that $(\phi, \omega) \in R$ denotes that ϕ is true regarding R given witness ω, and we call ϕ the (public) statement and call ω the (private) witness. Usually, zk-SNARK consists of a triple of algorithms, including Setup, Prover, and Verifier. The Setup algorithm produces a common reference string used throughout the other algorithms. The Prover algorithm, given

the public ϕ and the private ω, produces a proof π, attesting $(\phi, \omega) \in R$. The Verifier, on input the public ϕ and the proof π, returns a bit to reject (0) or accept (1) the proof.

The above zk-SNARK proof system shall satisfy the following security properties:

- *Completeness*: For $(\phi, \omega) \in R$, the prover can always use the witness ω to generate a valid proof regarding the correctness of the statement ϕ, thus passing the verification of the verifier.
- *Zero-Knowledge*: The proof reveals nothing beyond the validity of the statement, i.e., no bit of the private witness ω shall be leaked.
- *Proof-of-Knowledge*: A zk-SNARK satisfies soundness, if the prover cannot efficiently compute a valid proof for $(\phi, \omega) \notin R$. Sometimes, we further need an even stronger soundness property called proof-of-knowledge, that means it is computationally infeasible for any malicious prover to generate a proof without knowing a valid witness ϕ s.t. $(\phi, \omega) \in R$.

In addition to the above security features, zk-SNARK also satisfies:

- *Succinctness*: The proof is short and can be verified quickly (e.g., polylogarithmic in the circuit size of relation R).
- *Non-interactivity*: The proof is generated and published without requiring back-and-forth interaction between the prover and the verifier.

Note that both succinctness and non-interactivity are critical in the blockchain setting (Kosba, Miller, Shi, Wen, and Papamanthou, 2016): succinctness is essential due to the limited on-chain resources available for verifying complex proofs, while non-interactivity is necessary because consensus nodes cannot engage in back-and-forth communication with the prover during the execution of the consensus protocol.

3.5 Two-Party Computation (2PC) with Malicious Security

A two-party computation (2PC) protocol (Beaver, Micali, and Rogaway, 1990; Ben-Or, Goldwasser, and Wigderson, 1988; Goldreich, Micali, and Wigderson, 1987; Yao, 1982) allows two mutually distrustful parties to jointly compute a function over their private inputs while guaranteeing the correctness and privacy of the honest party, even if one of them is corrupted. A 2PC protocol with malicious security (Lindell and Pinkas, 2007; Shelat and Shen, 2011; Wang, Ranellucci, and Katz, 2017) further ensures these guarantees even when the corrupted party may arbitrarily deviate from the prescribed protocol to influence the computation or extract additional information.

However, a fundamental limitation of 2PC protocols lies in their lack of fairness assurance (in the general case) (Cleve, 1986): a malicious participant may obtain the function's output while simultaneously preventing the honest party from learning the result. Our design of a fair TLS data oracle must therefore overcome this inherent limitation of 2PC by leveraging the blockchain and tVDF, thus achieving fairness.

3.6 Abstract of Blockchain

We view the blockchain (which also supports smart contracts) as an ideal bulletin board $\mathcal{F}_{\mathcal{L}}$ (Choudhuri, Green, Jain, Kaptchuk, and Miers, 2017), which also maintains a dictionary $\mathcal{L}$ to track the balance of each user or smart contract. For example, after Alice transfers Ξx to Bob (assuming $\mathcal{L}[\mathsf{Alice}] \geq \Xi x$), then the balance of Alice $\mathcal{L}[\mathsf{Alice}]$ decreases by Ξx and the balance of Bob $\mathcal{L}[\mathsf{Bob}]$ increases by Ξx. Here we let Ξ denote the unit of cryptocurrency on the smart-contract-enabled blockchain.

4 TLS Oracle from "Encrypt-TLS-keys-to-future"

This section presents our protocol for extracting web data from TLS servers in a way that enables the verifier alone to eventually provide the public verifiability of already-transmitted web data. On one side, the design just relies on the non-colluding nature of the prover (i.e., the user client) and the verifier (i.e., the web oracle service provider), instead of assuming a trusted third party (that acts as a designated verifier to honestly forward the extracted TLS data to the public). On the other side, the design also achieves fairness to some extent: once some TLS data was transmitted, a malicious verifier no longer can prevent the prover from selectively disclosing the web data to the public in a verifiable way.

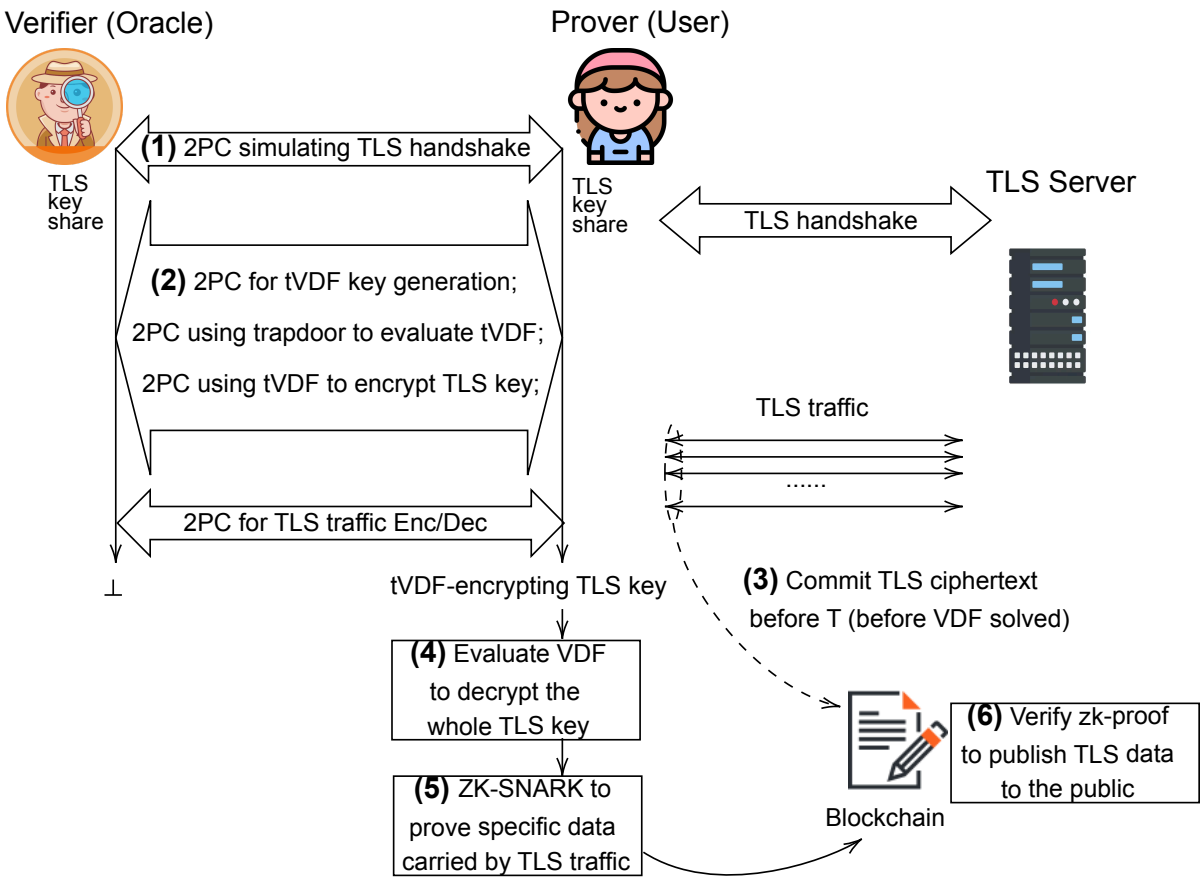

Fig. D1: Execution flow of our TLS oracle protocol with on-chain verifiability and a form of fairness.

4.1 Our TLS Oracle Protocol

The protocol integrates secure two-party computation (2PC) (Wang et al., 2017), a trapdoor verifiable delay function (tVDF) (Wesolowski, 2020), and zero-knowledge proof techniques (Groth, 2016), and is built atop the modern PKI (Cooper et al., 2008; ISO, 2022) and TLS protocols (Dierks and Rescorla, 2008; Rescorla, 2018). The involved parties include: (i) a smart-contract-enabled blockchain; (ii) a website hosting a user's critical data, having a digital certificate and running the standard TLS protocol; (iii) a user as prover and a oracle service provider as verifier, who engage in a collaborative TLS session to emulate a TLS client, thus extracting authentic data from the web server and later collectively disclosing the extracted data on the blockchain.

Protocol Overview. From a very high level, our TLS oracle protocol has an execution flow as Figure D1 illustrates, which proceeds in the following steps:

1. In this phase, the verifier and prover jointly execute a secure two-party computation (2PC) protocol to emulate a TLS client, while the TLS server continues to operate according to the standard TLS specification. At the end of the handshake, neither the prover nor the verifier individually learns the full symmetric TLS session keys. Instead, each party obtains an additive secret share of the session keys, which will later serve as input to subsequent 2PC protocols for AES-GCM encryption and decryption of TLS traffic.
2. Time-lock of TLS key: Next, the verifier and the prover jointly execute another 2PC protocol to perform a sequence of tasks that collectively encrypt the TLS session keys "to the future". These tasks include: (i) the distributed setup of a tVDF, (ii) trapdoor evaluation of the generated tVDF, and (iii) encryption of the TLS session keys using the tVDF output. Since we instantiate the tVDF using primitives over an RSA group, these 2PC computations can be securely realized by adapting existing maliciously secure protocols for secure RSA modulus generation (Burkhardt, Damgård,

Frederiksen, Ghosh, and Orlandi, 2023; Frederiksen, Lindell, Osheter, and Pinkas, 2018; Hazay, Mikkelsen, Rabin, Toft, and Nicolosi, 2019). [92]

3. Commitment of TLS traffic in time: The prover records the transcript of TLS traffic (e.g., AEM-GCM ciphertexts) on the blockchain, before a time that VDF can be feasibly computed without access to the trapdoor.
4. Prover gets TLS key by solving VDF: The prover solves VDF, thus decrypting the TLS session keys.
5. Prover leverages zk-SNARK to prove that specific data is carried in the authenticated TLS ciphertext.
6. The blockchain verifies the zero-knowledge proof, ensuring that specific data is indeed obtained from decryption of the committed TLS ciphertext.

Security Intuition. Informally, the above protocol achieves the following properties (conditioned on that the faithfulness of TLS server and blockchain):

- *Correctness.* If both the verifier and the prover are honest, the authentic data (that is hosted by the TLS server) can be published on the blockchain. This is trivial due to the correctness of all underlying cryptographic schemes.
- *Authenticity against malicious prover.* Despite that the prover could be arbitrarily malicious, the TLS data finally published on the blockchain must be authenticated from the TLS server. This is because the tVDF-encrypting TLS session key is committed to the blockchain (and secretly shared between the verifier and the prover), such that the prover cannot forge authenticated TLS traffic and publish fake TLS traffic on the blockchain before tVDF can be solved.
- *Fairness against malicious verifier.* Since tVDF can finally be solved by the prover, the TLS traffic posted on the blockchain finally can be solely decrypted by the prover (without the involvement of verifier). As such, the prover can always disclose certain TLS traffic (after solving tVDF) if such traffic is indeed transferred from the server to the prover.
- *Private selective disclosure.* The prover can only disclose certain non-sensitive data on the blockchain. This is because (i) the TLS session key is only obtained by the prover after tVDF is solved, and (ii) the zero-knowledge proof from the prover to disclose certain data does not leak additional information.

Efficiency consideration. Efficient instantiation of the above protocol workflow requires addressing several significant efficiency challenges. First, the 2PC protocol for tVDF key generation is highly expensive, as it requires the distributed generation of an RSA modulus (Burkhardt et al., 2023; Frederiksen et al., 2018; Hazay et al., 2019), which is well-known very costly. Moreover, the subsequent 2PC protocol for tVDF's trapdoor evaluation is also expensive.

However, the efficiency of these critical steps can be improved by the following steps: (i) we can enhance tVDF with an extra property of completeness under malicious setup (Chen et al., 2025), such that the prover can always solve tVDF even if the setup of tVDF is done by a malicious party; (ii) then let the verifier to locally run the key generation for tVDF, instead of a 2PC protocol; (iii) finally, we can let the verifier locally run the trapdoor evaluation of tVDF, and that the 2PC protocol for tVDF evaluation can be replaced with a less expensive 2PC protocol for tVDF verification.

4.2 Why not the Alternative Way through Modifying DECO?

One might suggest that our TLS oracle design is not the only possible approach to enabling public verifiability for prior designs such as DECO. In fact, an alternative modification to DECO (Zhang et al., 2020) could enable on-chain verification: instead of committing TLS traffic (Dierks and Rescorla, 2008; Rescorla, 2018) only to the verifier, the prover could commit the ciphertext directly to the blockchain. Then, once the verifier transmits its TLS key share to the prover, the prover can reconstruct the full TLS session keys and prove the correctness of the plaintext corresponding to the committed ciphertext.

However, this approach suffers from a fundamental drawback: it requires the verifier to reliably return and reveal its key share after the TLS communication has completed. This dependency makes the design unsuitable for fair exchange scenarios between fiat and crypto assets, particularly when the verifier is the party transacting fiat currency through the TLS session. If the verifier refuses to reveal

[92] Note that while several optimizations can be applied to improve efficiency of this step, we present a plain version of our oracle protocol for clarity and ease of understanding.

its key share, the prover is unable to complete the proof, even if the TLS event (e.g., payment) did in fact occur. This undermines fairness by enabling strategic aborts, where the verifier can benefit from the transaction while preventing the prover from proving it, thus introducing potential vulnerability in certain applications.

5 Application to Exchange across Web2 and Web3

A TLS oracle protocol with our form of fairness (i.e., the ability to prove certain authenticated data that has already been transmitted, even if the verifier aborts after the transmission) can serve as a critical building block for enabling fair peer-to-peer (P2P) exchange between fiat and cryptocurrency assets, such as between the balance of digital banks and Ethereum tokens (Wood et al., 2014).

Consider a scenario in which a Web3 user, Alice, wishes to exchange her cryptocurrency for fiat currency from a Web2 user, Bob, who completes the fiat payment via a TLS-secured connection to a trusted bank. In traditional protocols, fairness cannot be guaranteed without involving a trusted third party, as Alice could refuse to cooperate in helping Bob prove that he completed the fiat transfer.

In contrast, with our fairness-preserving oracle protocol, Bob is assured that even if Alice aborts the protocol, he can still extract a cryptographic proof of the fiat transaction's completion (e.g., proving that the bank indeed sent a webpage acknowledging a successful transfer). This property eliminates the dependency on Alice's continued cooperation and ensures that the success of the exchange depends solely on the verifiable occurrence of the payment event. Consequently, our protocol establishes a foundation for fair cross-domain asset exchange, bridging Web2 and Web3 ecosystems without reliance on external intermediaries.

Informally, the resulting decentralized P2P exchange protocol attains the following security guarantees:

- *Completeness.* If both Alice and Bob are honest, then Alice receives $\$b$ (i.e., $\Lambda[\mathsf{Alice}]$ increased by $\$b$), and Bob receives Ξa (i.e., $\mathcal{L}[\mathsf{Bob}]$ increased by Ξa).
- *Fairness.* Either (i) Alice receives $\$b$ digital asset and Bob receives Ξa cryptocurrency, or (ii) their cryptocurrency and digital asset balances remain unchanged.
- *Privacy.* Bob's private login credentials (e.g., bank passwords) remain confidential.

For space limit, here we only provide a very brief intuition of our P2P exchange across the Web2 and Web3 domains. Readers interested in further details may refer to our technical paper (Chen et al., 2025).

6 Conclusion and Future Directions

We presented a new TLS oracle protocol that achieves public verifiability and introduces a novel form of fairness, even when one party becomes uncooperative. At its core is the "encrypt-TLS-keys-to-future" paradigm, which allows a prover to reconstruct the TLS session key after a verifiable delay and use zero-knowledge proofs to attest to past TLS communications. Unlike prior approaches such as DECO, our protocol does not rely on a designated verifier remaining online and honest to enable the prover to complete its proof for the extracted TLS data.

To demonstrate the power of our design, we applied it to the problem of cross-domain fair exchange between fiat and crypto currencies. By allowing the fiat holder to (solely) extract the publicly verifiable evidence of a completed bank transaction over TLS, our protocol enables fair peer-to-peer asset exchange without reliance on trusted intermediaries. This construction addresses a long-standing challenge in interoperability between Web2 and Web3 systems.

Our work opens several avenues for future research. First, performance optimizations to the underlying 2PC schemes and other cryptographic protocols could reduce latency and improve scalability, making our paradigm practical for real systems. Second, expanding our model to support more expressive TLS-dependent statements would broaden its applicability beyond currency exchange across domains.

Acknowledgements. Yuan Lu is supported in part by the National Key Research and Development Project of China (2022YFB3102500), the Chinese Academy of Sciences (CAS) Project for Young Scientists in Basic Research (YSBR-035), and the National Natural Science Foundation of China Key Program (No.92270204). Qiang Tang is supported in part by the Australian Research Council (ARC) Discovery Project (DP250101739) and in part by the Research Awards from the Ethereum Foundation, Protocol Laboratories, and the Stellar Development Foundation.

Declarations

All authors certify that they have no affiliations with or involvement in any organization or entity with any financial interest or non-financial interest in the subject matter or materials discussed in this manuscript.

References

Backman A, Richer J, Sporny M (2024) Http message signatures. Tech. Rep. RFC 9421, Network Working Group

Beaver D, Micali S, Rogaway P (1990) The round complexity of secure protocols. In: Proceedings of the twenty-second annual ACM symposium on Theory of computing, pp 503–513

Ben-Or M, Goldwasser S, Wigderson A (1988) Completeness theorems for non-cryptographic fault-tolerant distributed computation. In: Proceedings of the twentieth annual ACM symposium on Theory of computing, pp 1–10

Ben-Sasson E, Chiesa A, Tromer E, et al (2014) Succinct {Non-Interactive} zero knowledge for a von neumann architecture. In: 23rd USENIX Security Symposium (USENIX Security 14), pp 781–796

Boneh D, Bonneau J, Bünz B, et al (2018) Verifiable delay functions. In: Annual international cryptology conference, Springer, pp 757–788

Burkhardt J, Damgård I, Frederiksen TK, et al (2023) Improved distributed rsa key generation using the miller-rabin test. In: Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security, pp 2501–2515

Chen L, Liu Y, Lu Y, et al (2025) Short paper: Decentralized fair exchange for assets across blockchain and the internet. In: 2025 Crypto Valley Conference (CVC), IEEE, pp 102–105

Choudhuri AR, Green M, Jain A, et al (2017) Fairness in an unfair world: Fair multiparty computation from public bulletin boards. In: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, pp 719–728

Cleve R (1986) Limits on the security of coin flips when half the processors are faulty. In: Proceedings of the eighteenth annual ACM symposium on Theory of computing, pp 364–369

Cooper D, Santesson S, Farrell S, et al (2008) Internet x. 509 public key infrastructure certificate and certificate revocation list (crl) profile. Tech. rep.

Dierks T, Rescorla E (2008) The transport layer security (tls) protocol version 1.2. Tech. Rep. RFC 5246, Network Working Group

Dworkin M (2007) Recommendation for block cipher modes of operation: Galois/Counter Mode (GCM) and GMAC. Tech. Rep. NIST Special Publication 800-38D, National Institute of Standards and Technology (NIST)

Dworkin MJ, Barker E, Nechvatal JR, et al (2001) Advanced encryption standard

Frederiksen TK, Lindell Y, Osheter V, et al (2018) Fast distributed rsa key generation for semi-honest and malicious adversaries. In: Annual International Cryptology Conference, Springer, pp 331–361

Gennaro R, Gentry C, Parno B, et al (2013) Quadratic span programs and succinct nizks without pcps. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques, Springer, pp 626–645

Goldreich O, Micali S, Wigderson A (1987) How to play any mental game. In: Proceedings of the nineteenth annual ACM symposium on Theory of computing, pp 218–229

Goldwasser S, Micali S, Rackoff C (1989) The knowledge complexity of interactive proof systems. SIAM Journal on Computing 18(1):186–208

Groth J (2016) On the size of pairing-based non-interactive arguments. In: Annual international conference on the theory and applications of cryptographic techniques, Springer, pp 305–326

Hazay C, Mikkelsen GL, Rabin T, et al (2019) Efficient rsa key generation and threshold paillier in the two-party setting. Journal of Cryptology 32(2):265–323

ISO (2022) Information technology — public key infrastructure — practices and policy framework. Tech. Rep. ISO/IEC 27099:2022

Kosba A, Miller A, Shi E, et al (2016) Hawk: The blockchain model of cryptography and privacy-preserving smart contracts. In: 2016 IEEE symposium on security and privacy (SP), IEEE, pp 839–858

Krawczyk H, Bellare M, Canetti R (1997) Hmac: Keyed-hashing for message authentication. Tech. Rep. RFC 2104, Network Working Group

Labs P (2017) Filecoin: A decentralized storage network. Tech. rep., URL https://filecoin.io/filecoin.pdf, whitepaper

Lauinger J, Ernstberger J, Finkenzeller A, et al (2025) Janus: Fast privacy-preserving data provenance for tls. Proceedings on Privacy Enhancing Technologies

Lindell Y, Pinkas B (2007) An efficient protocol for secure two-party computation in the presence of malicious adversaries. In: Annual international conference on the theory and applications of cryptographic techniques, Springer, pp 52–78

McGrew D (2008) An interface and algorithms for authenticated encryption. Tech. Rep. RFC 5116, Network Working Group

Nakamoto S (2008) Bitcoin: A peer-to-peer electronic cash system. Available at SSRN 3440802

Nir Y, Josefsson S, Pégourié-Gonnard M (2018) Elliptic curve cryptography (ecc) cipher suites for transport layer security (tls) versions 1.2 and earlier. Tech. rep.

Pietrzak K (2019) Simple verifiable delay functions. In: 10th innovations in theoretical computer science conference (itcs 2019), Schloss Dagstuhl–Leibniz-Zentrum für Informatik, pp 60–1

Qualys (????) SSL Pulse. https://www.ssllabs.com/ssl-pulse/, accessed: 2025-2-28

Rescorla E (2000) Http over tls. Tech. Rep. RFC 2818, Network Working Group

Rescorla E (2018) The transport layer security (tls) protocol version 1.3. Tech. Rep. RFC 8446, Internet Engineering Task Force (IETF)

Rogaway P (2002) Authenticated-encryption with associated-data. In: Proceedings of the 9th ACM Conference on Computer and Communications Security, pp 98–107

Salowey J, Choudhury A, McGrew D (2008) Aes galois counter mode (gcm) cipher suites for tls. Tech. Rep. RFC 5288, Network Working Group

Shelat A, Shen CH (2011) Two-output secure computation with malicious adversaries. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques, Springer, pp 386–405

Wang X, Ranellucci S, Katz J (2017) Authenticated garbling and efficient maliciously secure two-party computation. In: Proceedings of the 2017 ACM SIGSAC conference on computer and communications security, pp 21–37

Wesolowski B (2020) Efficient verifiable delay functions. Journal of Cryptology 33(4):2113–2147

Wood G, et al (2014) Ethereum: A secure decentralised generalised transaction ledger. Ethereum project yellow paper 151(2014):1–32

Xie X, Yang K, Wang X, et al (2024) Lightweight authentication of web data via garble-then-prove. In: Proceedings of the 33rd USENIX Conference on Security Symposium, pp 1957–1974

Yao AC (1982) Protocols for secure computations. In: 23rd annual symposium on foundations of computer science (sfcs 1982), IEEE, pp 160–164

Zhang F, Cecchetti E, Croman K, et al (2016) Town crier: An authenticated data feed for smart contracts. In: Proceedings of the 2016 ACM SIGSAC conference on computer and communications security, pp 270–282

Zhang F, Maram D, Malvai H, et al (2020) Deco: Liberating web data using decentralized oracles for tls. In: Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security, pp 1919–1938