PECO: methods to enhance the privacy of DECO protocol

Manuel B. Santos
manuel.batalha.dos.santos@ist.utl.pt
Instituto de Telecomunicações
Lisboa, Portugal
Departamento de Matemática, Instituto Superior Técnico, Universidade de Lisboa
Lisboa, Portugal

ABSTRACT

The DECentralized Oracle (DECO) protocol enables the verifiable provenance of data from Transport Layer Security (TLS) connections through secure two-party computation and zero-knowledge proofs. In this paper, we present PECO, an extension of DECO that enhances privacy features through the integration of two new private three-party handshake protocols (P3P-HS). PECO allows any web user to prove to a verifier the properties of data from TLS connections without disclosing the identity of the servers. Like DECO's three-party handshake protocol, PECO's P3P-HS methods do not require any changes on the server side. PECO offers two options: one that provides $k-$anonymity for the server's identity, and another that completely masks the server's identity from the verifier. PECO is based on three main protocols: (a) commit-and-proof zero-knowledge proofs (CP-ZKP) that enable the proof of relations under committed values in zero-knowledge, (b) verification of Elliptic Curve Digital Signature Algorithm (ECDSA) signatures under a committed public key without revealing the key (zkAttest), and (c) a proof of membership to verify that a committed key belongs to a set of keys. We estimate the performance of both P3P-HS protocols and compare it to TLS timeout using state-of-the-art implementations.

KEYWORDS

privacy, web attestation, zero-knowledge, TLS

1 INTRODUCTION

The Transport Layer Security (TLS) protocol is essential for providing secure communication between users and servers, ensuring the confidentiality and integrity of data. However, it does not allow third parties to verify the source of TLS data. The DECentralized Database (DECO) protocol [22] addresses this issue by enabling the ability to prove statements about data from TLS connections, without requiring permission from a central server. This allows for the provenance of data to be traced back to central points while maintaining the privacy of the data. For example, the DECO protocol is crucial for oracles in smart contracts, as it enables them to use data from other sources without any changes on the server side.

While DECO allows the data being accessed to be hidden, it does not conceal metadata such as the identity of the servers being accessed. This can be problematic in some cases, as the disclosure of server identities may reveal browsing habits and search patterns that raise significant privacy concerns [19, 14]. In other cases, however, the identification of data sources may be necessary.

This research aims to develop protocols that enable any user to prove statements about TLS data to any verifier, without revealing the identity of the server being accessed to the verifier. To this end, we present two protocols based on DECO: the k-anonymous Private dECentralized Oracle (k-PECO) and the Private dECentralized Oracle (PECO). The k-PECO protocol allows the user to hide the identity of the server from a set of k possible servers, while the PECO protocol enables the complete concealment of the server's identity. In certain situations, the k-PECO protocol may be most suitable for achieving the desired outcome, as it enables the demonstration that the data originates from a set of certified or qualified servers while preserving the privacy of the user's browsing activity. These systems could be used as privacy-preserving attestation mechanisms for browsing behavior, replacing current CAPTCHA systems with a zero-knowledge proof-of-behavior.

The main contributions of this paper are:

(1) The design of a protocol that enables users to prove statements about TLS data without revealing their browsing behavior by hiding the identity of the server;
(2) The evaluation of the proposed protocols using state-of-the-art implementation tools to estimate their performance.

2 BUILDING BLOCKS

2.1 Preliminaries

We write as $\mathbb{G}$ the set of elements, denoted by capital letters such as G and H, that form a group on an elliptic curve. $\mathbb{G}$ has an order of q and is generated by a point $G \in \mathbb{G}$ over the finite field $\mathbb{Z}_p$ of integers modulo a prime p.

The notation $\mathbf{com}(s)$ represents the commitment of a secret value s. This is an important building block in PECO and, throughout our exposition, we do not instantiate the type of commitment scheme. However, in practice, we use the Pedersen commitment scheme, which is a type of commitment scheme that is unconditionally hiding, computationally binding, and additively holomorphic [18]. The Pedersen commitment scheme also has a specific protocol for proving knowledge about the opening of a commitment without disclosing the opening to the verifier.

The PECO protocol includes a signature algorithm, which is a mathematical method used to verify the authenticity and integrity of a message m. The signature algorithm consists of two algorithms: Sign and Verify. The Sign algorithm is used to create a digital signature for a message by inputting the message and a secret key, sk, known only to the sender into the algorithm. The resulting signature value is attached to the message. We write:

$$\mathsf{DS} = \mathsf{Sign}(m, \mathsf{sk}).$$

The verification algorithm, Verify, is used by the recipient of the message to check the validity of the signature by inputting the

message, signature, and the sender's public key into the algorithm. We write:

$$\mathsf{v} = \mathsf{Verify}(m, \mathsf{DS}, \mathsf{pk}).$$

If the signature is valid ($\mathsf{v} = 1$), it confirms that the message has not been tampered with and was indeed sent by the owner of the secret key. We assume the hash function used in the signature scheme to be SHA256. This means the both Sign and Verify algorithms start by hashing the message m before signing or verifying the hash value $\mathsf{h}_m = \mathsf{SHA256}(m)$.

2.2 X.509 certificates

X.509 certificates are a widely used tool for ensuring authenticity and confidentiality on the Internet. They work by establishing a chain of trust, where the validity of each certificate is attested by the next in the chain.

We represent a certificate chain as a tuple

$$\mathsf{cert} = (\mathsf{c}_0, \mathsf{c}_1, \ldots, \mathsf{c}_n),$$

where c_0 is the certificate of some server S and c_n is the root certificate authority certificate. The public key of the entity that signs the certificate c_{i-1} is denoted as pk_i. Therefore, pk_1 is the public key of the entity that signs the server's certificate c_0 and pk_0 is the public key of the server S.

Each certificate has a message m and a digital signature DS_m. The message m is divided into various fields, such as the organization name, the server's email, public key, expiration duration, signature and public key algorithms, among others. Some of these fields, such as the server's email, can be sensitive and reveal the identity of the server being accessed. While this is generally acceptable, it can pose a privacy issue in the context of CAPTCHAs, where the user may unknowingly disclose the website they are accessing by sending the server's certificate to the verifier. Consequently, masking the identity of the server being accessed is an important step in the PECO protocol.

2.3 DECO

We start by providing an overview of the DECentralized Database [22] (DECO) protocol. Then, we highlight the steps in DECO that disclose the server's identity.

Structure and goal. The DECO protocol involves three parties: a prover P, a verifier V, and a website server S. It enables P to prove to V that some TLS data originated from S (proof of provenance) and statements about the data in a zero-knowledge manner, without revealing the data itself. The functionality of DECO is summarized in Figure 1.

Protocol. The DECO protocol consists of three phases:

(1) Three-party handshake phase: During this phase, the parties generate session keys in a way that prevents P from forging TLS sessions with S. This phase is further divided into two steps: key exchange step and key derivation step. In the key exchange step, P and V begin by verifying S's certificate and the corresponding signature of the exchanged key (we refer to this as the attestation phase). Then, P and V generate additive shares of the symmetric pre-master key shared with S. In the key derivation step, the parties derive the session keys (MAC and encryption keys). Note that during this last step, V does not communicate with S;
(2) Query phase: During this phase, P builds the queries together with V using secure two-party computation, and P sends them to S.;
(3) Proof generation phase: During this phase, P proves that the query was correctly generated and that the response from S satisfies some statement.

$\mathcal{F}_{\text{Oracle}}$ **Functionality between** V, P **and** S

P **input:** Θ_s.
V **input:** query template Query and a statement stmt.

(1) If at any point during the session, a message $(\text{sid}, \text{receiver}, m)$ with receiver $\in \{\text{S}, \text{P}, \text{V}\}$ is received from A, forward (sid, m) to receiver and forward any responses to A.
(2) Upon receiving input $(\text{sid}, \text{Query}, \text{stmt})$ from V, send $(\text{sid}, \text{Query}, \text{stmt})$ to P. Wait for P to reply with "ok" and Θ_s.
(3) Compute $\mathcal{Q} = \text{Query}(\Theta_s)$ and send $(\text{sid}, \mathcal{Q})$, to S and record its response $(\text{sid}, \mathcal{R})$. Send $(\text{sid}, |\mathcal{Q}|, |\mathcal{R}|)$ to A.
(4) Send $(\text{sid}, \mathcal{Q}, \mathcal{R})$ to P and $(\text{sid}, \text{stmt}(\mathcal{R}), \text{S})$ to V.

Figure 1: The oracle functionality [22].

In the key exchange step of the three-party handshake phase, we notice that the identity of S is only revealed to V during the attestation phase, where V verifies the certificate sent by S to P. Our main goal in this paper is to replace this attestation phase with a privacy-preserving version, in which V can verify that the data comes from a certified S but does not know which one specifically.

2.4 Secure multiparty computation

In secure multiparty computation (SMC), multiple parties, each with their own input, can jointly compute a function without disclosing their inputs to one another. This can be thought of as each party sending their input to a trusted third party, who then computes the function and returns the output to each party. The first solution to SMC was proposed by Yao in 1982 [20], who introduced the concept of garbled circuits, a key element in secure computation. While initially limited to only two parties, the protocol was later generalized to more parties by GMW[11], BGW [3] and BMR [2]. In order to improve performance, several implementation optimizations, including point-and-permute [2], row reduction [17], FreeXOR [15] and half gates [21], have been developed for the Yao's garbled circuit protocol.

2.5 Zero-knowledge proofs

Zero-knowledge proof (ZKP) systems were first introduced by Goldwasser, Micali, and Rachoff in 1985 [12], and later extended by Blum,

Feldman, and Micali to the non-interactive case in 1988 [6], under the assumption that the parties shared a common random string (CRS model). Since then, numerous protocols have been developed, with more efficient versions being introduced. One particularly important ZKP system from a practical perspective is the zero-knowledge succinct non-interactive argument of knowledge (zk-SNARK) [5]. Informally, zk-SNARK has the following properties:

(1) Completeness: The verifier always validates the prover's proof if both parties follow the protocol;
(2) Soundness (argument of knowledge): The prover can only convince the verifier if the proof holds.;
(3) Zero-knowledge: The verifier receives no information other than the validity of the statement being proved.
(4) Non-interactive: The prover sends a single message to the verifier.
(5) Succinctness: The proof is very short compared to the input, and it has a fast verification procedure.

More importantly for this work, zk-SNARKs can be extended to commit-and-prove zkSNARKs (CP-SNARKs) [8], which allow a prover to demonstrate knowledge of a certain relation with respect to a committed witness. This is useful because it allows multiple proofs to be composed based on the same committed input. In other words, the CP property ensures that the prover, denoted as P, uses the same input value to prove different statements to the verifier, denoted as V. Without this commitment, a malicious prover could potentially cheat the verifier. We will see in Section 3.2 that a malicious P can cheat V in case he is not committed to using the same input to prove two different statements.

We use the standard notation introduced by Camenisch-Stadler [7] to describe the general goal of zero-knowledge proofs:

$$\mathsf{ZK} - \mathsf{PoK}_x \{w : \mathcal{L}(w, x)\},$$

where the statement x is public information, the witness w is private information known only to the prover and the language $\mathcal{L}$ represents the condition that the statement and witness must satisfy. In cases where it is clear from context, the under-script x may be omitted from $\mathsf{ZK} - \mathsf{PoK}_x$.

2.6 zkAttest

The zkAttest protocol, introduced in the paper [10], allows a prover to prove the validity of an Elliptic Curve Digital Signature Algorithm (ECDSA) signature to a verifier, without revealing the actual public key used to create the signature. This is achieved through the use of a special made $\sum-$protocol, rather than the more commonly used SNARK-based approach.

One key feature of the zkAttest protocol is that the public key is committed, which means that the prover can prove additional properties about the key to the verifier. This allows for the creation of more complex and robust zero-knowledge proofs (ZKPs) that can be composed with other ZKPs. This is particularly useful in the context of the PECO protocols, which are presented in Section 3 of the paper. We refer to this protocol as Π_{zkAttest}. In summary, zkAttest allows for the following proof:

$$\mathsf{ZK} - \mathsf{PoK} \{\mathsf{pk} : \mathbf{com}(\mathsf{pk}) = \boldsymbol{c} \wedge \mathsf{Verify}(\mathsf{pk}, \sigma, m) = 1\},$$

where σ is the signature received and m is the message signed.

Overall, the zkAttest protocol offers a powerful and flexible tool for proving the validity of ECDSA signatures in a privacy-preserving manner. It has the potential to be used in a variety of applications where the ability to prove the authenticity of digital signatures is important, such as in the realm of secure authentication and authorization.

2.7 Proof of membership

Groth and Kohlweiss [13] recently proposed a $\sum-$protocol that allows a prover to demonstrate to a verifier that, from a set of public commitments, he knows the opening of one of the commitments to the value zero. This protocol has various potential applications, including demonstrating that a committed value is part of a publicly known set of values. In particular, a prover can use this protocol to prove to a verifier that his key belongs to a particular public set of keys. In summary, the proof of membership protocol allows the following proof:

$$\mathsf{ZK} - \mathsf{PoK} \{\mathsf{pk} : \mathbf{com}(\mathsf{pk}) = \boldsymbol{c} \wedge \mathsf{pk} \in \mathcal{S}\},$$

where $\boldsymbol{c}$ is the public committed value and $\mathcal{S}$ is the public set of keys. We denote by Π_{pm} the protocol that achieves such a proof.

One notable aspect of the Π_{pm} protocol is its relatively low communication complexity, which is logarithmic. In terms of computation, the protocol requires $O(N \log N)$ and $O(N)$ multiplications in Z_q for the prover and verifier, which is more efficient compared to other protocols such as the one proposed by Bayer and Groth in [1], which requires $O(N \log^2 N)$ multiplications for both the prover and verifier.

The combination of the protocols Π_{pm} and Π_{zkAttest} is a straightforward process due to the shared use of Pedersen commitments. This allows both protocols to use the same commitment value $\boldsymbol{c}$ and prove statements about it independently. The composition of these protocols was first introduced and implemented in the original zkAttest article [10], and it serves as a crucial building block for one of the privatization approaches of the DECO protocol.

3 PRIVATE DECO

The main goal of the Private DECO (PECO) functionality is to allow a user to verify statements about certified data without revealing its source. In this section, we will introduce two protocols that achieve this goal while offering different levels of anonymity. The $k-$anonymous PECO ($k-$PECO) obscures the identity of the server with $k-$anonymity, meaning it can be proven that the server belongs to a set of k servers without disclosing its specific identity. The PECO protocol completely hides the identity of the server.

We will begin by defining the $k-$PECO and PECO functionalities. Then, we present a prototype protocol that fails to realize PECO but serves as a motivation for the development of both $k-$PECO and PECO protocols.

3.1 Functionality definition

Like the DECO protocol, both $k-$PECO and PECO protocols involve three parties: a prover (P), a verifier (V) and a website server (S). These protocols allow P to prove some statement stmt to V about a query Query sent to S, without fully disclosing S identity to V. The DECO functionality is formally described in Figure 1. The $k-$PECO

functionality is denoted by $\mathcal{F}_{\text{kp-Oracle}}$ and the PECO functionality by $\mathcal{F}_{\text{p-Oracle}}$.

The difference between the two proposed functionalities ($\mathcal{F}_{\text{kp-Oracle}}$ and $\mathcal{F}_{\text{p-Oracle}}$) and the DECO functionality ($\mathcal{F}_{\text{Oracle}}$) is that $\mathcal{F}_{\text{kp-Oracle}}$ and $\mathcal{F}_{\text{p-Oracle}}$ functionalities do not send the identity S to V (Step 4 in Figure 1). Instead, we have that:

- $\mathcal{F}_{\text{kp-Oracle}}$ sends $(\text{sid}, \text{stmt}(\mathcal{R}), \mathcal{S})$ where $\mathcal{S}$ is a set of identities with size k;
- $\mathcal{F}_{\text{p-Oracle}}$ only sends $(\text{sid}, \text{stmt}(\mathcal{R}))$.

3.2 Strawman protocol

The following strawman protocol serves as a starting point for the development of two robust protocols that aim to protect the identity of S from being revealed to V. To achieve this goal, we will need to make modifications to DECO so that the identity of S is effectively concealed from V.

In DECO, the identity of S is only revealed to V during the key exchange phase (Step 1) of the three-party handshake (3P-HS) protocol. In the original 3P-HS protocol, P first receives a certificate, the server nounce r_S, and a signed ephemeral DH public key $Y_S = s_S \cdot G$ from S. After verifying the certificate and the signature, P forwards them to V, who performs the same verification process. The certificate contains the identity of S, and V uses the server's public key to verify the signature of Y_S. As a result, at this point in the protocol, V knows the identity of S.

To conceal the identity of S from V, we propose a new private 3P-HS (P3P-HS) protocol which modifies the original 3P-HS protocol to allow V to verify the certificate and the signature of Y_S without accessing identity-sensitive information or knowing S's public key. This privacy-preserving attestation mechanism is essential for ensuring the privacy of P while still allowing V to verify the authenticity of the certificate and the signature. When integrated into DECO, this new P3P-HS protocol forms the basis for $k-$PECO and PECO.

Recall that $\text{cert} = (\text{c}_0, \text{c}_1, \ldots, \text{c}_n)$ represents the certificate hierarchy, where c_0 is the certificate of the server S and c_n is the root certificate authority certificate. We denote by $\text{cert}[1:] = (\text{c}_1, \ldots, \text{c}_n)$ a slice of cert without the first certificate c_0. The public key of the entity that signs the server's certificate, c_0, is denoted as pk_1, and the digital signature in c_0 is denoted as DS. The public key of the server S is denoted as pk_0.

Strawman protocol. A strawman protocol for PECO is presented in Figure 2 and is denoted by $\Pi^{\text{P3P-HS}}_{\text{straw}}$. As previously mentioned, S's identity is only revealed during the 3P-HS phase while attesting both the certificate and Y_S's signature.

It's important to note that in the certification chain, we only need to conceal the message (m) in the first certificate, as the other certificates do not reveal the identity of the server. Therefore, the certificates up to the server's certificate can be checked in the clear without compromising the privacy of S. This is achieved in steps 4 and 5 of the $\Pi^{\text{P3P-HS}}_{\text{straw}}$ protocol (Figure 2). During these steps, P only sends the required material to V for verifying the validity of the digital signature DS present in c_0. This material includes the certificates $\text{cert}[1:]$, the signature DS, and the hash value hm. V can then use the public key pk_1 from c_1 to verify the signature DS against the hash value h_m. This masks the identity of the server because the hash value h_m does not reveal any information about the message m, which contains S's identity. In step 6 of the $\Pi^{\text{P3P-HS}}_{\text{straw}}$ protocol, we aim to hide the public key pk_0 of the server from V while still allowing V to verify the signature of Y_S using pk_0. This is achieved with secure two-party computation. This helps to protect the server's unique public key from being revealed to V.

The privacy-preserving attestation phase of the $\Pi^{\text{P3P-HS}}_{\text{straw}}$ protocol, which includes steps 4-6, allows for the verification of the certificate and the signature of Y_S without compromising the privacy of S. The remaining steps of the protocol are identical to the DECO protocol.

Security issue. The strawman $\Pi^{\text{P3P-HS}}_{\text{straw}}$ protocol addresses the two privacy issues of the DECO protocol by separately handling the verification of the certificate and the signature of Y_S. However, this separation leaves the protocol vulnerable to attacks by a malicious prover P^f. Specifically, P^f can forge TLS communication with

$\Pi^{\text{P3P-HS}}_{\text{straw}}$ protocol

Initialization:

(1) *Same as DECO protocol.* P samples nounce r_c and sends ClientHello(r_c) to S to start a standard TLS handshake.

Step 1: key exchange:

(2) The prover P receives the server's nounce r_S, the server's ephemeral DH public key Y_S, its signature σ and its certificate cert.

(3) P verifies that cert is a valid certificate and that σ is a valid signature under Y_S and pk_0, the public key contained in c_0.

(4) P sends to V the following tuple:

$$(r_c, r_S, \text{cert}[1:], \text{DS}, \text{h}_\text{m}, Y_S, \sigma),$$

where $\text{cert}[1:]) = (\text{c}_1, \ldots, \text{c}_n)$, DS is the digital signature in the server's certificate c_0 and $\text{h}_\text{m} = \text{SHA256}(\text{m})$, for m to be the message of c_0.

(5) V checks the validity of the chain $\text{cert}[1:]$ and the digital signature DS against pk_1 and h_m.

(6) To check the validity of the signed ephemeral DH public key Y_S, V and P undergo a two-party secure protocol that computes $\text{Verify}(Y_S, \sigma, \text{pk}_0)$ under secret input pk_0.

Step 2: key derivation:

(7) *Same as DECO protocol.*

P **output:** k^{Enc}, k_P^{MAC}.
V **output:** k_V^{MAC}.
S **output:** k^{Enc}.

Figure 2: The strawman protocol.

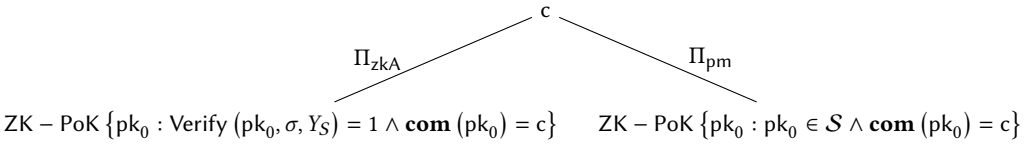

Figure 3: Flow of the attestation phase of $k-$PECO protocol.

the server S by generating a private key s_S^f, forging a different ephemeral DH public key

$$Y_S^f = s_S^f \cdot G,$$

and signing it with a random private key to generate σ^f. The malicious prover can then use the corresponding public key as his private input in the secure two-party computation and convince V that σ^f is a valid signature of Y_S^f. With knowledge of s_S^f, the prover can then find the MAC key, $\mathsf{k}^{\mathrm{MAC}}$, and include the encryption of arbitrary messages in the transcript, proving zero-knowledge statements about the transcript.

The security issue of the strawman protocol stems from the fact that V does not check that the public key of the valid certificate verifies the signature σ of Y_S. As a result, P^f can exploit this weakness to corrupt the protocol and forge TLS communication.

3.3 $k-$PECO

As we have discussed, the strawman $\Pi_{\mathrm{straw}}^{\mathrm{P3P-HS}}$ protocol is vulnerable to attacks by a malicious prover P^f, who can forge TLS communication with the server S by finding the MAC key. This weakness occurs because the verifier V does not check that the public key of the valid certificate verifies the signature σ of Y_S. To address this issue, we need to ensure that the prover P is not able to sign the ephemeral DH public key Y_S sent to the verifier V. In other words, V must confirm that the public key of the valid certificate is the one that verifies the signature σ of Y_S. This helps to prevent P from finding the MAC key, $\mathsf{k}^{\mathrm{MAC}}$, before the proof generation phase of the DECO protocol.

$k-$PECO protocol. In this section, we introduce a new protocol called $k-$anonymous private DECO ($k-$PECO), which is based on a private three-party handshake (P3P-HS) protocol referred to as the $\Pi_{k-\mathrm{PECO}}^{\mathrm{P3P-HS}}$ protocol. The details of $\Pi_{k-\mathrm{PECO}}^{\mathrm{P3P-HS}}$ are presented in Figure 4. $k-$PECO is achieved by replacing the DECO 3P-HS phase by the P3P-HS phase presented. The other two phases are the same as in DECO.

The private attestation mechanism of the $\Pi_{k-\mathrm{PECO}}^{\mathrm{P3P-HS}}$ protocol prevents a malicious prover from exploiting the weakness discussed previously. It requires the prover P to prove to the verifier V that the signature of Y_S is verified by a public key belonging to a set of accepted public keys. This enables P to conceal the identity of S within a set of k accepted servers, $\mathcal{S}$, while also ensuring that P has not forged Y_S. As a result, the privacy and security of the protocol is enhanced.

The private attestation mechanism of the $\Pi_{k-\mathrm{PECO}}^{\mathrm{P3P-HS}}$ protocol is composed of two building blocks: the zkAttest protocol [10] and a proof of membership [13]. As previously mentioned, the

$\Pi_{k-\mathrm{PECO}}^{\mathrm{P3P-HS}}$ protocol

Initialization:

(1) *Same as DECO protocol.* P samples nounce r_c and sends ClientHello(r_c) to S to start a standard TLS handshake.

Step 1: key exchange:

(2) The prover P receives the server's nounce r_S, the server's ephemeral DH public key Y_S, its signature σ and its certificate cert.
(3) P verifies that cert is a valid certificate and that σ is a valid signature signed by pk_0, the public key contained in c_0.
(4) P sends to V the following tuple:
$$(r_c, r_S, Y_S, \sigma).$$
(5) P commits to pk_0, $\mathsf{c} = \mathbf{com}\left(\mathsf{pk}_0\right)$.
(6) V and P run the Π_{zkA} protocol on σ and Y_S, with c. This proves that under the commitment c, the public key pk_0 verifies the signature σ of Y_S.
(7) V and P run the Π_{pm} protocol on c and public $\mathcal{S}$. This proves that under the commitment c, pk_0 is in $\mathcal{S}$, without disclosing pk_0 to V.

Step 2: key derivation:

(8) *Same as DECO protocol.*

P **output:** $\mathsf{k}^{\mathrm{Enc}}$, $\mathsf{k}_P^{\mathrm{MAC}}$.
V **output:** $\mathsf{k}_V^{\mathrm{MAC}}$.
S **output:** $\mathsf{k}^{\mathrm{Enc}}$.

Figure 4: The $k-$PECO protocol.

zkAttest protocol is a $\sum-$protocol that allows a verifier to verify an ECDSA signature on a message using a committed public key, without revealing the actual public key. We refer to this protocol as ΠzkA and, formally, we have that it proves in zero-knowledge the following statement:

$$\mathsf{ZK-PoK}\left\{\mathsf{pk}_0 : \mathsf{Verify}\left(\mathsf{pk}_0, \sigma, Y_S\right) = 1 \wedge \mathbf{com}\left(\mathsf{pk}_0\right) = \mathsf{c}\right\}.$$

The proof of membership protocol, on the other hand, allows a prover to prove in zero-knowledge that a committed public key belongs to a given set of public keys, $\mathcal{S}$. We refer to this protocol as Π_{pm} and, formally, we have it proves in zero-knowledge the

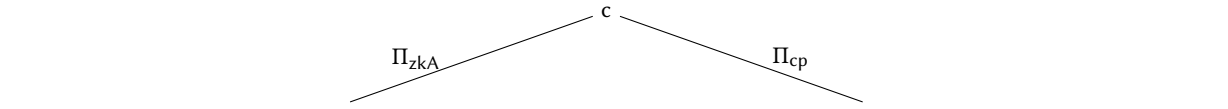

Figure 5: Flow of the attestation phase of PECO protocol. m is the message present in the certificate c_0.

following statement:

$$\text{ZK} - \text{PoK}\left\{\text{pk}_0 : \text{pk}_0 \in \mathcal{S} \wedge \textbf{com}\left(\text{pk}_0\right) = \text{c}\right\}.$$

When these two protocols are composed on the same public key commitment, the public key used to verify the signature of Y_S can be confirmed to belong to the set of accepted keys. This prevents the prover P from forging a signature σ^f for a different DH key Y_S^f. The flow of the attestation phase of the $\Pi_{k-\text{PECO}}^{\text{P3P}-\text{HS}}$ protocol is shown in Figure 3.

Security. The k−PECO protocol presented above addresses the security flaw present in the strawman protocol. Specifically, the prover P is unable to forge a signature σ^f for a generated ephemeral DH public key Y_S^f. Steps 5 and 6 of the $\Pi_{k-\text{PECO}}^{\text{P3P}-\text{HS}}$ protocol ensure that the public key used to sign Y_S belongs to a set of accepted public keys, thereby ensuring that P does not have access to the corresponding secret key. As a result, the security of the protocol is improved.

3.4 PECO

In the previous section, we presented the k−anonymous private DECO protocol, $\Pi_{k-\text{PECO}}^{\text{P3P}-\text{HS}}$, which allows the prover P to hide the identity of the server S from the verifier V with k−anonymity. While k−PECO provides a level of privacy by disclosing only a set of possible identities for S, it also ensures that the data being proven comes from a set of certified (or allowed) servers.

On the other hand, in this section we will present a protocol that fully hides the identity of S from V, while still allowing P to prove statements about TLS data. Each protocol has its own trade-offs and each may be more suitable for different use-cases.

PECO protocol. We introduce a new protocol called private DECO (PECO), which is designed to fully conceal the identity of the server S from the verifier V. PECO is based on a private three-party handshake (P3P-HS) protocol, referred to as the $\Pi_{\text{PECO}}^{\text{P3P}-\text{HS}}$ protocol, as shown in Figure 6. To implement PECO, the 3P-HS phase of the DECO protocol is replaced with the P3P-HS protocol presented in this section, while the other two phases of DECO remain unchanged. This allows PECO to provide stronger privacy protection than the k−PECO protocol, as it does not disclose any possible identities for S. Recall that, in this context, a certificate is a tuple consisting of a message and a signature. The server's certificate is represented by $\text{c}_0 = (\text{m}, \text{DS})$.

The private attestation mechanism of the $\Pi_{\text{PECO}}^{\text{P3P}-\text{HS}}$ protocol relies on the interoperability of different zero-knowledge proofs with committed values (also known as commit-and-open zero-knowledge proofs or CP-ZKP [4]). This phase aims to verify that

$\Pi_{\text{PECO}}^{\text{P3P}-\text{HS}}$ protocol

Initialization:

(1) *Same as DECO protocol.* P samples nounce r_c and sends ClientHello(r_c) to S to start a standard TLS handshake.

Step 1: key exchange:

(2) The prover P receives the server's nounce r_S, the server's ephemeral DH public key Y_S, its signature σ and its certificate cert.
(3) P verifies that cert is a valid certificate and that σ is a valid signature signed by pk_0, the public key contained in c_0.
(4) P sends to V the following tuple:
$$(r_c, r_S, \text{cert}[1:], \text{DS}, \text{h}_\text{m}, Y_S, \sigma),$$
where $\text{cert}[1:]) = (\text{c}_1, \ldots, \text{c}_n)$, DS is the digital signature in the server's certificate c_0 and $\text{h}_\text{m} = \text{SHA256}(\text{m})$, for m to be the message of c_0.
(5) P commits to pk_0, $\text{c} = \textbf{com}\left(\text{pk}_0\right)$.
(6) V checks the validity of the chain cert[1 :].
(7) V and P run the Π_{zkA} protocol on σ and Y_S, with a (Pedersen) commitment of the key pk_0, c. This proves that under the commitment, the public key pk_0 verifies the signature σ of Y_S.
(8) V and P run the Π_{cp} protocol on c, pk_1, DS and h_m. This proves that under the commitment, pk_0 is a substring of the message m that verifies against DS and public key pk_1, without disclosing pk_0 to V.

Step 2: key derivation:

(8) *Same as DECO protocol.*

P **output:** k^{Enc}, k_P^{MAC}.
V **output:** k_V^{MAC}.
S **output:** k^{Enc}.

Figure 6: The PECO protocol.

the committed public key pk_0 is included in the message m of the server's certificate c_0 *and* that it verifies the signature σ of Y_S. These two checks are referred to as the certificate proof protocol (Π_{cp}) and the zero-knowledge attest protocol (Π_{zkAttest} [10]), respectively. The Π_{cp} protocol is presented below and the ΠzkAttest protocol is

Π_{zkA} (s)	Π_{pm} (s)	DECO 3P-HS WAN Online (s)	Total k−PECO 3P-HS estimation (s)	TLS timeout (s)
0.5	10	2.9	~ 13.4	$\sim 10-15$

Table 1: Total estimation time for k−PECO 3P-HS protocol and comparison with standard TLS timeout.

Π_{zkA} (s)	Π_{cp} (s)	DECO 3P-HS WAN Online (s)	Total PECO 3P-HS estimation (s)	TLS timeout (s)
0.5	14.5	2.9	~ 18	$\sim 10-15$

Table 2: Total estimation time for PECO 3P-HS protocol and comparison with standard TLS timeout. The estimation time for Π_{cp} considers the LegoAC1 protocol for the SHA256 implementation in the LegoSNARK framework [8]. We only included the proving and verification time as the key generation step is independent of the inputs.

the same as in the k−PECO protocol. The flow of the attestation phase of the PECO protocol is shown in Figure 5.

Certificate proof protocol. The Π_{cp} protocol is realized with a CP-ZKP system (LegoSNARK [8] or LegoGroth16 [16]). Since both LegoSNARK or LegoGroth16 use Pedersen commitments, it is feasible to compose any of them with the $\Pi_{zkAttest}$ protocol [10] as it also uses Pedersen commitments for the public key. Formally, the Π_{cp} protocol proves in zero-knowledge the following statement:

$$ZK-PoK\left\{pk_0, m : pk_0 \text{ is a substring of } m \wedge \mathbf{com}\left(pk_0\right) = c\right\}.$$

The Π_{cp} protocol goes as follows. The verifier uses the public key pk_1 of the certificate c_1 to open the signature in c_0, DS, i.e. V computes $h = Dec_{pk_0}(DS)$. Then, P proves in zero-knowledge to V that $h = SHA256(m)$ where pk_0 from $\mathbf{com}\left(pk_0\right)$ is a substring of m.

Security. Similarly to the k−PECO protocol presented above, the PECO 3P-HS avoids the security flaw of the strawman protocol. Again, the prover P is not able to forge a signature σ^f of a generated ephemeral DH public key Y_S^f. Steps 6 and 7 ensure that the public key used to sign Y_S is the same public key in the server's certificate c_0, guaranteeing that P does not know its corresponding secret key.

4 IMPLEMENTATION

In this section, we present the implementation details and tools used for both the k−PECO and PECO protocols. Both protocols are based on the DECO protocol but feature a privacy-preserving version of the 3P-HS protocol called the P3P-HS protocol. We will compare the P3P-HS approaches used in both protocols to the standard 3P-HS protocol proposed by DECO. For reference, in the WAN setting, the online time for the 3P-HS protocol using TLS 1.2 is approximately 2.9 seconds.

4.1 k−PECO

The k−PECO protocol consists of the DECO protocol with a privacy-preserving attestation mechanism in the 3P-HS protocol, referred to as the Π^{P3P-HS}_{k-PECO} protocol. This protocol is built using the proof of membership and zkAttest protocols, which are implemented in TypeScript. The code for these protocols can be accessed at this repository [10].

The performance of the zkAttest protocol is as follows: the time for verification is 0.5 seconds, and it takes approximately 10 seconds to prove membership on a list of several thousand with no significant slowdown. These values, along with the time for the DECO 3P-HS phase (see [22], Table 1), can be used to estimate the total time for the k−PECO 3P-HS protocol. It is worth noting that this state-of-the-art implementation may cause TLS handshake errors with the server S if it exceeds the standard TLS timeout. These details are summarized in Table 1.

4.2 PECO

Similarly to k−PECO, the PECO protocol consists of the DECO protocol with a privacy-preserving attestation mechanism in the 3P-HS protocol, referred to as the Π^{P3P-HS}_{PECO} protocol. This protocol is implemented using the certificate proof (Π_{cp}) and zkAttest protocols. The zkAttest protocol is implemented in the zkp-ecdsa repository [10]. The certificate proof protocol can be implemented using either the LegoSNARK implementation available at this repository [8], or the LegoGroth16 (a LegoSNARK version of the Groth16 protocol) available at this repository [16]. A fork of the latter repository with Circom support for LegoGroth16 is also available at this repository [9].

The LegoSNARK protocol reports a proving time of 0.9 seconds and a verification time of 1.8 milliseconds for SHA256 with 512-bit inputs. Assuming that an X.509 certificate has a size of approximately 1 Kbyte, it is necessary to run SHA256 with 512-bit inputs approximately 16 times to prove that the committed key pk_0 is part of the pre-image of a given digest. This results in a total time of approximately 14.5 seconds to generate and verify a single proof. These details are summarized in Table 2. It is worth noting that this state-of-the-art implementation is slightly above the limit of the standard TLS timeout, which may cause TLS handshake errors with the server S. Therefore, some optimization may be necessary to avoid such errors.

5 CONCLUSION

The DECentralized Database (DECO) protocol is a way to verify the provenance of data from Transport Layer Security (TLS) connections in zero-knowledge. We have developed an extension to DECO called PECO, which includes two new private three-party handshake protocols (P3P-HS) to enhance privacy features. PECO allows web users to prove to a verifier properties of data from TLS connections without disclosing the identity of the servers. It offers two options: one that provides $k-$anonymity for the server's identity (k-PECO), and another that completely masks the server's identity from the verifier (PECO). PECO is based on three main protocols: commit-and-proof zero-knowledge proofs (CP-ZKP), verification of Elliptic Curve Digital Signature Algorithm (ECDSA) signatures under a committed public key without revealing the key (zkAttest), and a proof of membership to verify that a committed key belongs to a set of keys. The main contributions of this paper are the design of these protocols and an evaluation of their performance using state-of-the-art implementation tools.

The authors suggest an implementation of both the k-PECO and PECO protocols using the DECO protocol and the corresponding P3P-HS protocols. The k-PECO protocol uses the proof of membership and zkAttest protocols, while the PECO protocol uses the certificate proof and zkAttest protocols. The zkAttest protocol is implemented in the zkp-ecdsa [10] repository, and the certificate proof protocol can be implemented using either the LegoSNARK or LegoGroth16 protocol. The k-PECO protocol has an estimated online time of approximately 13.4 seconds, while the PECO protocol has an estimated online time of slightly above the standard TLS timeout of $10 - 15$ seconds. These times may cause TLS handshake errors with the server, so some optimization may be necessary to avoid such errors.

ACKNOWLEDGMENTS

This work was funded by Fundação para a Ciência e a Tecnologia (FCT) through National Funds under Award SFRH/BD/144806/2019, Award UIDB/50008/2020, and Award UIDP/50008/2020; in part by the Regional Operational Program of Lisbon; by FCT, I.P.

REFERENCES

[1] Stephanie Bayer and Jens Groth. 2013. Zero-knowledge argument for polynomial evaluation with application to blacklists. In *Advances in Cryptology – EUROCRYPT 2013*. Springer Berlin Heidelberg, 646–663. DOI: 10.1007/978-3-642-38348-9_38.

[2] D. Beaver, S. Micali, and P. Rogaway. 1990. The round complexity of secure protocols. In *Proceedings of the Twenty-Second Annual ACM Symposium on Theory of Computing* (STOC '90). Association for Computing Machinery, Baltimore, Maryland, USA, 503–513. ISBN: 0897913612. DOI: 10.1145/100216.100287.

[3] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. 1988. Completeness theorems for non-cryptographic fault-tolerant distributed computation. In *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing* (STOC '88). Association for Computing Machinery, Chicago, Illinois, USA, 1–10. ISBN: 0897912640. DOI: 10.1145/62212.62213.

[4] Daniel Benarroch, Matteo Campanelli, Dario Fiore, Jihye Kim, Jiwon Lee, Hyunok Oh, and Anaïs Querol. 2021. Proposal: commit-and-prove zero-knowledge proof systems and extensions. In.

[5] Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. 2012. From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again. In *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference* (ITCS '12). Association for Computing Machinery, Cambridge, Massachusetts, 326–349. ISBN: 9781450311151. DOI: 10.1145/2090236.2090263.

[6] Manuel Blum, Paul Feldman, and Silvio Micali. 1988. Non-interactive zero-knowledge and its applications. In *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing* (STOC '88). Association for Computing Machinery, Chicago, Illinois, USA, 103–112. ISBN: 0897912640. DOI: 10.1145/62212.62222.

[7] Jan Camenisch and Markus Stadler. 1997. Efficient group signature schemes for large groups. In *Advances in Cryptology — CRYPTO '97*. Springer Berlin Heidelberg, 410–424. DOI: 10.1007/bfb0052252.

[8] Matteo Campanelli, Dario Fiore, and Anaïs Querol. 2019. LegoSNARK. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. ACM, (Nov. 2019). DOI: 10.1145/3319535.3339820.

[9] Dock. 2022. Circom language integration: anonymous credentials protocol update. Retrieved Dec. 26, 2022 from https://blog.dock.io/circom-language-integration/.

[10] Armando Faz-Hernández, Watson Ladd, and Deepak Maram. 2021. Zkattest: ring and group signatures for existing ecdsa keys. In *Selected Areas in Cryptography*. Riham AlTawy and Andreas Hülsing, (Eds.) Available at https://github.com/cloudflare/zkp-ecdsa. v0.2.5 Accessed Nov 2022. Springer International Publishing, Cham, (Oct. 2021), 68–83. ISBN: 978-3-030-99277-4. DOI: 10.1007/978-3-030-99277-4_4.

[11] O. Goldreich, S. Micali, and A. Wigderson. 1987. How to play ANY mental game. In *Proceedings of the nineteenth annual ACM conference on Theory of computing - STOC '87*. ACM Press. DOI: 10.1145/28395.28420.

[12] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. 1989. The knowledge complexity of interactive proof systems. *SIAM Journal on Computing*, 18, 1, (Feb. 1989), 186–208. DOI: 10.1137/0218012.

[13] Jens Groth and Markulf Kohlweiss. 2015. One-out-of-many proofs: or how to leak a secret and spend a coin. In *Advances in Cryptology - EUROCRYPT 2015*. Springer Berlin Heidelberg, 253–280. DOI: 10.1007/978-3-662-46803-6_9.

[14] Michael Kan. 2020. Cloudflare dumps google's reCAPTCHA over privacy concerns, costs. Retrieved Dec. 21, 2022 from https://www.pcmag.com/news/cloudflare-dumps-googles-recaptcha-over-privacy-concerns-costs.

[15] Vladimir Kolesnikov. 2005. Gate evaluation secret sharing and secure one-round two-party computation. In *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 136–155. DOI: 10.1007/11593447_8.

[16] Pratyush Mishra, Kobi Gurkan, and Pascal Berrang. 2021. Legogro16. https://github.com/kobigurk/legogro16. (2021).

[17] Moni Naor, Benny Pinkas, and Reuban Sumner. 1999. Privacy preserving auctions and mechanism design. In *Proceedings of the 1st ACM Conference on Electronic Commerce* (EC '99). Association for Computing Machinery, Denver, Colorado, USA, 129–139. ISBN: 1581131763. DOI: 10.1145/336992.337028.

[18] Torben Pryds Pedersen. [n. d.] Non-interactive and information-theoretic secure verifiable secret sharing. In *Advances in Cryptology — CRYPTO '91*. Springer Berlin Heidelberg, 129–140. DOI: 10.1007/3-540-46766-1_9.

[19] Katharine Schwab. 2019. Google's new reCAPTCHA has a dark side. Retrieved Dec. 21, 2022 from https://www.fastcompany.com/90369697/googles-new-recaptcha-has-a-dark-side.

[20] Andrew C. Yao. 1982. Protocols for secure computations. In *Proceedings of the 23rd Annual Symposium on Foundations of Computer Science* (SFCS '82). IEEE Computer Society, USA, 160–164.

[21] Samee Zahur, Mike Rosulek, and David Evans. 2015. Two halves make a whole. In *Advances in Cryptology - EUROCRYPT 2015*. Springer Berlin Heidelberg, 220–250. DOI: 10.1007/978-3-662-46803-6_8.

[22] Fan Zhang, Deepak Maram, Harjasleen Malvai, Steven Goldfeder, and Ari Juels. 2020. DECO: liberating web data using decentralized oracles for TLS. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*. ACM, (Oct. 2020). DOI: 10.1145/3372297.3417239.