DiStefano: Decentralized Infrastructure for Sharing Trusted Encrypted Facts and Nothing More

Private and Efficient Commitments for TLS-encrypted Data

Sofía Celi*, Alex Davidson†, Hamed Haddadi*¶, Gonçalo Pestana‡ and Joe Rowell§
* Brave Software, cherenkov@riseup.net, hamed@brave.com
† NOVA LINCS & Universidade NOVA de Lisboa, a.davidson@fct.unl.pt
‡ Hashmatter, gpestana@hashmatter.com
§Information Security Group, Royal Holloway, University of London, joe.rowell@rhul.ac.uk
¶Imperial College London, h.haddadi@imperial.ac.uk

Abstract—**We design DiStefano: an efficient, maliciously-secure framework for generating private commitments over TLS-encrypted web traffic, for verification by a designated third-party. DiStefano provides many improvements over previous TLS commitment systems, including: a modular protocol specific to TLS 1.3, support for arbitrary verifiable claims over encrypted data, client browsing history privacy amongst pre-approved TLS servers, and various optimisations to ensure fast online performance of the TLS 1.3 session. We build a permissive open-source implementation of DiStefano integrated into the BoringSSL cryptographic library (used by Chromium-based Internet browsers). We show that DiStefano is practical in both LAN and WAN settings for committing to facts in arbitrary TLS traffic, requiring $< 1\,\mathrm{s}$ and $\leq 80\,\mathrm{KiB}$ to execute the complete online phase of the protocol.**

I. Introduction

The Transport-Layer Security (TLS) protocol [1] provides encrypted and authenticated channels between clients and servers on the Internet. Such channels commonly transmit *trusted information* about users behind clients such as proofs of age [2], social security statuses [3], and accepted purchase information. While various applications would benefit from learning such data points, doing so represents an obvious privacy concern [4]–[9]. Exporting such information as anonymous credentials is non-trivial since the information resides in an encrypted and authenticated channel. Meanwhile, both legislation (such as GDPR [10]) and standards bodies (such as W3C [11]) have made usage of privacy-preserving data credentials a priority.

Designated-Commitment TLS (DCTLS) protocols (also known as *three-party handshake* protocols) provide modified TLS handshakes that allow exporting certain claims on the TLS channel to a designated *verifier*. The protocols perform handshakes that secret-share private session data amongst a client and a verifier, and compute the handshake and record-layer phases in two-party computation (2PC). Examples of DCTLS protocols include DECO [2], TLSNotary/PageSigner [12][1], TownCrier [13], Garble-then-Prove [14], and Janus [15]. Similar techniques are also used to produce zero-knowledge middleboxes [16] (for proving that client traffic adheres to corporate browsing policies, for example), and for devising multi-party TLS clients/servers. Prominent examples of the latter are Oblivious TLS [17] and MPCAuth [18].

Unfortunately, while previous works claim practicality, all such DCTLS protocols appear insufficient for wide-scale usage. First, no protocol explicitly provides secure support for TLS 1.3. Support for TLS 1.3 surpassed that of 1.2 around December 2020 [19] and, according to Cloudflare Radar [20], TLS 1.3 now accounts for 63% of secure network traffic as opposed to 8.7% for TLS 1.2, so it is imperative that protocols support this version. When DCTLS protocols do support TLS 1.3, the security analysis is lacking and/or efficiency concerns that surround implementing TLS 1.3 ciphersuites and protocol steps in (maliciously-secure) 2PC are overlooked. Existing security arguments also lack in *agility*, meaning that they only apply for a static protocol, ciphersuite and 2PC primitives. This is a critical concern: primitives used by DECO have been already shown to be insecure [21], [22]. Client privacy is also neglected as the protocols reveal the server that clients communicate with to the verifier, revealing their browsing history. Second, from a deployability perspective, no fully-featured open-source implementation of a DCTLS protocol exists that achieves strong security guarantees, or much less one that interoperates with Internet browsing tools.

Our work. We design DiStefano (Fig. 1), a DCTLS protocol that securely generates private commitments over TLS 1.3 data. Security is proven using a novel standalone model that permits cryptographic agility by allowing to swap various schemes depending on the desired ciphersuite. DiStefano is provided as a permissive open-source implementation[2] integrated into the widely-used BoringSSL library,[3] where 2PC functionality is provided by emp [23]. With respect

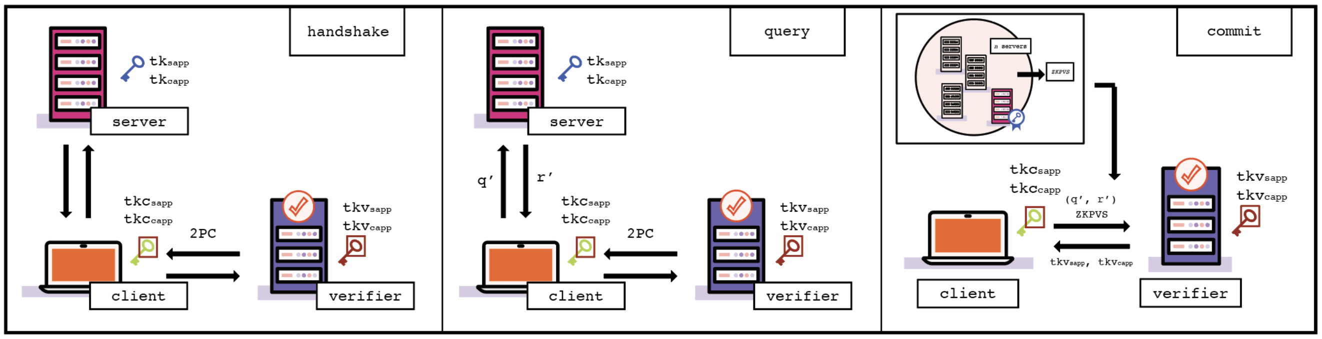
[1]We refer exclusively here to original PageSigner as TLSNotary does not appear to have a fixed cryptographic design.

[2]https://github.com/brave-experiments/DiStefano

[3]This library is used by most Chromium-based Internet browsers, that make up a dominant share of all browser usage.

Network and Distributed System Security (NDSS) Symposium 2025
24-28 February 2025, San Diego, CA, USA
ISBN 979-8-9894372-8-3
https://dx.doi.org/10.14722/ndss.2025.241367
www.ndss-symposium.org

Figure 1. An overview of the DiStefano protocol. In the **handshake** and **query** phases, the client performs the TLS 1.3 handshake and record-layer phases in conjunction with the verifier using 2PC to secret-share traffic keys and other session data for establishing a secure session with the server (secret-shared keys are represented with a square over the key). In the **commitment** phase, the client authenticates the server to the verifier using a zero-knowledge proof of valid TLS signatures (denoted by ZKPVS, see Section A-C), and commits to some encrypted session data, before receiving the verifier's secret TLS session shares.



to the client's privacy, DiStefano supports zero-knowledge authentication amongst N verifier-approved TLS servers by using a zero-knowledge proof of valid signatures. Finally, the commitments generated by DiStefano can be used to produce any type of verifiable private claim, either non-interactively using zero-knowledge proofs or interactively using 2PC. Note that, in this work, we value the building of a modular framework for solving the core commitment functionality, and leave the implementation of the subsequent proving stage up to the implementer, as this can differ significantly depending on the application (see Section IV-D). To ensure high performance, a number of optimisations were made to the cryptographic functionality and software implementation of DiStefano. We show that the online portions of the handshake and record-layer phases can be executed in $500\,\mathrm{ms}$ and $\sim 750\,\mathrm{ms}$ in a LAN setting, and with $5\,\mathrm{KiB}$ and around $4\,\mathrm{KiB}$ of bandwidth, respectively, for $2\,\mathrm{KiB}$ of TLS communication. In a WAN setting, there are modest increases in timing that are largely explained by the increase in latency. All in all, the online costs of DiStefano fall way under a second, which is far below standard TLS handshake timeout times of 10-20 seconds [24].

Formal contributions. Our formal contributions are:

- A private Designated-Commitment TLS 1.3 (DCTLS) protocol, DiStefano (Section IV), with a modular, standalone security framework that guarantees security in the presence of malicious adversaries (Section VI).
- Novel optimisations that allow running secure 2PC TLS 1.3 clients with higher efficiency (Section V).
- An open-source, Chromium-compliant implementation integrated into BoringSSL.
- Experimental analysis showing that DiStefano is practical (in LAN/WAN settings) for committing to various sizes of Internet traffic (Section VII).

II. Background

A. General Notation

Vectors are denoted by lower-case bold letters. We use $\mathsf{len}(s)$ to denote the length of $s \in \{0,1\}^*$. The symbol $[m]$ indicates the set $\{1, 2, \ldots, m\}$. We write $a \leftarrow b$ to assign the value of b to a, and $a \leftarrow\!\!\$\, S$ to assign a uniformly sampled element from the set S. λ denotes the security parameter.

We denote a finite field of characteristic q as $\mathbb{F}_q$ and the m-dimensional vector space over $\mathbb{F}_q$ as $\mathbb{F}_{q^m}$. We are primarily concerned with the smallest field, $\mathbb{F}_2$, where the additive operation on $a, b \in \mathbb{F}_2$ is simply an *exclusive-or* operation, $a \oplus b$, with multiplication corresponding to the AND operation. We extend this notation to refer to operations on m-dimensional vectors $\mathbf{a}, \mathbf{b} \in \mathbb{F}_{2^m}$, writing $\mathbf{a} \oplus \mathbf{b}$ and $\mathbf{a} \cdot \mathbf{b}$ to refer to addition and multiplication, respectively. Note that while addition in $\mathbb{F}_{2^m}$ is simply m XOR operations, multiplication over $\mathbb{F}_{2^m}$ requires extra logic compared to multiplications over $\mathbb{F}_2$. We write elliptic curves with a generator G over $\mathbb{F}_q$ as $EC(\mathbb{F}_q)$.

For a security game Game used by a cryptographic scheme Δ, we denote the advantage of an algorithm $\mathcal{A}$ in Δ by $\mathsf{Adv}^{\mathsf{game}}_{\mathcal{A},\Delta}(\lambda)$, where:

$$\mathsf{Adv}^{\mathsf{game}}_{\mathcal{A},\Delta}(\lambda) = \Pr[\mathcal{A} \text{ succeeds}] - \Pr[\mathcal{A} \text{ fails}]. \qquad (1)$$

We say that Δ is secure with respect to Game, iff $\mathsf{Adv}^{\mathsf{game}}_{\mathcal{A},\Delta}(\lambda) \leq \mathsf{negl}(\lambda)$, for some negligible function $\mathsf{negl}(\lambda)$ and security parameter λ.

B. *Background on* DCTLS *Protocols*

Designated-Commitment (DCTLS) TLS protocols allow a client ($\mathcal{C}$) to generate commitments to TLS session data communicated with a server ($\mathcal{S}$) that can be sent to a designated third-party verifier ($\mathcal{V}$). They consist of the following phases (which are described in Appendix B): a ($\mathcal{V}$-assisted) handshake phase, a ($\mathcal{V}$-assisted) query execution phase, and a commitment phase. Previous work, such as in DECO and tools like PageSigner, provide explicit attestation functionality for proving facts about the committed TLS session (using zero-knowledge proofs). Note that, without such commitments, proving statements that use TLS data as sources of truth must assume either a trustworthy client, or $\mathcal{C}$ must allow $\mathcal{V}$ to read their TLS traffic in the clear.

DCTLS over TLS 1.3. Previous DCTLS protocols focused on TLS 1.2, with an informal (and mostly incomplete) extension to TLS 1.3. Recall that TLS 1.3 emerged in response to dissatisfaction with the outdated design of the TLS 1.2 handshake, its two-round-trip overhead, and the increasing number of practical attacks [25]–[28]. The necessary changes introduced by TLS 1.3 to improve performance and deployability are significant stumbling blocks for applying previous DCTLS protocols directly. Thus, in this work, we focus on TLS 1.3, and highlight explicit changes to DCTLS protocols that are required for handling the substantial protocol-level differences. We provide an overview of the standard TLS 1.3 handshake, and its standard notation defined in [29], in the full version [30].

Description of DCTLS phases. In Fig. 1, we summarise the stages of DCTLS for establishing commitments to TLS 1.3 encrypted traffic between $\mathcal{C}$ and $\mathcal{S}$ to be sent to a designated $\mathcal{V}$. In the following, we describe how the different stages of the protocol function in relation to the various stages of the TLS 1.3 protocol [1]. The following is an informal description of TLS 1.3 (1-RTT with certificate-based authentication) when extended to support DCTLS-like protocols.

Handshake phase. In this phase, $\mathcal{S}$ learns the same secret session parameters as in standard TLS 1.3, while $\mathcal{C}$ and $\mathcal{V}$ learn shares of the session parameters that only $\mathcal{C}$ would normally learn. This requires $\mathcal{C}$ and $\mathcal{V}$ to engage in the core TLS 1.3 protocol using a series of 2PC functionalities.

We focus on the default mode for establishing a secure TLS 1.3 session using (EC)DH ciphersuites, and certificate-based authentication between $\mathcal{C}$ and $\mathcal{S}$. In this mode, the handshake starts with $\mathcal{C}$ sending a `ClientHello` (`CH`) message to $\mathcal{S}$. This message advertises the supported (EC)DH groups and the ephemeral (EC)DH keyshares specified in the `supported_groups` and `key_shares` extensions, respectively. The `CH` message also advertises the signature algorithms supported. It also contains a nonce and a list of supported symmetric-key algorithms (ciphersuites). Note that for DCTLS protocols, the ephemeral keyshares $Z \in EC(\mathbb{F}_c)$ are generated as a combination of additive shares ($z_X \leftarrow\$ \mathbb{F}_c, Z_X = z_X \cdot G$) for $X \in \{\mathcal{C}, \mathcal{V}\}$, where $Z = Z_{\mathcal{C}} + Z_{\mathcal{V}} \in EC(\mathbb{F}_c)$.

$\mathcal{S}$ processes the `CH` message and chooses cryptographic parameters for the session. If (EC)DH key exchange is used, $\mathcal{S}$ sends a `ServerHello` (`SH`) message containing a `key_share` extension with their (EC)DH key, corresponding to one of the `key_shares` advertised by $\mathcal{C}$. The `SH` message also contains a $\mathcal{S}$-generated nonce and the ciphersuite chosen. An ephemeral shared secret is computed at both ends, which requires $\mathcal{C}$ and $\mathcal{V}$ to engage in 2PC computation. Afterwards, all subsequent handshake messages are encrypted using keys derived from this secret. Once this derivation is performed, $\mathcal{V}$'s keys can be revealed to $\mathcal{C}$ to perform local encryption/decryption of handshake messages, as these keys are considered independent from the eventual session secret derived at the end of the handshake [29].

The server $\mathcal{S}$ then sends a certificate chain (in the `ServerCertificate` message -`SCRT`-), and a message that contains a proof that they possess the private key corresponding to the public key advertised in the leaf certificate. This proof is a signature over the handshake transcript and it is sent in the `ServerCertificateVerify` (`SCV`) message. $\mathcal{S}$ also sends the `ServerFinished` (`SF`) message that provides integrity of the handshake up to this point. It contains a message authentication code (MAC) over the entire transcript, providing key confirmation and binding $\mathcal{S}$'s identity to any computed keys. Optionally, $\mathcal{S}$ can send a `CertificateRequest` (`CR`) message, prior the `SCRT` message, requesting a certificate from $\mathcal{C}$.

At this point, $\mathcal{S}$ can immediately send application data to the unauthenticated $\mathcal{C}$. Upon receiving $\mathcal{S}$'s messages, $\mathcal{C}$ verifies the signature of the `SCV` message and the MAC of `SF`. If requested, $\mathcal{C}$ responds with their own authentication messages, `ClientCertificate` and `ClientCertificateVerify`, to achieve mutual authentication. Finally, $\mathcal{C}$ must confirm their view of the handshake by sending a MAC over the handshake transcript in the `ClientFinished` (`CF`) message. The MAC generation must also be computed in 2PC.

Now, the handshake is completed, and $\mathcal{C}$ and $\mathcal{S}$ can derive the key material required by the subsequent *record layer* to exchange authenticated and encrypted application data. This derivation is performed in 2PC, and $\mathcal{C}$ and $\mathcal{V}$ both hold shares of all the secret parameters needed to encrypt traffic using the specified encryption ciphersuite. In this work, we specifically target AES-GCM, since over 90% of TLS 1.3 traffic uses this ciphersuite [31].

Record Layer (Query Execution) phase. $\mathcal{C}$ sends a query q (in encrypted form $\hat{\mathsf{q}}$) to $\mathcal{S}$ with help from $\mathcal{V}$. Specifically, since the session keys are secret-shared, $\mathcal{C}$ and $\mathcal{V}$ jointly compute the encryptions of these queries in 2PC. Encrypted responses, $\hat{\mathsf{r}}$, can then be decrypted using a similar procedure to reveal $\mathcal{S}$'s response r to $\mathcal{C}$. This is important for running tools in a browser, or any multi-round protocol, where subsequent queries depend on previous responses.

Commitment phase. After querying $\mathcal{S}$ and receiving a response r, $\mathcal{C}$ commits to the session by forwarding the ciphertexts to $\mathcal{V}$, and receives $\mathcal{V}$'s session key shares in exchange. Hence, $\mathcal{C}$ can verify the integrity of r, and later prove statements about it. The fact that $\mathcal{C}$ sends commitments before they receive $\mathcal{V}$'s shares means that $\mathcal{V}$ can trust subsequent attestations over the commitments.

Limitations of approaches. Existing DCTLS schemes have serious security, performance, and deployability limitations. They either only work with old/deprecated TLS versions (1.2 and under) and offer no privacy from the oracle (PageSigner [32]), or rely on trusted hardware (Town Crier [13]) against which various attacks exist [33]. Another class of oracle schemes assumes cooperation from $\mathcal{S}$ by installing TLS extensions [34], or by changing application-layer logic [35]. These approaches suffer from two fundamental problems: they break legacy compatibility, causing a significant barrier to wide adoption; and only provide conditional exportability as $\mathcal{S}$ has the sole discretion to determine which data can be exported, and can censor export attempts. While DECO [2] promises to solve these problems, its non-modular security design, combining all of the individual phases of the protocol, makes it impossible to swap individual pieces of functionality (without rewriting the entire security proof). These limitations have the following repercussions.

Security. Some primitives used by DECO have since been shown to be insecure [21], [22], and the security proof only targets TLS 1.2. General guidance is offered for handling TLS 1.3, but is not formally specified. More worryingly, the security argument is all-encompassing, proving security for a combined session involving the TLS handshake, subsequent record protocol, and the zero-knowledge attestation layer (this is common in other constructions too [13], [18]). This significantly harms *cryptographic agility*, since *any* change to the utilised primitives, protocol, or ciphersuites theoretically dictates that an entirely new proof be written. Lack of cryptographic agility is a significant source of cryptographic vulnerabilities in real-world systems [36].

Privacy. Explicit authentication of $\mathcal{S}$ to $\mathcal{V}$ during the handshake is mandated due to the non-modular security proof, which is harmful for $\mathcal{C}$'s browsing privacy. This is somewhat necessitated by TLS 1.2's handshake flow, where the identity of $\mathcal{S}$ is sent in the clear in the first message they sent. By contrast, TLS 1.3 rearranges the handshake flow slightly, and the certificate is only exchanged in an encrypted form on later messages. Because of this, DiStefano achieves modular privacy properties and retains the privacy of $\mathcal{C}$ (we never disclose the identity $\mathcal{S}$ to $\mathcal{V}$, as this will reveal the client's browsing history, for example).

Performance. Certain underlying cryptographic tools (such as oblivious transfer protocols) have seen remarkable improvements following DECO's publication [37], [38]. However, certain parts of the transformations needed to handle the AES-GCM ciphersuite detailed by DECO are underspecified, and naively lead to high costs during 2PC execution.

Deployability. Recent DCTLS protocols [14], [15], [39], [40] are either aimed entirely at TLS 1.2 [14], are entirely theoretical [39], or use semi-honest 2PC to achieve reasonable performance [15], [40]. We note that the use of semi-honest 2PC must be applied carefully to guarantee overall malicious security of the protocol, and (to the best of our knowledge) no TLS 1.3 attestation mechanism has yet been proposed that completely satisfies this. In fact, we discuss in the full version [30] that using semi-honest 2PC primitives may lead to potential attacks. Moreover, even when semi-honest 2PC is used, performance is lacking and public implementations are rare. One recent example is the recently proposed Janus protocol [15], which is accompanied by a reference implementation, but without integration into existing TLS libraries. Janus achieves a reported handshake time of around $0.51\,\mathrm{s}$ in a LAN setting with around $113\,\mathrm{KiB}$ of handshake traffic. By contrast, DiStefano achieves malicious security guarantees, with around $0.1\,\mathrm{ms}$ of online time, and exchanging much less data: around $28\,\mathrm{KiB}$ for the online phase of the handshake. Thus, we conclude that using malicious 2PC is not a bottleneck for current protocols.

C. Overview of DiStefano

Due to the limitations of the previous DCTLS protocols, we aim to build a protocol that works for TLS 1.3, improves privacy guarantees for $\mathcal{C}$, does not require specific hardware or extensions, and can be easily integrated into common applications. Overall, DiStefano achieves the following.

- The creation of a maliciously-secure framework that generates binding and hiding commitments over data communicated during TLS 1.3 sessions.
- Cryptographic optimisations that ensure practical running costs, and experimental analysis showing that DiStefano is ready for real workflows.
- A publicly-available implementation integrated into the TLS library that browsers use, with no need for specialized hardware or installing extra extensions.

We believe that DiStefano is an essential step-forward for showing that DCTLS *can* be implemented in practice.

Overview of required optimisations. Our implementation of DiStefano requires several optimisations to achieve its performance. We reduce the number of rounds required to derive AES-GCM secret shares (cf. Section V) by a factor of around 500 compared to prior art (PageSigner), and reduce the required bandwidth by around a factor of 3. Moreover, we carefully combine multiple sub-circuits used in the TLS handshake to reduce the number of re-computed secrets and circuit invocations (cf. Section IV-A). We emphasise that a significant portion of our engineering effort was dedicated to fine-tuning at a low level, and we view this as a valuable contribution in its own regard: we aspire that this facilitates seamless adaptation of our code by future researchers.

III. Secure Multi-Party Computation

Two-party secure computation (2PC) protocols allow parties p_1 and p_2 to jointly compute generic functions $f(s_1, s_2)$ over their private inputs s_1 and s_2. The security of the protocols ensures that nothing of each input is revealed to the other party, except for what f naturally reveals [41]. There are two common approaches for 2PC protocols. *Garbled circuits protocols* [42], [43] encode f as a boolean circuit and evaluate an encrypted variant of the circuit across two parties. *Threshold secret-sharing* protocols (e.g. SPDZ [44], [45], or MASCOT [46]), typically operate by first producing some random multiplicative triples (referred to as *Beaver triples* [47]) before additively sharing secret inputs with some extra information. Garbled circuit protocols are particularly well-suited to secure evaluation of binary circuits, such as AES or SHA-256. The cost of a garbled circuit is normally evaluated in terms of the number of AND gates due to the *Free-XOR* optimisation [48]. In contrast, threshold secret-sharing schemes are typically well-suited for computing arithmetic operations,

Figure 2. The DiStefano 1-RTT handshake protocol. Shorthands correspond to those defined in [29]. Purple represents messages sent or calculated by $\mathcal{V}$, orange by the client, pink by the server, and black for 2PC calculations between the client and verifier. Messages with an asterisk (*) are optional, and those within braces ({}) are encrypted.

Verifier	**Client**	**Server**
		static (Sig): $\mathsf{pk}_{\mathcal{S}}, \mathsf{sk}_{\mathcal{S}}$
`ClientHello:`	`ClientHello:`	`ServerHello:`
$z_v \leftarrow_\$ \mathbb{Z}_q,\ Z_v \leftarrow g^{z_v}$	$x_c \leftarrow_\$ \mathbb{Z}_q,\ X_c \leftarrow g^{x_c}$	$y_s \leftarrow_\$ \mathbb{Z}_q$
`+ClientKeyShare:` $\mathrm{SSK} \leftarrow Z_v + X_c$	`+ClientKeyShare:` $\mathrm{SSK} \leftarrow Z_v + X_c$	`+ServerKeyShare:` $Y_s \leftarrow_\$ g^{y_s}$
	Forward SKS to Verifier	
$ssk_v \leftarrow Y_s^{z_v},\ t_v \leftarrow \mathsf{ECtF}(ssk_v)$	$ssk_c \leftarrow Y_s^{x_c},\ t_c \leftarrow \mathsf{ECtF}(ssk_c)$	
		$\mathrm{DHE} \leftarrow \mathrm{SSK}^{y_s}$
$\mathrm{HS}^v \oplus \mathrm{HS}^c \leftarrow \mathsf{HKDF.Extract}(\varnothing, t_v + t_c)$		$\mathrm{HS} \leftarrow \mathsf{HKDF.Extract}(\varnothing, \mathrm{DHE})$
$\mathrm{CHTS}^v \oplus \mathrm{CHTS}^c \leftarrow \mathsf{HKDF.Expand}(\mathrm{HS}^v \oplus \mathrm{HS}^c, \mathrm{Label}_1 \parallel \mathrm{H}_0)$		$\mathrm{CHTS} \leftarrow \mathsf{HKDF.Expand}(\mathrm{HS}, \mathrm{Label}_1 \parallel \mathrm{H}_0)$
$\mathrm{SHTS}^v \oplus \mathrm{SHTS}^c \leftarrow \mathsf{HKDF.Expand}(\mathrm{HS}^v \oplus \mathrm{HS}^c, \mathrm{Label}_2 \parallel \mathrm{H}_0)$		$\mathrm{SHTS} \leftarrow \mathsf{HKDF.Expand}(\mathrm{HS}, \mathrm{Label}_2 \parallel \mathrm{H}_0)$
$\mathrm{dHS}^v \oplus \mathrm{dHS}^c \leftarrow \mathsf{HKDF.Expand}(\mathrm{HS}^v \oplus \mathrm{HS}^c, \mathrm{Label}_3 \parallel \mathrm{H}_1)$		$\mathrm{dHS} \leftarrow \mathsf{HKDF.Expand}(\mathrm{HS}, \mathrm{Label}_3 \parallel \mathrm{H}_1)$
$\mathsf{tk}^v_{chs} \oplus \mathsf{tk}^c_{chs} \leftarrow \mathsf{DeriveTK}(\mathrm{CHTS}^v \oplus \mathrm{CHTS}^c)$		$\mathsf{tk}_{chs} \leftarrow \mathsf{DeriveTK}(\mathrm{CHTS})$
$\mathsf{tk}^v_{shs} \oplus \mathsf{tk}^c_{shs} \leftarrow \mathsf{DeriveTK}(\mathrm{SHTS}^v \oplus \mathrm{SHTS}^c)$		$\mathsf{tk}_{shs} \leftarrow \mathsf{DeriveTK}(\mathrm{SHTS})$
		{`+EncryptedExtensions` }
		{`+CertificateRequest` }*
		{`+ServerCertificate` }: $\mathsf{pk}_{\mathcal{S}}$
		{`+ServerCertificateVerify` }: $\mathsf{Sig}_{\mathcal{S}} \leftarrow \mathsf{Sign}(\mathsf{sk}_{\mathcal{S}}, \mathrm{Label}_7 \parallel \mathrm{H}_3)$
$\mathsf{fk}_{\mathcal{S}} \leftarrow \mathsf{HKDF.Expand}(\mathrm{SHTS}^v \oplus \mathrm{SHTS}^c, \mathrm{Label}_4 \parallel \mathrm{H}_\epsilon)$		$\mathsf{fk}_{\mathcal{S}} \leftarrow \mathsf{HKDF.Expand}(\mathrm{SHTS}, \mathrm{Label}_4 \parallel \mathrm{H}_\epsilon)$
		{`+ServerFinished` } : $SF \leftarrow \mathsf{HMAC}(\mathsf{fk}_{\mathcal{S}}, \mathrm{H}_4)$
	Forward encrypted {`EE`},. . .,{`SF`} to Verifier	
Reveal SHTS^v to Client		
	Derive tk_{chs}using SHTS^v	
	abort if $\mathsf{Verify}(\mathsf{pk}_{\mathcal{S}}, \mathrm{Label}_7 \parallel \mathrm{H}_3, \mathsf{Sig}_{\mathcal{S}}) \neq 1$	
	abort if `SF` $\neq \mathsf{HMAC}(\mathsf{fk}_{\mathcal{S}}, \mathrm{H}_4)$	
	Forward `SF`, $\sigma \leftarrow \Pi.\mathsf{Prove}(R, \mathsf{Sig}_{\mathcal{S}}, \mathrm{Label}_7 \parallel \mathrm{H}_3)$ to Verifier	
	Reveal $\mathsf{fk}_{\mathcal{S}}$ to Verifier	
	Forward H_4, H_3 and H_2 to Verifier	
abort if `SF` $\neq \mathsf{HMAC}(\mathsf{fk}_{\mathcal{S}}, \mathrm{H}_4)$ or $0 \leftarrow \Pi.\mathsf{Verify}(R, \sigma, \mathrm{Label}_7 \parallel \mathrm{H}_3)$		
$\mathrm{MS}^v \oplus \mathrm{MS}^c \leftarrow \mathsf{HKDF.Extract}(\mathrm{dHS}^v \oplus \mathrm{dHS}^c, \varnothing)$		$\mathrm{MS} \leftarrow \mathsf{HKDF.Extract}(\mathrm{dHS}, 0)$
$\mathrm{CATS}^v \oplus \mathrm{CATS}^c \leftarrow \mathsf{HKDF.Expand}(\mathrm{MS}^v \oplus \mathrm{MS}^c, \mathrm{Label}_5 \parallel \mathrm{H}_2)$		$\mathrm{CATS} \leftarrow \mathsf{HKDF.Expand}(\mathrm{MS}, \mathrm{Label}_5 \parallel \mathrm{H}_2)$
$\mathrm{SATS}^v \oplus \mathrm{SATS}^c \leftarrow \mathsf{HKDF.Expand}(\mathrm{MS}^v \oplus \mathrm{MS}^c, \mathrm{Label}_6 \parallel \mathrm{H}_2)$		$\mathrm{SATS} \leftarrow \mathsf{HKDF.Expand}(\mathrm{MS}, \mathrm{Label}_6 \parallel \mathrm{H}_2)$
$\mathsf{tk}^v_{capp} \oplus \mathsf{tk}^c_{capp} \leftarrow \mathsf{DeriveTK}(\mathrm{CATS}^v \oplus \mathrm{CATS}^c)$		$\mathsf{tk}_{capp} \leftarrow \mathsf{DeriveTK}(\mathrm{CATS})$
$\mathsf{tk}^v_{sapp} \oplus \mathsf{tk}^c_{sapp} \leftarrow \mathsf{DeriveTK}(\mathrm{SATS}^v \oplus \mathrm{SATS}^c)$		$\mathsf{tk}_{sapp} \leftarrow \mathsf{DeriveTK}(\mathrm{SATS})$
	{`+ClientCertificate` }*: $\mathsf{pk}_{\mathcal{C}}$	
	{`+ClientCertificateVerify:`}*: $\mathsf{Sig}_{\mathcal{C}} \leftarrow \mathsf{Sign}(\mathsf{sk}_{\mathcal{C}}, \mathrm{Label}_8 \parallel \mathrm{H}_5))$	
Reveal CHTS^v to Client		
	$\mathsf{fk}_{\mathcal{C}} \leftarrow \mathsf{HKDF.Expand}(\mathrm{CHTS}^v \oplus \mathrm{CHTS}^c, \mathrm{Label}_4 \parallel \mathrm{H}_\epsilon)$	
		$\mathsf{fk}_{\mathcal{C}} \leftarrow \mathsf{HKDF.Expand}(\mathrm{CHTS}, \mathrm{Label}_4 \parallel \mathrm{H}_\epsilon)$
	{`+ClientFinished:`} `CF` $\leftarrow \mathsf{HMAC}(\mathsf{fk}_{\mathcal{C}}, \mathrm{H}_6)$	
		abort if $\mathsf{Verify}(\mathsf{pk}_c, \mathrm{Label}_8 \parallel \parallel \mathrm{H}_5, \mathsf{Sig}_{\mathcal{C}}) \neq 1$
		abort if `CF` $\neq \mathsf{HMAC}(\mathsf{fk}_{\mathcal{C}}, \mathrm{H}_6)$

such as modular exponentiation. We calculate their cost in terms of their number of rounds and bandwidth requirements.

MPC primitives. We use both types of 2PC protocols: maliciously-secure authenticated garbling implementation provided by emp [49] for binary operations, and we base 2PC arithmetic operations on *oblivious transfer* (OT).

Definition 1 (Oblivious Transfer (OT))**.** *An oblivious transfer scheme,* OT*, consists of the following algorithms:*

- $\mathsf{OT.Gen}(1^\lambda)$*: outputs any key material.*
- $\mathsf{OT.Exec}(m_0, m_1, b)$*: The sender P_1 inputs messages m_0 and m_1, and the receiver P_2 inputs a bit $b \in \{0, 1\}$. The receiver P_2 learns m_b, while P_1 learns nothing.*

We realise the OT functionality via the actively secure IKNP [50], [51] extension and the Ferret [37] OT scheme. Both rely on the security of information theoretic MACs, the *learning parity with noise* (LPN) assumption, and on randomness assumptions about hash functions, see [52].

Using OT as a building block, we realise the remaining 2PC functionality needed by using *multiplicative-to-additive* (MtA) secret sharing schemes.

Definition 2 (MtA)**.** *An MtA scheme,* MtA, *consists of the following algorithms:*

- $\mathsf{MtA.Gen}(1^\lambda)$*: outputs any needed key material.*
- $\mathsf{MtA.Mul}(\alpha, \beta)$*: each P_i supplies a_i, learning as output b_i, such that $\sum b_i = \Pi_i a_i$.*

For malicious security, we expand this definition with an additional algorithm, $\mathsf{MtA.Check}(a_1, \ldots, b_1, \ldots)$, to check share consistency. Existing works [2], [12], [15] realise MtA with an approach [53] based on Paillier encryption [54]. We deviate from this approach to improve efficiency [55, §5], and to mitigate the need for range proofs [21], [22] (necessary for achieving malicious security). We realise the MtA functionality using the schemes introduced in [56] and [57], [58] for rings of characteristic > 2 and 2, respectively. The schemes require OT functionality and are instantiated with 128-bit statistical and computational security. We note that whilst the security of [57], [58] reduces directly to an NP-hard encoding problem [59], to the best of our knowledge, there is no computational hardness proof for [56].

ECtF. During the *Key Exchange* phase of the handshake of DCTLS, both $\mathcal{V}$ and $\mathcal{C}$ hold additive shares Z_v and Z_c of a shared ECDH key $(x, y) = \text{DHE}$. Given that all key derivation operations are carried out on the x co-ordinate of Z, we use the *elliptic curve to field* (ECtF) functionality [2] to produce additive shares t_v and t_c of the x coordinate, which is an element in $\mathbb{F}_q$. Using these shares as inputs to the subsequent 2PC operations to derive the handshake secrets allows running all computation in a binary circuit, which results in a substantial performance improvement compared with attempting to combine arithmetic and binary approaches in a garbled circuit. We stress that use of the ECtF functionality improves performance: we estimate that computing just the x co-ordinate of $Z_v + Z_c$ in a garbled circuit would be more expensive than deriving all TLS session secrets, requiring around 1.7M AND gates for an elliptic curve over a field with a 256-bit prime. From a security perspective, we remark that the security of the ECtF functionality reduces to the security of the underlying secure multiplication protocol. We achieve malicious security by instantiating the multiplication with a maliciously-secure MtA scheme.

IV. DiStefano Protocol

In this section, we fully describe each of the phases of the DiStefano protocol (formal ideal functionalities are given in Appendix B). A diagram of the full protocol is found in Fig. 2. In the full version of our work [30], for comparison, we provide a diagram of TLS 1.3 and a summary of the shorthands used (from [29]). The security analysis is found in Section VI.

A. Handshake Phase: HSP

We use the similar overarching mechanism for the handshake phase as described in Section II-B, but focused exclusively on TLS 1.3 with AES-GCM as the AEAD scheme (Section A-B), using ECDH for the shared key generation, and using ECDSA certificates. The 2PC ideal functionalities that we use are defined in the full version [30]. The protocol can be adapted to work with any other TLS 1.3-compliant ciphersuites that are compatible with 2PC.

At a high-level, we adapt the TLS 1.3 handshake by treating $\mathcal{C}$ and $\mathcal{V}$ as a *single* TLS client from the perspective of $\mathcal{S}$. For this, we reverse the "traditional" flow of the TLS 1.3 handshake by having $\mathcal{C}$ and $\mathcal{V}$ each prepare an additively shared ephemeral key share SSK, as seen in Fig. 2. This can be computed without 2PC.

$\mathcal{C}$ then sends the CH and the CKS messages, advertising SSK as part of the `key_shares` extension. $\mathcal{S}$ then processes these messages and, in turn, sends a SH message back to $\mathcal{C}$ containing a freshly sampled ECDH `key_share` Z_s. At this stage, $\mathcal{S}$ computes the shared ECDH key as $DHE = x_s \cdot SSK$ and continues to derive all traffic secrets (i.e. $\text{CHTS}, \text{SHTS}, \mathsf{tk}_{shs}, \mathsf{tk}_{chs}$). Once $\mathcal{C}$ and $\mathcal{V}$ receive the SH message, they derive additive shares of the shared ECDH key as $E = x_c \cdot Y_s + x_v \cdot Y_s$. As TLS 1.3 key derivation operates on the x coordinate of the shared key, $\mathcal{C}$ and $\mathcal{V}$ convert their additive shares of $E = (E_x, E_y)$ into additive shares $E_x = t_c + t_v$ by running the ECtF functionality. With E_x computed, $\mathcal{C}$ and $\mathcal{V}$ proceed to run the handshake key derivation circuit in 2PC, with each party learning shares $\text{HS}^v \oplus \text{HS}^c = \mathsf{HKDF.Extract}(\varnothing, t_v + t_c)$. In practice, this process is carried out inside a garbled circuit that produces shares of CHTS, SHTS and dHS, as well as the SF message key $\mathsf{fk}_{\mathcal{S}}$. This key is provided to both $\mathcal{C}$ and $\mathcal{V}$. This circuit comprises of around 800K AND gates, which is similar to DECO's circuit size for TLS 1.2. We delay the derivation of the traffic keys as it provides authenticity guarantees to $\mathcal{V}$.

Authentication phase. $\mathcal{S}$ sends the CR (if wanted), SCRT, SCV and SF messages. The SF message is computed by first deriving a finished key $\mathsf{fk}_{\mathcal{S}}$ from SHTS and then computing a MAC tag over a hash of all the previous handshake messages. At this point, $\mathcal{S}$ is able to compute the client application traffic secret, CATS, and the server application traffic secret, SATS. $\mathcal{S}$ can also start sending encrypted application data (under tk_{sapp}) while waiting for the final flight of $\mathcal{C}$ messages.

$\mathcal{C}$ receives the encrypted messages from $\mathcal{S}$ and, in turn, forwards them (encrypted) to $\mathcal{V}$ alongside a commitment to their share of CHTS/SHTS. This commitment is necessary to make AES-GCM act as a committing cipher from the perspective of $\mathcal{V}$, which allows $\mathcal{V}$ to disclose their shares of CHTS and SHTS to $\mathcal{C}$ without compromising authenticity guarantees. As $\mathcal{C}$ now knows the entirety of CHTS and SHTS, they are able to locally derive the handshake keys tk_{chs} and tk_{shs}, allowing them to check $\mathcal{S}$'s certificate and

Figure 3. The DiStefano query phase. Purple represents messages sent or calculated by $\mathcal{V}$, orange by $\mathcal{C}$, pink by $\mathcal{S}$, and black for 2PC between $\mathcal{C}$ and $\mathcal{V}$.

Verifier **Client** **Server**

$$\hat{q} \leftarrow \mathsf{AEAD.Enc}(\mathsf{tk}^{v}_{capp} \oplus \mathsf{tk}^{c}_{capp}, IV_c, q)$$

$$q \leftarrow \mathsf{AEAD.Dec}(\mathsf{tk}_{capp}, IV_c, \hat{q})$$

$$\hat{r} \leftarrow \mathsf{AEAD.Enc}(\mathsf{tk}_{sapp}, IV_s, r)$$

$$r \leftarrow \mathsf{AEAD.Dec}(\mathsf{tk}^{v}_{sapp} \oplus \mathsf{tk}^{c}_{sapp}, IV_s, \hat{r})$$

SF messages without the involvement of $\mathcal{V}$. Moreover, as $\mathcal{C}$ now knows tk_{chs} they are also able to respond to the CR if one exists. $\mathcal{C}$ then forwards an authentic copy of the hashes H_2, H_3, and H_4 to $\mathcal{V}$, allowing them to check the SF message as is done in TLS 1.3. Notice that $\mathcal{C}$ does not forward the decrypted SCRT message to $\mathcal{V}$, as this message reveals the identity of $\mathcal{S}$. Let R be a set of TLS certificates, corresponding to a set of pre-approved TLS servers ($\mathcal{S}$s). At this point, $\mathcal{C}$ can (optionally) send to $\mathcal{V}$ a zero-knowledge proof ($\sigma \leftarrow \Pi.\mathsf{Prove}(R, \mathsf{Sig}, \mathrm{Label}_7 \,\|\, \mathrm{H}_3)$) of a valid TLS signature (ZKPVS), Sig, as long as $\mathcal{S} \in R$[4]. $\mathcal{V}$ can then check the validity of the produced ZKPVS proof (if present) by checking that $1 \leftarrow \Pi.\mathsf{Verify}(R, \sigma, \mathrm{Label}_7 \,\|\, \mathrm{H}_3)$.[5] Similarly, $\mathcal{C}$ can selectively reveal the blocks containing the SF message, allowing $\mathcal{V}$ to validate the SF (note that $\mathsf{fk}_{\mathcal{S}}$ was computed in 2PC, but it is revealed after the commitment). Finally, $\mathcal{C}$ and $\mathcal{V}$ derive the shares of the traffic secrets $\mathrm{MS}, \mathrm{CATS}, \mathrm{SATS}$ and the traffic keys $\mathsf{tk}_{sapp}, \mathsf{tk}_{capp}$ in 2PC. In practice, we instantiate this derivation as a garbled circuit that contains around 700K AND gates. Note that this circuit cannot cheaply be combined with the handshake secret derivation circuit, as deriving the traffic keys requires a hash of the unencrypted handshake transcript. This would require decrypting and hashing large messages inside a garbled circuit, which is expensive.

B. Query Execution Phase: QP

Once HSP has completed, $\mathcal{C}$ and $\mathcal{V}$ move into the query phase (Fig. 3). For simplicity, we describe this portion of the protocol in terms of a single round of queries, before extending the phase to multiple rounds.

During the query phase of the protocol, $\mathcal{C}$ produces a series of queries $q = q_1, \ldots, q_n$ and jointly encrypts these with $\mathcal{V}$, with both parties learning a vector of ciphertexts $\hat{q}$ as output. Then, $\mathcal{C}$ forwards $\hat{q}$ to $\mathcal{S}$, receiving an encrypted response $\hat{r}$ in exchange. At this stage of the protocol, $\mathcal{C}$ forwards $\hat{r}$ to $\mathcal{V}$ so that both parties may verify the tags on $\hat{r}$: both parties learn a single bit indicating if the tag check passed or not. In practice, we instantiate this portion of the protocol using the AES-GCM approach described in Section V. There is no explicit dependence on AES-GCM: any AEAD cipher supported by TLS 1.3 will suffice. We highlight this and provide the general security formalisation of the phase in Section VI.

[4]We detail a formalisation of ZKPVS schemes in Section A-C. Practical variants of such schemes exist for ECDSA signatures [60]–[62].

[5]Sending and verifying the proof can be alternatively performed later in the DCTLS protocol, without compromising security.

Extending the query phase to multiple rounds is straightforward using AES-GCM. We discuss the details of committing to ciphertexts in Section V-A, but the main idea is that, as each ciphertext block q_i is encrypted with a unique key $e_i = \mathsf{AES.Enc}(k, IV + i)$, $\mathcal{C}$ and $\mathcal{V}$ can arbitrarily reveal their shares of e_i at any stage of the query phase, provided an appropriate commitment has been made beforehand. As these key shares are ephemeral, revealing them does not compromise the shares derived during the HSP. The security of this approach directly reduces to the difficulty of recovering an AES key from many known plaintext/ciphertext pairs. This permits many useful applications, as $\mathcal{C}$ and $\mathcal{V}$ can now nest commitment rounds inside of the query phase.

C. Commitment Phase: CP

The objectives of the commitment phase (Fig. 4) are: i. to assure $\mathcal{V}$ of the authenticity of $\mathcal{S}$ (as belonging to the pre-approved set R) without revealing the exact server $\mathcal{C}$ communicated with; and ii. to allow $\mathcal{C}$ to learn secrets held by $\mathcal{V}$ only after producing binding commitments to a specific portion of the TLS session with $\mathcal{S}$.[6]

To validate the authenticity of the server, $\mathcal{V}$ verifies a proof (with a ZKPVS scheme) of the TLS server that they communicated with as one of N servers from which $\mathcal{V}$ accepts attestations.[7] After verification, $\mathcal{C}$ can now commit to and reveal information about the application traffic they witness. For this, first, we define a commitment scheme (Γ) that can be implicitly constructed using the outputs of QP with the following algorithms.

- $(\hat{q}_i, \hat{r}_i) \leftarrow \Gamma.\mathsf{Commit}(\mathsf{sp}_{\mathcal{C}}, (\hat{q}, \hat{r}, i))$: For the input i, output the ciphertexts $(\hat{q}_i, \hat{r}_i)$ corresponding to the i^{th} query q_i, and the response r_i.
- $\mathsf{sp}_{\mathcal{V}} \leftarrow \Gamma.\mathsf{Challenge}((\hat{q}_i, \hat{r}_i))$: Output the secret parameters of $\mathcal{V}$: reveals $\mathcal{V}$'s key share and a challenge.
- $b \leftarrow \Gamma.\mathsf{Open}((\mathsf{sp}_{\mathcal{C}}, \mathsf{sp}_{\mathcal{V}}), (\hat{q}_i, \hat{r}_i), (q_i, r_i))$: Check that $(\hat{q}_i, \hat{r}_i)$ decrypts to (q_i, r_i), and output $b = 1$ on success, and $b = 0$ otherwise.

Note that the client simply commits to encrypted traffic exchanged during QP (using 2PC to encrypt and decrypt the traffic). When it comes to opening the encrypted traffic, the protocol requires $\mathcal{V}$ to reveal their key secret share, so that $\mathcal{C}$ can decrypt and then reveal the plaintext values. We give a concrete construction of Γ that is perfectly hiding and computationally binding, based on AES-GCM, in Section C-B.

D. Subsequent Phases

It is important to note that in the real DiStefano protocol, $\mathcal{C}$ does not send any unencrypted values to $\mathcal{V}$. Instead, both parties should execute a protocol that proves certain facts about the DCTLS commitments, without revealing anything else. This could be done using zero-knowledge proofs, selective opening strategies (as is used in DECO), or subsequent 2PC.

[6]Note this applies for both the handshake and record phases.

[7]This could be performed during the handshake phase. For performance reasons, it is preferable to do so in the commitment phase, when online communication is no longer constrained by potential handshake time-outs.

Figure 4. The DiStefano commitment phase with a commitment scheme, Γ, and a ZKPVS scheme, Π for TLS signatures under a set R of pre-approved servers. Purple represents messages sent or calculated by $\mathcal{V}$, and orange by $\mathcal{C}$.

Verifier **Client**

$\sigma \leftarrow \Pi.\mathsf{Prove}(R, \mathsf{Sig}, \mathsf{Label}_7 \parallel \mathsf{H}_3)$

abort if $1 \neq \Pi.\mathsf{Verify}(\mathsf{R}, \sigma, \mathsf{Label}_7 \parallel \mathsf{H}_3)$

$c \leftarrow \Gamma.\mathsf{Commit}((\mathsf{tk}^c_{capp}, \mathsf{tk}^c_{sapp}), (\hat{q}, \hat{r}))$

Forward $\mathsf{tk}^v_{capp}, \mathsf{tk}^v_{sapp}$ to Client

The formal commitment opening process that we described previously can be used for this, since $\mathcal{C}$ can now use the combined secret parameters $(\mathsf{sp}_{\mathcal{C}}, \mathsf{sp}_{\mathcal{V}})$ to prove any statement about the commitment $(\hat{q}_i, \hat{r}_i)$. Note that the proving process can inadvertently leak the identity of $\mathcal{S}$ (invalidating the ZKPVS proof) if a certain data is assumed from $\mathcal{S}$'s traffic. See the full version [30] for a wider discussion.

V. Adapting AES-GCM For 2PC

AES-GCM is an authenticated encryption with associated data (AEAD) cipher that features prominently inside TLS implementations, with some works reporting that over 90% of all TLS1.3 traffic is encrypted using AES-GCM [31]. We use this algorithm for both encrypting the corresponding handshake messages and any application traffic. Here we describe how to commit to decryptions of ciphertexts in AES-GCM (Section V-A), as well as optimisations that make it more amenable to 2PC evaluation of the encryption and decryption procedures (Section V-B). We briefly recall how AES-GCM operates.

AES-GCM Encryption. Let k and IV refer to an encryption key and initialisation vector. Given as input a sequence of n appropriately padded plaintext blocks $M = (M_1, \ldots, M_n)$, AES-GCM applies counter-mode encryption to produce the ciphertext blocks $C_i = M_i \oplus \mathsf{AES.Enc}(k, IV + i)$. To ensure authenticity, the algorithm outputs a tag $\tau = \mathsf{Tag}_k(A, C, k, IV)$ computed over C and any associated data A (i.e. any data that is authenticated but not encrypted [63], e.g, protocol headers or metadata) as follows:

- Given some vector $\mathbf{x} \in \mathbb{F}^m_{2^{128}}$, we define the polynomial $P_x = \sum_{i=1}^{m} x_i \cdot h^{m-i+1}$ over $\mathbb{F}_{2^{128}}$.
- Assuming that C and A are properly padded, we compute τ (where $h = \mathsf{AES.Enc}(k, 0)$) as: $\tau(A, C, k, IV) = \mathsf{AES.Enc}(k, IV) \oplus P_{A||C||\mathsf{len}(A)||\mathsf{len}(C)}(h)$.

A. Commitment to AES-GCM Ciphertexts

In DiStefano, both $\mathcal{C}$ and $\mathcal{V}$ learn all AES-GCM ciphertext blocks $C_i = M_i \oplus \mathsf{AES.Enc}(k, IV + i)$ produced by $\mathcal{S}$, where the session key $k = k_c + k_v$ is secret-shared across both $\mathcal{C}$ and $\mathcal{V}$. We briefly describe how $\mathcal{C}$ can commit to the received ciphertext blocks C_i without revealing their key share k_c. Note first that the use of AES-GCM in an AEAD setting leads to a non-committing cipher [64], meaning that an adversary in possession of a key k and a valid ciphertext block $C_i = \mathsf{AES.Enc}(k, IV + i) \oplus M_i$ can find a distinct $k' \neq k$ such that $C_i = M'_i \oplus \mathsf{AES.Enc}(k', IV + i)$ is a valid ciphertext. From the perspective of DCTLS protocols, this non-committing nature of the algorithm presents a challenge, as $\mathcal{C}$ typically only proves statements after learning the entirety of the key. We circumvent the issue as follows.

Assume that $\mathcal{C}$ receives a single tuple of an AES-GCM ciphertext and tag, (C_i, τ), from $\mathcal{S}$ that they wish to decrypt. As $\mathcal{C}$ only holds a share (k_c) of k, $\mathcal{C}$ cannot decrypt C_i by themselves. Thus, $\mathcal{C}$ forwards (C_i, τ) to $\mathcal{V}$, and they engage in a maliciously-secure 2PC protocol to validate τ on C_i (Algorithm 2). If it succeeds, then both $\mathcal{C}$ and $\mathcal{V}$ are convinced that C_i is a valid ciphertext under k. Yet, we must be careful how we reveal $M_i = \mathsf{AES.Enc}(k, IV + i) \oplus C_i$ to each party as revealing M_i to $\mathcal{C}$ allows them to mount the outlined non-committing attack, while revealing M_i to $\mathcal{V}$ would violate the privacy guarantees of DCTLS. In order to resolve this issue, we use a modified 2PC AES decryption protocol that, after checking that the client input masks match the commitments held by $\mathcal{V}$ and validating τ, outputs $e_i = \mathsf{AES.Enc}(k, IV + i)$ to $\mathcal{C}$ and a commitment, E_i, to e_i to $\mathcal{V}$. With this, notice that $\mathcal{C}$ is unable to exploit the non-committing nature of AES-GCM as a binding commitment to e_i is created and validated by $\mathcal{V}$. Moreover, $\mathcal{C}$ can now reveal individual blocks to $\mathcal{V}$ without requiring either party to reveal their key share: $\mathcal{C}$ can reveal a particular block C_i by simply forwarding e_i to $\mathcal{V}$. In other words, this tweak allows $\mathcal{C}$ to engage in a *selective opening* protocol with $\mathcal{V}$ (we provide details of this technique in the full version [30]). Producing these set of commitments is cheap as, in practice, we simply require $\mathcal{C}$ to commit to n unique masks b_i and then output $E_i = \mathsf{AES.Enc}(k, IV + i) \oplus b_i$. We formalise this scheme in Section C-B. Checking random-oracle commitments in 2PC is practical. Using a low-depth hash function, such as LowMC [65], this would cost 4370 AND gates for the full commitment check [66], which is cheaper than a single AES block evaluation (6400 AND gates). Above all, the practical costs of the 2PC commit scheme are low, and we demonstrate this using a higher-depth, AES-based hash (see Section VII).

B. 2PC Performance Optimisations

We now discuss some cryptographic optimisations that are necessary for ensuring the high performance of AES-GCM encryption/decryption during online TLS operations. The ideal functionalities that we use to describe AES-GCM in 2PC and the security proofs of the optimisations are given in Section C-A.

Efficiency. Despite its simplicity, executing AES-GCM encryptions in a multi-party setting can be challenging due to the use of binary and arithmetic operations. For example, whilst AES operations are well-suited for garbled circuits, a single multiplication over $\mathbb{F}_{2^{128}}$ typically requires around 16K AND gates, increasing the cost by nearly a factor of 3. To mitigate this cost, both [2] and [12] recommend computing shares of the powers of h (denoted as $\{h^i\}$) during an offline setup stage, amortising the cost across the entire session. In certain settings, this cost can be reduced further by restricting how many powers of h are used: for example, MPCAuth [18] employs a clever message slicing strategy to minimise the value of i with the assumption that servers only send rather small ciphertexts (e.g., at most 4KiB). However, TLS 1.3 servers can send ciphertexts bigger than that in order to avoid overhead: web servers, for example, that deliver video stream [67] or large files [68] might send responses significantly larger than 4KiB, requiring more ciphertext blocks to be handled. It is recommended [69], for example, that when the connection congestion window is large and a large stream (e.g., streaming video) is transferred, that the size of the TLS 1.3 record should be increased to span multiple TCP packets (up to 16KiB) to reduce framing and CPU overhead. There is no "optimal" record size, but rather it is dynamically adjusted based on the state of the TCP connection. Due to this, in DiStefano we allow for this flexibility, as we explicitly target the largest possible TLS ciphertext of 16KiB, which corresponds to $i = 1024$.

Assuming that a sharing $(\{h_c^i\}, \{h_v^i\})$ exists, producing tags in 2PC is straightforward: tagging n blocks requires two local polynomial evaluations (writing $\tau_c = P_{A||c||\mathsf{len}(A)||\mathsf{len}(c)}(\{h_c^i\})$ and $\tau_v = P_{A||c||\mathsf{len}(A)||\mathsf{len}(c)}(\{h_v^i\})$, respectively) over $\mathbb{F}_{2^{128}}$ and $n+1$ 2PC evaluations of AES [2], [12]. The final tag is achieved by simply computing $\tau = \tau_c + \tau_v \oplus \mathsf{AES.Enc}(k_c + k_v, IV_c)$. In order to make this more efficient, it is necessary to initially construct a 2PC protocol that evaluates the ciphertext c and outputs to both parties, and then have a subsequent protocol that computes the tag for this ciphertext, based on the local polynomials submitted by the client.

Our optimisations. DECO gives few details on how to compute shares of the powers of h, other than that they are computed in a 2PC session. We remark that calculating these shares in a garbled circuit is unlikely to be feasible: our adapted version of MPCAuth's share derivation circuit contained around 17M AND gates, and required over 900MiB and 18GiB of network traffic and memory, respectively, just for the pre-processing stage. For comparison, our circuits for TLS 1.3 secret derivation contain around 1.3M AND gates in total, which is approximately a factor of 14 smaller. Thus, using only garbled circuits is unlikely to be feasible.

Several other approaches exist for computing the shares of $\{h^i\}$. For instance, PageSigner [12] reduces computing additive shares of h^i to simply computing shares using MtA computations. Given an initial additive sharing $h = h_c + h_v$, $\mathcal{C}$ and $\mathcal{V}$ iteratively compute additive shares of $\ell_c + \ell_v = h^n = (h_c + h_v)^{n-1}(h_c + h_v)$ for $1 < n \leq 1024$. This approach permits an additional optimisation: as $(x+y)^2 = x^2 + y^2$ over $\mathbb{F}_{2^{128}}$, each party can compute shares of even powers of h locally. Taking this optimisation into account, producing shares in this way costs a total of 1022 MtA operations. However, as computing shares of any odd h^i requires first computing shares of h^{i-2}, the approach seems to require around 500 rounds, which is likely too slow for a WAN setting.

We improve upon this by replacing the additive sharing $h = h_1 + h_2$ with $h = h_1/h_2$, i.e. using a multiplicative sharing. By using multiplicative shares, we can run each MtA computation in parallel, with each P_i supplying $h_i, {h_i}^3, \ldots, {h_i}^{1023}$ as input. This optimisation asymptotically halves the number of MtA operations and reduces the round complexity to a single round, as the only costly operation is the share's computation, which is not carried out inside the garbled circuit. However, this tweak does require a slightly more complicated scheme for computing the initial sharing of h, as we now must compute multiplication over $\mathbb{F}_{2^{128}}$, taking the size of the circuit for deriving the initial shares to around 23K AND gates. In practice, we reduce the size of this circuit to around 18K AND gates by instead using a carry-less Karatsuba [70] algorithm. Whilst this still represents an increase of around a factor of 3 compared to the additive circuit, the reduction in MtA operations and rounds means that we are able to achieve an end-to-end speed-up of around a factor of 3. We discuss these results in more detail in Section VII.

VI. Security Analysis

Previous DCTLS protocols use all-encompassing ideal functionalities and Universally-Composable (UC) security proofs [71], proving that the entire flow from handshake to attestation is secure. This is problematic for cryptographic agility, as it means that any modification to the TLS ciphersuite, 2PC functionality, or protocol extensions would necessitate a complete rewrite of the proof. Such agility is critical for building flexible secure systems, that can be modified easily even if primitives and systems change [36].

We reimagine the security model for DCTLS protocols in two ways. First, our analysis breaks the protocol down into three phases: handshake, query, and commitment; and guarantees security of each independently. Second, where possible, we adapt the security analysis to either the *standalone* or *game-based* security models. UC security proofs are particularly useful when building atomic protocol primitives, intended for arbitrary composition with other primitives. Since DCTLS is a high-level protocol that is likely to be used as a single application, we believe that our approach captures a natural security requirement, without the added complexity for ensuring UC security. Without this complexity, modifying both individual phases of the protocol and cryptographic primitives is an easier task for future work. We give a short overview of how we model security for each phase in the following. The full standalone security model is covered in the full version [30].

Handshake phase. We model the handshake phase as in the work of Oblivious TLS [17]. Essentially, this model proves that we can satisfy the original guarantees proven about TLS 1.3 [29] (i.e. related to *Match* and *Multi-Stage* security) even when executing certain functionalities in 2PC. One key difference noted by [17] is that the presence of the verifier means that potential TLS 1.3 adversaries can alter the derivation of secrets in the client, and thus security is based on a modified *Shifted PRF ODH* assumption, see [17, Definition 2] for more details. Oblivious TLS produces a UC-security proof for the handshake phase of the protocol, based on maliciously-secure 2PC primitives, and two parties. In our setting, we have an information imbalance, in that $\mathcal{V}$ has fewer powers than one of the two parties analysed in [17], as *only* $\mathcal{C}$ interacts with $\mathcal{S}$ directly, while $\mathcal{V}$ does not interact with $\mathcal{S}$ directly at all. As a result, the analysis of [17] is sufficient for covering our required security guarantees, as long as each 2PC primitive is maliciously-secure. We provide an overview of the protocol execution and a detailed analysis of security in the full version [30].

Query execution phase. The query phase of DiStefano essentially amounts to considering a 2PC realisation of the record-layer of the TLS 1.3 protocol. We define 2PC ideal functionalities that abstract the core encryption and decryption functionality for application traffic. We show that we can prove security of this phase based on the 2PC realisation of the AES-GCM functionalities that we formalise in Section C-A (Algorithms 1 and 2), that together implement the functionality and optimisations described in Section V. To prove security of alternative ciphersuites, it is simply a matter of implementing the 2PC ideal functionalities using different primitives.

Commitment phase. We model the commitment phase in a game-based security model. We provide multiple security notions that evaluate the capacity of the protocol to satisfy: *session privacy* (SPriv), i.e. committed sessions are indistinguishable; *1-out-of-N authentication* (SAuth^1_n), i.e. the client is forced to successfully authenticate the TLS server amongst N possible apriori-chosen servers; and *session unforgeability* (SUnf), i.e. the client cannot arbitrarily forge sessions that did not occur.

In the end, we show that DiStefano satisfies these properties based on the commitment scheme devised from AES-GCM (Section C-B), and the zero-knowledge proof scheme (ZKPVS) for valid TLS signatures [61] (Section A-C).

VII. Experimental Analysis

Implementation. In order to enable easy integration with other cryptographic libraries and browsers, we implemented a full prototype of DiStefano in C++.[8] This implementation contains around 14k lines of code, tests and documentation, and implements the whole protocol. We developed this implementation using C++ best practices, and we hope that this effort is useful for other researchers. Concretely, our implementation of DiStefano uses BoringSSL for TLS functionality and emp for all MPC functionality. BoringSSL is the only cryptographic library supported by Chromium-based Internet browsers. As far as we are aware, our implementation contains primitives and circuits that are not available elsewhere. Our implementation also contains a modified version of MPCAuth's circuit generation to produce the relevant garbled circuits. We further reduce the online cost of MPCAuth's secret sharing scheme by using a pre-determined splitting scheme for specific secrets.[9] We list the changes made alongside our prototype .[10] Note that further performance improvements, including multithreading, are not addressed in this implementation and are the subject of future work.

Results. We evaluated the performance of DiStefano in LAN and WAN settings. For the LAN environment, we use a consumer-grade device (a Macbook air M1 with $8\,\mathrm{GB}$ of RAM) for $\mathcal{C}$, and a server-grade device (an Intel Xeon Gold 6138 with $32\,\mathrm{GB}$ of RAM) for $\mathcal{V}$ and $\mathcal{S}$. All communication used in TLS 1.3 was carried out using a single thread over a $1\,\mathrm{Gbps}$ network with a latency of around $16\,\mathrm{ms}$. For the WAN setting, we provide various settings depending on regions. We place $\mathcal{C}$ in Paris, France using a AWS EC2 "t2.2xlarge" machine for all settings, but locate the AWS EC2 "t2.2xlarge" machine for $\mathcal{V}$ and $\mathcal{S}$ in three regions: Ohio, USA; London, UK; and Seoul, South Korea. We report their latency based on [72]. For the first region, the median round-trip latency is estimated at $94.15\,\mathrm{ms}$ as reported by 13 measurements, and the first and third quartiles are $92.83\,\mathrm{ms}$ and $95.53\,\mathrm{ms}$, respectively. For the second region, the median round-trip latency is estimated at $9.07\,\mathrm{ms}$ as reported by 13 measurements, and the first and third quartiles are $2.08\,\mathrm{ms}$ and $3.31\,\mathrm{ms}$, respectively. For the thrid region, the median round-trip latency is estimated at $247.91\,\mathrm{ms}$ as reported by 13 measurements, and the first and third quartiles are $246.57\,\mathrm{ms}$ and $248.87\,\mathrm{ms}$, respectively. Timings and bandwidth measurements are computed as the mean of 50 samples, and are represented in milliseconds and mebibytes, respectively (1 MiB is 2^{20} bytes).

Table I gives results for each individual circuit used in DiStefano. Each circuit is evaluated without amortization, meaning these timings do not benefit from the amortized pre-processing available in emp. We note that the 2PC-GCM circuit includes both encryption and decryption of traffic, as specified in Algorithms 1 and 2, using a random-oracle (AES-based) commitment scheme. As the most expensive operation of these circuits will only be used once per session, we do not expect that employing amortisation will yield a substantial speed-up. However, employing amortisations for common operations (e.g. AES-GCM tagging and verification) may lead to faster running times (see [49, §7] for concrete speed-ups). We also compare the offline time using the original implementation of authenticated garbling (LeakyDeltaOT [49]) against FerretCOT. Our results show that FerretCOT outperforms the original OT for large circuit sizes in both bandwidth and running time. However, for smaller circuits it appears that the original OT is faster at the cost of more bandwidth. Given that the pre-processing times are proportional to the size of the circuits, our results appear to be predominantly network bound. The results also highlight that our Karatsuba-based circuit achieves modest gains in both bandwidth and time over the naive circuit.

[8]https://github.com/brave-experiments/DiStefano

[9]This approach is less flexible than MPCAuth. For example, DiStefano only supports 2 parties, whereas MPCAuth supports arbitrarily many.

[10]https://github.com/brave-experiments/DiStefano, section "Changes introduced".

Table I
GARBLED CIRCUIT TIMINGS AND BANDWIDTH.

Circuit	OT	Offline	Online	Bandwidth
AES-GCM share (K)	LD	2340	34.92	21.04
AES-GCM share (K)	FC	2683	59.48	9.009
AES-GCM share (N)	LD	2678	39.09	25.63
AES-GCM share (N)	FC	2853	61.36	10.35
AES-GCM Tag	LD	1019	22.30	7.604
AES-GCM Tag	FC	2336	22.22	5.010
AES-GCM Verify	LD	1032	21.16	7.746
AES-GCM Verify	FC	2277	21.24	5.130
TLS 1.3 HS (*P256*)	LD	51470	93.16	305.1
TLS 1.3 HS (*P256*)	FC	19847	88.90	113.3
TLS 1.3 HS (*P384*)	LD	51610	95.38	305.8
TLS 1.3 HS (*P384*)	FC	19940	89.88	113.6
TLS 1.3 TS	LD	51450	95.21	243.7
TLS 1.3 TS	FC	18820	99.25	91.12
2PC-GCM (*256B*)	LD	18690	82	131.4
2PC-GCM (*256B*)	FC	10971	80	47.5
2PC-GCM (*512B*)	LD	31534	136	206.5
2PC-GCM (*512B*)	FC	16010	142	77.75
2PC-GCM (*1KiB*)	LD	57485	252	409.4
2PC-GCM (*1KiB*)	FC	26154	301	151.2
2PC-GCM (*2KiB*)	LD	114820	728	815.4
2PC-GCM (*2KiB*)	FC	48764	763	299.3

Each garbled circuit is reported in terms of offline/online times (ms) and total bandwidth (MiB) costs. "K" means Karatsuba and "N" means Naive. "LD" refers to "LeakyDeltaOT" and "FC" means "FerretCOT". Optimal cases are highlighted in green.

Table II
PRIMITIVE TIMINGS AND BANDWIDTH.

Primitive (algorithm)	Time (ms)	Bandwidth (MiB)
ECtF (*P256*)	336.1	0.768
ECtF (*P384*)	335.5	1.295
ECtF (*P521*)	421.4	2.442
MtA (*P256*)	33.67	0.086
MtA (*P384*)	40.65	0.127
MtA (*P521*)	55.83	0.241
AES-GCM powers (mul.)	1694	0.049
AES-GCM powers (add.)	5926	0.080
AES-GCM powers (GC)	—	900

Table II shows the results for each arithmetic primitive used. The running times and bandwidth usage are notably low, suggesting that these primitives will not pose a bottleneck, even in constrained network environments. Notably, the tweak introduced in Section V-B reduces the running time by a factor of around 3, whilst also halving the required bandwidth for the multiplication (this ignores bandwidth used by shared setup). This all represents an improvement of around 4 orders of magnitude over using a garbled circuit.

The timings indicate that our implementation of DiStefano is competitive with DECO, with the DCTLS online portion taking $\approx 500\,\text{ms}$ to complete for a 256-bit secret in a LAN setting. These conclusions are consistent across both the individual and E2E timings (cf. Table III). Our WAN experiments model (as seen in Table IV) realistic E2E executions in different regions of the world with low to high latency. The results indicate that the running times roughly increase

Table III
E2E TIMINGS (MS) AND BANDWIDTH (MIB) FOR DCTLS IN LAN SETTINGS (WITH LATENCY $\approx 16\,\text{ms}$).

Process	LAN (ms)	Bandwidth (MiB)
Offline costs		
$\mathcal{C}/\mathcal{S}$ Key Share	1.3167	6.67572e-05
$\mathcal{C}/\mathcal{V}$ execute ECtF	0.008083	9.53674e-07
Circuit Preprocessing	6280.08	220.484
Online costs		
$\mathcal{S}$ sends cert.	0.011375	3.14713e-05
Derive traffic secrets	33.1389	0.0276108
Derive GCM shares	136.573	0.0488291

Table IV
E2E TIMINGS (MS) AND BANDWIDTH (MIB) FOR DCTLS IN WAN SETTINGS. ALL TIMES ARE REPORTED IN MS.

Process	WAN-Ohio	WAN-London	WAN-Seoul
Offline costs			
$\mathcal{C}/\mathcal{S}$ Key Share	102.4682	12.2567	252.1662
$\mathcal{C}/\mathcal{V}$ execute ECtF	112.6829	2.5680	285.9367
Circuit Preprocessing	8958.3445	6236.5134	10417.6417
Online costs			
$\mathcal{S}$ sends cert.	125.9581	2.3673	12.2567
Derive traffic secrets	130.9039	43.6089	273.2959
Derive GCM shares	494.3445	149.2039	872.4691

by the amount of latency introduced. The only deviations occurred are related to either circuit preprocessing — which increases it, but is an offline amortisable cost — and the derivation of GCM shares — which reflects the multi-round trip time nature of this part of the protocol. Even so, the GCM shares derivation still takes less than a second, and the total online costs are significantly lower than standard online TLS handshake timeout times which, while configurable, are typically between 10 and 20 seconds [24]. Finally, we note that increasing latencies only appear to impact the protocol sublinearly, and thus we expect that DiStefano across a variety of browsing scenarios.

A. Comparisons with prior work

DECO-like protocols. We note that comparisons between our results and previous DECO-like protocols for committing to TLS 1.2 traffic [2], [12] should be made carefully. In the case of DECO, their implementation is not publicly available, and we were unable to reproduce any of their results. Moreover, as our implementation is single-threaded, we are unable to take advantage of emp's multi-threaded pre-processing. Given that [49, §7] reports an order of magnitude increase in bandwidth due to multi-threading, it is not surprising that our offline times are an order of magnitude higher. However, our online timings are comparable with DECO, and parallelising the pre-processing stage would likely mitigate any discrepancies.[11] As pre-processing can be carried out before, we do not consider

[11]Notably, [2] does not mention if the pre-processing is multi-threaded.

this a major issue. It is also difficult to compare our timings to PageSigner. Their original implementation is written entirely in Javascript, preventing the usage of dedicated hardware resources. Given that our implementation is instead written in C++, we might expect DiStefano to be faster. PageSigner also follows a semi-honest security model and targets TLS 1.2, which are incompatible with DiStefano.

Janus. The work of Janus [15] builds similar DCTLS functionality for certain ciphersuites of TLS 1.3. Their reported timings assume a latency of $0\,\mathrm{ms}$, which fails to model the impact of network bandwidth on protocol runtimes. As such, we can only make coarse-grained analysis of their handshake and record-layer functionality, for their reported $256\,\mathrm{B}$ queries $2\,\mathrm{kB}$ responses. The overall online communication is $1072\,\mathrm{KiB}$, while offline communication is $320\,\mathrm{MiB}$. In DiStefano, the online communication is $76\,\mathrm{KiB}$, while offline communication is $220.5\,\mathrm{MiB}$. In terms of runtime, Janus reports around $2\,\mathrm{s}$ of offline costs, and $1.55\,\mathrm{s}$ of online costs (with latency of $0\,\mathrm{ms}$) when run on an Apple Macbook M1 processor. For DiStefano, in the LAN setting (with latency estimated at $16\,\mathrm{ms}$), offline runtimes are $6.28\,\mathrm{s}$, which is clearly slower than the claims of Janus. However, the online runtimes are significantly quicker: around $0.1\,\mathrm{s}$. This provides much more flexibility in the critical phase of the protocol, which is prone to server timeouts, while the offline phase can be performed at any moment. We reinforce that this comparison is not completely accurate as Janus does not report a full LAN/WAN simulation, where the impact of network latency is included. Finally, as explained in the full version [30], Janus shifts the verification of the 2PC functionality to later stages of the protocol, which can lead to violations of the malicious security model. DiStefano, in comparison, is proven secure in a fully malicious setting.

VIII. Discussion

A. Related Work

As noted in Section II, DiStefano is an instance of a DCTLS protocol. Other alternatives exist, but all have limitations as noted in Section II-B. We summarise the comparison in Table V, and discuss further below.

Table V
COMPARISON OF DCTLS-LIKE PROTOCOLS. †SEE SECTION VIII-A

Protocol	TLS 1.3	Attest	Ring auth
DECO-like [2], [13]	✗	✓	✗
MPCAuth [18]	✓	✗	✗
Oblivious TLS [17]	✓	✗	✗
ZKMiddleboxes [16]	✓	$\mathcal{C} \rightarrow \mathcal{S}$	✗
Janus [15]	✓†	✓	✗
DiStefano	✓	✓	✓

The DECO and PageSigner protocols, for example, only (formally) work for TLS 1.2 and under, and provide limited privacy. TownCrier [13] has similar problems, and requires using trusted computing functionality. Recently, the PECO protocol [39] was proposed, which informally extends the DECO protocol to support TLS 1.3, but provides no formal guarantees nor implementation of it.

MPCAuth [18] allows a user to authenticate to N servers independently by doing the work of only authenticating to one. An N-for-1 authentication system consists of many servers and users. Each user has a number of authentication factors they can use to authenticate. The user holds a secret s that they wish to distribute among the N servers. The protocol consists of two phases. In the *enrollment* phase, the user provides the servers with a number of authentication factors, which the servers verify using authentication protocols: these protocols use a mechanism called "TLS-in-SMPC" that allows N servers to jointly act as a TLS client endpoint to communicate with another TLS server. A single server from the N authorised cannot decrypt any TLS traffic, and, after authenticating with these factors, the client secret-shares s and distributes the shares across the servers. In the *authentication* phase, the user runs the MPCAuth protocols for the authentication factors and, once it is authenticated, the N servers can perform computation over s for the user, which is application-specific (such as key recovery, for instance).

The *Oblivious TLS* protocol [17] allows for any TLS endpoint to obliviously interact with another TLS endpoint, without the knowledge that it is interacting with a multi-party computation instance. It consists of the following phases: i) *Multi-Party Key Exchange*, which is the key exchange phase of the TLS handshake ran in an MPC manner by performing an exponentiation between a known public key and a secret exponent, where the output remains secret; ii) *Threshold Signing*, which is the authentication phase of the TLS handshake done by having the TLS transcript signed with EdDSA Schnorr-based signatures in a threshold protocol; and iii) *Record Layer* which is ran by using authenticated encryption, based on AES-GCM, inside MPC.

Recent work on zero-knowledge middleboxes for TLS 1.3 traffic [16] has many similarities with techniques used in DCTLS protocols. However, the verifier is considered to be an on-path proxy that receives and forwards encrypted traffic between the parties (similar to the proxy model of DECO [2]). Furthermore, the client only produces commitments to their own traffic, rather than the traffic received from the server. Applications include increased corporate oversight and enaction of Internet browsing policies to be enforced by middleboxes, which are naturally thwarted when all client traffic is sent encrypted over TLS.

Finally, the Janus protocol [15] targets TLS 1.3, but with the limitation that the client must fix a ciphersuite apriori, while DiStefano allows for full negotiation of any TLS-supported ciphersuite. In addition, the Janus security model leaves open the possibility of theoretical attacks by malicious actors, see in the full version [30] for more details.

Concurrent work. Xie et al. [14] propose a series of optimisations to the MPC protocols used inside DECO, targeting TLS 1.2. Whilst most of these improvements are orthogonal to our work, one interesting optimisation is a faster approach

Table VI
RESULTS FOR RUNNING KeyUpdate IN 2PC.

Circuit	OT	Offline (ms)	Online (ms)	Bandwidth (MiB)
KeyUpdate	LD	10540	29.95	98.54
KeyUpdate	FC	7960	31.96	31.61

for deriving TLS traffic secrets inside garbled circuits. This approach is reminiscent of the highly optimised CBC-HMAC protocol proposed in DECO [2, §4.2.1] for computing tags in 2PC. We remark that incorporating this particular optimisation into our secret derivation process seems non-trivial: we discuss these difficulties in the full version [30].

B. Applications

Attestations. DiStefano produces commitments to encrypted TLS 1.3 data which, as noted in [2], can be used as the basis of zero-knowledge proofs (or *attestations*) for showing that certain facts are present in such data. However, such attestations can also be constructed via different methods, using cooperative decryption of certain ciphertext blocks, or more generic 2PC techniques. The DECO protocol provides examples that they can prove certain statements for, including proof of confidential financial information, and proof of age. It should be noted that TLS sessions could serve as the basis for more generic user credentials, proving arbitrary facts about a user. For a complete summary, see [2].

Concrete applications of DiStefano include: i. leveraging attestations to produce strong signals of anti-fraudulent behavior by attesting the inclusion of statements over bank account balances or proofs of transactions, which helps prove that a user is not engaging in fraud, as such signals are expensive for bots to replicate; ii. using attestations to verify "real" events by producing statements that confirm the origin and trustworthiness of data, helping to demonstrate that it was not generated by Artificial Intelligence (AI); and iii. attesting to a user's honesty (by exporting multiple TLS attestations) over a long time period, to grant them access to other privacy-preserving protocols [73], [74].

C. Limitations

Our implementation of DiStefano does not support key rotation via KeyUpdate messages or full 0-RTT mode, but this limitation is not major: it can be circumvented by simply re-running the HSP.[12] We also provide no concrete instantiation of the zero-knowlege primitives that can be used to create attestations, but they should follow the guidelines stated in Section IV-D. Said proofs must also be mindful of user privacy concerns: if proof circuits explicitly target server-specific HTML formats, this will undo the zero-knowledge authentication privacy guarantees of the ZKPVS approach. Note also that the ZKPVS scheme preserves anonymity only amongst the set R: we address this point in Section A-C.

DCTLS protocols assume user commitments are *meaningful*, and that $\mathcal{S}$ stores only *correct* data. Suppose a user wants to provide a proof of their age from a government agency website. They will log in to the website and then run DiStefano to produce a commitment to their age, based on the data present. This process assumes the authentication process is sound, which may not be the case (if the account is stolen or fake). $\mathcal{V}$ should only accept commitments from a $\mathcal{S}$ that it trusts to correctly store user data.

It is important to note that DCTLS protocols could become actively harmful tools for monitoring or censoring client traffic in certain applications, especially in automated systems without human involvement. For instance, they could be exploited to scan and censor specific statements posted by digital activists on social media in order to censor them, which is a known technique used around the world [75], [76]. Thus, we would like to emphasise that deployment of tools such as DiStefano must be considered carefully in such contexts. Furthermore, DCTLS can be subject to different legal and compliance issues if considered as a form of webscraping. The compliance of DCTLS tools with a given website's terms of service, for example, is an application-specific question in their legal context.

D. Browser Integration

DiStefano can be integrated into any browser that uses BoringSSL, e.g. Google Chrome/Brave, easily. As our changes to BoringSSL itself are rather minimal, it would be possible to describe our changes as a series of deltas in a version control system, which can then be applied during the process of building the browser based on build flags.[13] We leave the completion and deployment as future work.

IX. CONCLUSION

We build DiStefano, a DCTLS protocol that generates private commitments to encrypted TLS 1.3 data. We use a modular, standalone security framework that provides malicious security, and guarantees privacy for client browsing patterns amongst pre-approved servers. We provide an open-source integration in BoringSSL, and demonstrate the online efficiency of DiStefano for believable workloads.[14]

ACKNOWLEDGEMENT

We would like to thank Xiao Wang for his help with the various aspects of EMP and for sharing an initial integration of Ferret; Jonathan Hoyland, Yashvanth Kondi and Peter Scholl for useful conversations; Benjamin Livshits for discussing interesting applications for this work. Large parts of this work were completed while all of the authors were affiliated with Brave Software. Alex Davidson's work was supported by NOVA LINCS via the grants with reference codes UIDB/04516/2020 (DOI: 10.54499/UIDB/04516/2020) and UIDP/04516/2020 (DOI: 10.54499/UIDP/04516/2020), and by the financial support of FCT.IP. Joe Rowell was supported by EPSRC grant EP/P009301/1.

[12] For completeness, we benchmarked the cost of running the KeyUpdate operation in a garbled circuit, see Table VI.

[13] Indeed, such a system is already used for the Brave Browser.

[14] https://github.com/brave-experiments/DiStefano

References

[1] E. Rescorla, "The transport layer security protocol version 1.3," Internet Requests for Comments, RFC Editor, RFC 8446, August 2018.

[2] F. Zhang, D. Maram, H. Malvai, S. Goldfeder, and A. Juels, "DECO: Liberating web data using decentralized oracles for TLS," in *ACM CCS 2020*, J. Ligatti, X. Ou, J. Katz, and G. Vigna, Eds. ACM Press, Nov. 2020, pp. 1919–1938.

[3] M. Rosenberg, J. White, C. Garman, and I. Miers, "`zk-creds`: Flexible anonymous credentials from zkSNARKs and existing identity infrastructure," Cryptology ePrint Archive, Report 2022/878, 2022. [Online]. Available: https://eprint.iacr.org/2022/878

[4] J. T. Cross, "Age verification in the 21st century: Swiping away your privacy," *J. Marshall J. Computer & Info. L.*, vol. 23, p. 363, 2004.

[5] B. Szoka and A. D. Thierer, "Coppa 2.0: The new battle over privacy, age verification, online safety & free speech," *Progress & Freedom Foundation Progress on Point Paper No*, vol. 16, 2009.

[6] M. Yar, "Protecting children from internet pornography? a critical assessment of statutory age verification and its enforcement in the uk," *Policing: An International Journal*, vol. 43, no. 1, pp. 183–197, 2020.

[7] P. Blake, "Age verification for online porn: more harm than good?" *Porn Studies*, vol. 6, no. 2, pp. 228–237, 2019.

[8] R. S. Lear and J. D. Reynolds, "Your social security number or your life: Disclosure of personal identification information by military personnel and the compromise of privacy and national security," *BU Int'l LJ*, vol. 21, p. 1, 2003.

[9] J. J. Darrow and S. D. Lichtenstein, "Do you really need my social security number-data collection practices in the digital age," *NCJL & Tech.*, vol. 10, p. 1, 2008.

[10] European Commission, "GDPR: Right to Portability, Art. 20," https://gdpr-info.eu/art-20-gdpr/. Accessed 5th September 2023., 2014.

[11] A. Guy, M. Sporny, D. Reed, and M. Sabadello, "Decentralized identifiers (DIDs) v1.0," W3C, W3C Recommendation, Jul. 2022, https://www.w3.org/TR/2022/REC-did-core-20220719/.

[12] P. Team, "PageSigner: One-click website auditing," Website, 2023, https://old.tlsnotary.org/pagesigner. Accessed 04/04/2023.

[13] F. Zhang, E. Cecchetti, K. Croman, A. Juels, and E. Shi, "Town crier: An authenticated data feed for smart contracts," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 270–282. [Online]. Available: https://doi.org/10.1145/2976749.2978326

[14] X. Xie, K. Yang, X. Wang, and Y. Yu, "Lightweight authentication of web data via garble-then-prove," Cryptology ePrint Archive, Report 2023/964, 2023. [Online]. Available: https://eprint.iacr.org/2023/964

[15] J. Lauinger, J. Ernstberger, A. Finkenzeller, and S. Steinhorst, "Janus: Fast privacy-preserving data provenance for tls 1.3," Cryptology ePrint Archive, Paper 2023/1377, 2023, https://eprint.iacr.org/2023/1377. [Online]. Available: https://eprint.iacr.org/2023/1377

[16] P. Grubbs, A. Arun, Y. Zhang, J. Bonneau, and M. Walfish, "Zero-knowledge middleboxes," in *USENIX Security 2022*, K. R. B. Butler and K. Thomas, Eds. USENIX Association, Aug. 2022, pp. 4255–4272.

[17] D. Abram, I. Damgård, P. Scholl, and S. Trieflinger, "Oblivious TLS via multi-party computation," in *CT-RSA 2021*, ser. LNCS, K. G. Paterson, Ed., vol. 12704. Springer, Cham, May 2021, pp. 51–74.

[18] S. Tan, W. Chen, R. Deng, and R. A. Popa, "MPCAuth: Multi-factor authentication for distributed-trust systems," in *2023 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, May 2023, pp. 829–847.

[19] H. Lee, D. Kim, and Y. Kwon, "Tls 1.3 in practice:how tls 1.3 contributes to the internet," in *Proceedings of the Web Conference 2021*, ser. WWW '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 70–79. [Online]. Available: https://doi.org/10.1145/3442381.3450057

[20] Cloudflare, "TLS 1.2 vs. TLS 1.3 vs. QUIC: Distribution of secure traffic by protocol," 2023, accessed 11/04/2023. [Online]. Available: https://radar.cloudflare.com/adoption-and-usage#tls-1-2-vs-tls-1-3-vs-quic

[21] D. Tymokhanov and O. Shlomovits, "Alpha-rays: Key extraction attacks on threshold ecdsa implementations," Cryptology ePrint Archive, Paper 2021/1621, 2021, https://eprint.iacr.org/2021/1621. [Online]. Available: https://eprint.iacr.org/2021/1621

[22] N. Makriyannis and U. Peled, "A note on the security of gg18," 2021, https://info.fireblocks.com/hubfs/A_Note_on_the_Security_of_GG.pdf. [Online]. Available: https://info.fireblocks.com/hubfs/A_Note_on_the_Security_of_GG.pdf

[23] X. Wang, A. J. Malozemoff, and J. Katz, "Emp-toolkit: Efficient multiparty computation toolkit," https://github.com/emp-toolkit, 2016.

[24] IBM, "Handshake timer," 2023, https://www.ibm.com/docs/en/zos/3.1.0?topic=considerations-handshake-timer. Accessed 06/02/24.

[25] D. Adrian, K. Bhargavan, Z. Durumeric, P. Gaudry, M. Green, J. A. Halderman, N. Heninger, D. Springall, E. Thomé, L. Valenta, B. VanderSloot, E. Wustrow, S. Zanella-Béguelin, and P. Zimmermann, "Imperfect Forward Secrecy: How Diffie-Hellman Fails in Practice," in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '15. New York, NY, USA: Association for Computing Machinery, 2015, pp. 5–17.

[26] K. Arai and S. Matsuo, "Formal verification of TLS 1.3 full handshake protocol using proverif (Draft-11)," IETF TLS mailing list, 2016. [Online]. Available: https://mailarchive.ietf.org/arch/msg/tls/NXGYUUXCD2b9WwBRWbvrccjjdyI

[27] N. Aviram, S. Schinzel, J. Somorovsky, N. Heninger, M. Dankel, J. Steube, L. Valenta, D. Adrian, J. A. Halderman, V. Dukhovni, E. Käsper, S. Cohney, S. Engels, C. Paar, and Y. Shavitt, "DROWN: Breaking TLS using SSLv2," in *25th USENIX Security Symposium (USENIX Security 16)*. Austin, TX: USENIX Association, August 2016, pp. 689–706. [Online]. Available: https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/aviram

[28] B. Beurdouche, K. Bhargavan, A. Delignat-Lavaud, C. Fournet, M. Kohlweiss, A. Pironti, P.-Y. Strub, and J. K. Zinzindohoue, "A messy state of the union: Taming the composite state machines of tls," in *2015 IEEE Symposium on Security and Privacy*, 2015, pp. 535–552.

[29] B. Dowling, M. Fischlin, F. Günther, and D. Stebila, "A cryptographic analysis of the TLS 1.3 handshake protocol," *Journal of Cryptology*, vol. 34, no. 4, p. 37, Oct. 2021.

[30] S. Celi, A. Davidson, H. Haddadi, G. Pestana, and J. Rowell, "DiStefano: Decentralized infrastructure for sharing trusted encrypted facts and nothing more," Cryptology ePrint Archive, Report 2023/1063, 2023. [Online]. Available: https://eprint.iacr.org/2023/1063

[31] R. Holz, J. Hiller, J. Amann, A. Razaghpanah, T. Jost, N. Vallina-Rodriguez, and O. Hohlfeld, "Tracking the deployment of tls 1.3 on the web: A story of experimentation and centralization," *SIGCOMM Comput. Commun. Rev.*, vol. 50, no. 3, p. 3–15, jul 2020. [Online]. Available: https://doi.org/10.1145/3411740.3411742

[32] T. Team, "TLSNotary: Proof of data authenticity," 2023, https://tlsnotary.github.io/landing-page/. Accessed 04/04/2023.

[33] J. V. Bulck, M. Minkin, O. Weisse, D. Genkin, B. Kasikci, F. Piessens, M. Silberstein, T. F. Wenisch, Y. Yarom, and R. Strackx, "Foreshadow: Extracting the keys to the intel SGX kingdom with transient Out-of-Order execution," in *27th USENIX Security Symposium (USENIX Security 18)*. Baltimore, MD: USENIX Association, Aug. 2018, p. 991–1008. [Online]. Available: https://www.usenix.org/conference/usenixsecurity18/presentation/bulck

[34] H. Ritzdorf, K. Wüst, A. Gervais, G. Felley, and S. Capkun, "TLS-N: Non-repudiation over TLS enablign ubiquitous content signing," in *NDSS 2018*. The Internet Society, Feb. 2018.

[35] A. Backman, J. Richer, and M. Sporny, "Signing http messages," IETF draft, accessed 14/11/2022. [Online]. Available: https://www.ietf.org/archive/id/draft-ietf-httpbis-message-signatures-04.html

[36] D. Ott, K. Paterson, and D. Moreau, "Where is the research on cryptographic transition and agility?" *Commun. ACM*, vol. 66, no. 4, p. 29–32, mar 2023. [Online]. Available: https://doi.org/10.1145/3567825

[37] K. Yang, C. Weng, X. Lan, J. Zhang, and X. Wang, "Ferret: Fast extension for correlated OT with small communication," in *ACM CCS 2020*, J. Ligatti, X. Ou, J. Katz, and G. Vigna, Eds. ACM Press, Nov. 2020, pp. 1607–1626.

[38] M. Rosulek and L. Roy, "Three halves make a whole? Beating the half-gates lower bound for garbled circuits," in *CRYPTO 2021, Part I*, ser. LNCS, T. Malkin and C. Peikert, Eds., vol. 12825. Virtual Event: Springer, Cham, Aug. 2021, pp. 94–124.

[39] M. B. Santos, "Peco: methods to enhance the privacy of deco protocol," Cryptology ePrint Archive, Paper 2022/1774, 2022, https://eprint.iacr.org/2022/1774. [Online]. Available: https://eprint.iacr.org/2022/1774

[40] K. Y. Chan, H. Cui, and T. H. Yuen, "Dido: Data provenance from restricted tls 1.3 websites," Cryptology ePrint Archive, Paper 2023/1056, 2023, https://eprint.iacr.org/2023/1056. [Online]. Available: https://eprint.iacr.org/2023/1056

[41] Y. Lindell and B. Pinkas, "Secure multiparty computation for privacy-preserving data mining," Cryptology ePrint Archive, Report 2008/197, 2008. [Online]. Available: https://eprint.iacr.org/2008/197

[42] A. C. Yao, "Protocols for secure computations," in *23rd Annual Symposium on Foundations of Computer Science (sfcs 1982)*, 1982, pp. 160–164.

[43] O. Goldreich, S. Micali, and A. Wigderson, "Proofs that yield nothing but their validity and a methodology of cryptographic protocol design (extended abstract)," in *27th FOCS*. IEEE Computer Society Press, Oct. 1986, pp. 174–187.

[44] I. Damgård, V. Pastro, N. P. Smart, and S. Zakarias, "Multiparty computation from somewhat homomorphic encryption," in *CRYPTO 2012*, ser. LNCS, R. Safavi-Naini and R. Canetti, Eds., vol. 7417. Springer, Berlin, Heidelberg, Aug. 2012, pp. 643–662.

[45] M. Keller, "MP-SPDZ: A versatile framework for multi-party computation," in *ACM CCS 2020*, J. Ligatti, X. Ou, J. Katz, and G. Vigna, Eds. ACM Press, Nov. 2020, pp. 1575–1590.

[46] M. Keller, E. Orsini, and P. Scholl, "MASCOT: Faster malicious arithmetic secure computation with oblivious transfer," in *ACM CCS 2016*, E. R. Weippl, S. Katzenbeisser, C. Kruegel, A. C. Myers, and S. Halevi, Eds. ACM Press, Oct. 2016, pp. 830–842.

[47] D. Beaver, "Efficient multiparty protocols using circuit randomization," in *CRYPTO'91*, ser. LNCS, J. Feigenbaum, Ed., vol. 576. Springer, Berlin, Heidelberg, Aug. 1992, pp. 420–432.

[48] V. Kolesnikov and T. Schneider, "Improved garbled circuit: Free XOR gates and applications," in *ICALP 2008, Part II*, ser. LNCS, L. Aceto, I. Damgård, L. A. Goldberg, M. M. Halldórsson, A. Ingólfsdóttir, and I. Walukiewicz, Eds., vol. 5126. Springer, Berlin, Heidelberg, Jul. 2008, pp. 486–498.

[49] X. Wang, S. Ranellucci, and J. Katz, "Authenticated garbling and efficient maliciously secure two-party computation," in *ACM CCS 2017*, B. M. Thuraisingham, D. Evans, T. Malkin, and D. Xu, Eds. ACM Press, Oct. / Nov. 2017, pp. 21–37.

[50] Y. Ishai, J. Kilian, K. Nissim, and E. Petrank, "Extending oblivious transfers efficiently," in *CRYPTO 2003*, ser. LNCS, D. Boneh, Ed., vol. 2729. Springer, Berlin, Heidelberg, Aug. 2003, pp. 145–161.

[51] M. Keller, E. Orsini, and P. Scholl, "Actively secure OT extension with optimal overhead," in *CRYPTO 2015, Part I*, ser. LNCS, R. Gennaro and M. J. B. Robshaw, Eds., vol. 9215. Springer, Berlin, Heidelberg, Aug. 2015, pp. 724–741.

[52] C. Guo, J. Katz, X. Wang, and Y. Yu, "Efficient and secure multiparty computation from fixed-key block ciphers," in *2020 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, May 2020, pp. 825–841.

[53] R. Gennaro and S. Goldfeder, "Fast multiparty threshold ECDSA with fast trustless setup," in *ACM CCS 2018*, D. Lie, M. Mannan, M. Backes, and X. Wang, Eds. ACM Press, Oct. 2018, pp. 1179–1194.

[54] P. Paillier, "Public-key cryptosystems based on composite degree residuosity classes," in *EUROCRYPT'99*, ser. LNCS, J. Stern, Ed., vol. 1592. Springer, Berlin, Heidelberg, May 1999, pp. 223–238.

[55] H. Xue, M. H. Au, X. Xie, T. H. Yuen, and H. Cui, "Efficient online-friendly two-party ECDSA signature," in *ACM CCS 2021*, G. Vigna and E. Shi, Eds. ACM Press, Nov. 2021, pp. 558–573.

[56] I. Haitner, N. Makriyannis, S. Ranellucci, and E. Tsfadia, "Highly efficient OT-based multiplication protocols," in *EUROCRYPT 2022, Part I*, ser. LNCS, O. Dunkelman and S. Dziembowski, Eds., vol. 13275. Springer, Cham, May / Jun. 2022, pp. 180–209.

[57] J. Doerner, Y. Kondi, E. Lee, and a. shelat, "Secure two-party threshold ECDSA from ECDSA assumptions," in *2018 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, May 2018, pp. 980–997.

[58] ——, "Threshold ECDSA from ECDSA assumptions: The multiparty case," in *2019 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, May 2019, pp. 1051–1066.

[59] R. Impagliazzo and M. Naor, "Efficient cryptographic schemes provably as secure as subset sum," *Journal of Cryptology*, vol. 9, no. 4, pp. 199–216, Sep. 1996.

[60] J. Groth and M. Kohlweiss, "One-out-of-many proofs: Or how to leak a secret and spend a coin," in *EUROCRYPT 2015, Part II*, ser. LNCS, E. Oswald and M. Fischlin, Eds., vol. 9057. Springer, Berlin, Heidelberg, Apr. 2015, pp. 253–280.

[61] S. Celi, S. Levin, and J. Rowell, "Cdls: Proving knowledge of committed discrete logarithms with soundness," Cryptology ePrint Archive, Paper 2023/1595, 2023, https://eprint.iacr.org/2023/1595. [Online]. Available: https://eprint.iacr.org/2023/1595

[62] A. Faz-Hernández, W. Ladd, and D. Maram, "ZKAttest: Ring and group signatures for existing ECDSA keys," in *SAC 2021*, ser. LNCS, R. AlTawy and A. Hülsing, Eds., vol. 13203. Springer, Cham, Sep. / Oct. 2022, pp. 68–83.

[63] D. McGrew, "An interface and algorithms for authenticated encryption," Internet Requests for Comments, RFC Editor, RFC 5116, January 2008.

[64] P. Grubbs, J. Lu, and T. Ristenpart, "Message franking via committing authenticated encryption," in *CRYPTO 2017, Part III*, ser. LNCS, J. Katz and H. Shacham, Eds., vol. 10403. Springer, Cham, Aug. 2017, pp. 66–97.

[65] M. R. Albrecht, C. Rechberger, T. Schneider, T. Tiessen, and M. Zohner, "Ciphers for MPC and FHE," in *EUROCRYPT 2015, Part I*, ser. LNCS, E. Oswald and M. Fischlin, Eds., vol. 9056. Springer, Berlin, Heidelberg, Apr. 2015, pp. 430–454.

[66] N. Agrawal, J. Bell, A. Gascón, and M. J. Kusner, "MPC-friendly commitments for publicly verifiable covert security," in *ACM CCS 2021*, G. Vigna and E. Shi, Eds. ACM Press, Nov. 2021, pp. 2685–2704.

[67] N. T. Blog, "How netflix brings safer and faster streaming experiences to the living room on crowded networks using tls 1.3," https://tinyurl.com/mrydxrsw, 2020.

[68] M. Silverlock and G. Redner, "Bringing modern transport security to google cloud with tls 1.3," https://cloud.google.com/blog/products/networking/tls-1-3-is-now-on-by-default-for-google-cloud-services, 2020.

[69] I. Grigorik, *High Performance Browser Networking: What every web developer should know about networking and browser performance*. O'Reilly Media, Incorporated, 2013. [Online]. Available: https://books.google.pt/books?id=X_SsMQEACAAJ

[70] S. Gueron and M. E. Konavis, "Intel® carry-less multiplication instruction and its usage for computing the gcm mode," 2014, accessed 14/03/2023. [Online]. Available: https://www.intel.com/content/dam/develop/external/us/en/documents/clmul-wp-rev-2-02-2014-04-20.pdf

[71] R. Canetti, "Universally composable security: A new paradigm for cryptographic protocols," in *42nd FOCS*. IEEE Computer Society Press, Oct. 2001, pp. 136–145.

[72] CloudPing, "Aws latency monitoring," https://www.cloudping.co/grid/p_25/timeframe/1D, 2024.

[73] L. Tulloch and I. Goldberg, "Lox: Protecting the social graph in bridge distribution," *Privacy Enhancing Technologies*, vol. 2023, no. 1, 2023. [Online]. Available: https://petsymposium.org/popets/2023/popets-2023-0029.pdf

[74] L. N. R. Seny Kamara, "Cryptography for grassroot organising," https://iacr.org/submit/files/slides/2023/rwc/rwc2023/69/slides.pdf, 2023.

[75] G. Awwad and K. Toyama, "Digital repression in palestine," in *CHI*. ACM, 2024. [Online]. Available: https://dl.acm.org/doi/pdf/10.1145/3613904.3642422

[76] D. Xue, A. Ablove, R. Ramesh, G. K. Danciu, and R. Ensafi, "Bridging barriers: A survey of challenges and priorities in the censorship circumvention landscape," in *USENIX Security Symposium*. USENIX, 2024. [Online]. Available: https://www.usenix.org/system/files/usenixsecurity24-xue-bridging.pdf

[77] M. Naor and M. Yung, "Public-key cryptosystems provably secure against chosen ciphertext attacks," in *22nd ACM STOC*. ACM Press, May 1990, pp. 427–437.

[78] T. Iwata, K. Ohashi, and K. Minematsu, "Breaking and repairing GCM security proofs," in *CRYPTO 2012*, ser. LNCS, R. Safavi-Naini and R. Canetti, Eds., vol. 7417. Springer, Berlin, Heidelberg, Aug. 2012, pp. 31–49.

[79] R. L. Rivest, A. Shamir, and Y. Tauman, "How to leak a secret," in *ASIACRYPT 2001*, ser. LNCS, C. Boyd, Ed., vol. 2248. Springer, Berlin, Heidelberg, Dec. 2001, pp. 552–565.

[80] K. Q. Nguyen, F. Bao, Y. Mu, and V. Varadharajan, "Zero-knowledge proofs of possession of digital signatures and its applications," in *ICICS 99*, ser. LNCS, V. Varadharajan and Y. Mu, Eds., vol. 1726. Springer, Berlin, Heidelberg, Nov. 1999, pp. 103–118.

[81] S. R. Department, "Global number of customers at the largest banks in the united kingdom (uk) in 2022," Statista, Aug 29 2023, http://tinyurl.com/statista-banks.

Appendix A
Additional Cryptographic Preliminaries

We provide some additional cryptographic preliminaries that are required for arguing the security of our system.

A. Commitment Schemes

Definition 3 (Commitment scheme)**.** *A commitment scheme* Γ *is a tuple consisting of the following algorithms:*

- $\Gamma.\mathsf{Gen}(1^\lambda)$*: outputs some secret parameters* sp*;*
- $\Gamma.\mathsf{Commit}(\mathsf{sp}, x)$*: outputs a commitment* c*;*
- $\Gamma.\mathsf{Challenge}(c)$*: outputs a random challenge* t*;*
- $\Gamma.\mathsf{Open}(\mathsf{sp}, c, t, x)$*: outputs a bit* $b \in \{0, 1\}$.

An interactive commitment scheme, $\widetilde{\Gamma}$, between a committer, $\mathcal{C}$, and a revealer, $\mathcal{R}$, proceeds as follows:

- $\mathcal{C}$ runs $\mathsf{sp} \leftarrow \Gamma.\mathsf{Gen}(1^\lambda)$, and sends $c \leftarrow \Gamma.\mathsf{Commit}(\mathsf{sp}, x)$ to $\mathcal{R}$;
- $\mathcal{R}$ sends $t \leftarrow \Gamma.\mathsf{Challenge}(c)$ to $\mathcal{C}$;
- $\mathcal{C}$ sends x to $\mathcal{R}$;
- $\mathcal{R}$ outputs $b \overset{?}{=} 1$, for $b \leftarrow \Gamma.\mathsf{Open}(\mathsf{sp}, c, t, x)$.

We show in Section C-B that AES-GCM ciphertext commitment scheme in Section V-A is perfectly binding and computationally hiding for TLS 1.3 encrypted data. A high-level overview of the commitment phase based on this scheme is given in Section IV-C.

B. Authenticated Encryption

An authenticated encryption with associated data (AEAD) scheme considers a keyspace $\mathcal{K}$, a message space $\mathcal{M}$, a ciphertext space $\mathcal{X}$, and a tag space $\mathcal{T}$, and is defined using the following algorithms.

- $k \leftarrow \mathsf{AEAD}.\mathsf{keygen}(1^\lambda)$: Outputs a key $k \leftarrow\!\!\$\, \mathcal{K}$.
- $(C, \tau) \leftarrow \mathsf{AEAD}.\mathsf{Enc}(k, m; A)$: For a key $k \in \mathcal{K}$, message $m \in \mathcal{M}$, and associated data $A \in \{0, 1\}^*$, outputs a ciphertext $C \in \mathcal{X}$ and a tag $\tau \in \mathcal{T}$.
- $m \vee \bot \leftarrow \mathsf{AEAD}.\mathsf{Dec}(k, C, \tau; A)$: For a key $k \in \mathcal{K}$, ciphertext $C \in \mathcal{M}$, tag $\tau \in \mathcal{T}$, and associated data $A \in \{0, 1\}^*$, outputs a message $m \in \mathcal{M}$ or $\bot$.

Any AEAD scheme must satisfy the following guarantees.

Definition 4 (Correctness)**.** AEAD *is correct if and only if the following holds true.*

$$\Pr\left[m \leftarrow \mathsf{AEAD}.\mathsf{Dec}(k, C, \tau; A) \middle| \substack{k \leftarrow \mathsf{AEAD}.\mathsf{keygen}(1^\lambda) \\ (C,\tau) \leftarrow \mathsf{AEAD}.\mathsf{Enc}(k,m;A)}\right] = 1$$

Definition 5 (Security)**.** *An* AEAD *scheme is secure if it satisfies the IND-CCA notion of security [77].*

It is widely known that the AES-GCM block cipher mode of operation satisfies these guarantees [78], where $\mathcal{K} = \{0, 1\}^\lambda$, $\mathcal{M} = \{0, 1\}^*$, $\mathcal{C} = \{0, 1\}^*$. In other words, it can tolerate messages of arbitrary length and produce ciphertexts accordingly.

C. Zero-knowledge Signature Verification

We require a scheme for constructing zero-knowledge proofs of knowledge of valid signatures (ZKPVS) of signatures produced during TLS exchanges. Such a scheme considers a prover and a verifier, where the prover holds a valid signature σ issued by a keypair sk, vk, and the verifier holds a list $R = \{\mathsf{vk}_i\}_{i \in [m]}$ of all valid public verification keys, where $\mathsf{vk} \in R$. Previous work has produced practical schemes for proving knowledge of ECDSA signatures (e.g. see ZKAttest [62] and CDLS [61]), noting their similarity to ring signatures [79], in particular. Similar approaches for other TLS-compliant signature schemes (e.g. based on RSA) exist [80], but do not appear to be practical for our application (though practical constructions would have immediate value for our work).

While some previous work refers to the zero-knowledge functionality that we require as ring signature schemes [60]–[62], we note that the functionality differs in that the eventual proofs are constructed over standard signatures (by non-signing entities). We give a detailed formalisation of the required functionality in the full version [30].

Instantiations. As mentioned above, it is possible to instantiate the required functionality with a specific proof scheme that generates signatures under ECDSA private keys that preserve anonymity amongst a set of known ECDSA verification keys (e.g. see [60]–[62]). This means that we can directly instantiate our DCTLS protocol for servers using ECDSA signing. Supporting TLS signatures of other types requires practical instantiations of ZKPVS schemes for the specific signing method.

Server anonymity. As noted throughout, we introduce the possibility for $\mathcal{C}$ to prove to $\mathcal{V}$ that the communicating TLS server, $\mathcal{S}$, belongs to a pre-approved set, R, of server identities. This list of identities can be constructed simply from a list of TLS certificates, and uses a compliant ZKPVS scheme (Section A-C) for generating proofs of valid TLS signatures. We note here that anonymity is only preserved amongst the set R, and is highly application-specific: if R only contains a single identity, then anonymity is not guaranteed. However, there are various applications where R is likely to be non-trivial, such as in the case where $\mathcal{C}$ would like to generate a commitment to a bank balance that is in a range. In principle, our approach would allow generating commitments to balances provided by any of a pre-approved list of banks. In doing so, this would preserve $\mathcal{C}$'s privacy by hiding the identity of the bank they have an account with. Similar mechanisms can be built for generating commitments with respect to governmental identities (e.g. social security statuses associated with EU member states). While targeting any given applications are beyond the scope of this work, we believe that the provisioning of the capability for generating such proofs (which previous works do not provide) can provide meaningful privacy enhancements for clients. Note also that a larger R will have an impact on the performance of the ZKPVS scheme employed, since their complexity is typically dependent on $|R|$. Nonetheless, we believe that even a modest value (*e.g.* $|R| = 10$) would grant meaningful privacy protection. As a concrete example, banking sectors are typically highly consolidated — in the UK, there are four banks that hold the majority of customer accounts [81] — . Providing privacy for this "small" sets can be meaningful for many clients.

Appendix B
DCTLS Formal Description

The three phases (HSP, QP, CP) of a generic three-party TLS (DCTLS) protocol are formally described (in terms of their inputs and outputs) below.

- $(\mathsf{pp}, \mathsf{sp}_{\mathcal{C}}, \mathsf{sp}_{\mathcal{S}}, \mathsf{sp}_{\mathcal{V}}) \leftarrow \mathsf{DCTLS.HSP}(1^\lambda)$: The handshake phase takes as input a security parameter, and computes a TLS handshake between $\mathcal{S}$, and an effective client that consists of both $\mathcal{C}$ and $\mathcal{V}$. The public/secret parameters $(\mathsf{pp}, \mathsf{sp}_{\mathcal{S}})$ learnt by $\mathcal{S}$ are the same as in a standard TLS handshake. The secret parameters learned by $\mathcal{C}$ ($\mathsf{sp}_{\mathcal{C}}$) and $\mathcal{V}$ ($\mathsf{sp}_{\mathcal{V}}$) are shares of the secret parameters learnt by a standard TLS client [29], so that neither party can compute encrypted traffic alone.
- $(\mathsf{r}, \hat{\mathsf{q}}, \hat{\mathsf{r}}) \leftarrow \mathsf{DCTLS.QP}(\mathsf{pp}, \mathsf{sp}_{\mathcal{C}}, \mathsf{sp}_{\mathcal{S}}, \mathsf{sp}_{\mathcal{V}}, \mathsf{q})$: The query phase takes the public and secret parameters of each party as input, along with a query, q, that is to be sent to $\mathcal{S}$. This phase requires $\mathcal{S}$ to construct a response, r, to q and return it to $\mathcal{C}$. The phase outputs both q and r, and also vectors of TLS ciphertexts ($\hat{\mathsf{q}}$ and $\hat{\mathsf{r}}$) that encrypt the client queries and the server responses. $\hat{\mathsf{q}}$ and $\hat{\mathsf{r}}$ are vectors containing blocks of the TLS ciphertext encrypting q and r, respectively.
- $b \leftarrow \mathsf{DCTLS.CP}(\mathsf{pp}, \mathsf{sp}_{\mathcal{C}}, \mathsf{sp}_{\mathcal{V}}, \mathsf{q}, \mathsf{r}, \hat{\mathsf{q}}, \hat{\mathsf{r}}, (i, j))$: The commitment phase outputs a bit b, where $b = 1$ if $\mathcal{C}$ constructs a valid opening of $\hat{\mathsf{q}}_i$ and $\hat{\mathsf{r}}_j$ with respect to the unencrypted q and r. Broadly speaking, $\mathcal{C}$ sends to $\mathcal{V}$ the TLS-encrypted ciphertexts, before $\mathcal{V}$ sends $\mathsf{sp}_{\mathcal{V}}$ to $\mathcal{C}$, and then $\mathcal{C}$ opens the commitments. Note that a valid opening could be proving in zero-knowledge that $\hat{\mathsf{r}}_j$ encrypts a value in a given range, or using 2PC to decrypt the block directly.

Appendix C
Security of AES-GCM Optimisations

A. Secure 2PC Encryption & Decryption

2PC functionalities. We consider the 2PC functionalities for encryption and decryption in 2PC-AES-GCM as given in Algorithm 1 and Algorithm 2, respectively. Our ideal functionality also covers the nonce uniqueness requirement of AES-GCM. We note that in practice these additional checks do not seem to affect the running time by much: for example, our prototype garbled circuit implementation only requires around 768 extra AND gates, representing around a 10% increase over an AES circuit.

Security argument. We now argue the security of computing encryptions and decryptions with respect to the ideal functionalities described in the full version [30]. We implicitly assume that 2PC evaluations of the polynomial P and the AES functionality (using garbled circuits) are secure with respect to malicious adversaries, and that AES-GCM is a secure AEAD scheme. These security guarantees are assumed in previous work [2], [17], [18], but are not made explicit. We require them when proving that the query phase of DiStefano is secure.

Algorithm 1 2PC-AES-GCM Encrypt

```
Require: $k = k_c + k_v, IV_c, IV_v, \{h_c^i\}_{i\in[n]}, \{h_v^i\}_{i\in[n]}$
Require: $\mathcal{C}$ inputs a message $M$
Require: $IV_c$ and $IV_v$ must not have been supplied for encryption previously.
Ensure: $\mathcal{C}$ learns $((C_1, \ldots, C_n), \tau(A, C, k, IV))$.
Ensure: $\mathcal{V}$ learns $(C_1, \ldots, C_n)$.
  if $IV_c \neq IV_v$ then
      return Error                         ▷ The IVs must match.
  end if
  Parse $M$ as $M_1 \| \ldots \| M_n$          ▷ $M_i$ fits AES blocksize
  $C = (C_i \leftarrow \mathsf{AES.Enc}(k_c + k_v, IV_c + i) \oplus M_i)_{i\in[n]}$
  $\tau_c \leftarrow P_{A||C||\mathsf{len}(A)||\mathsf{len}(C)}(\{h_c^i\})$
  $\tau_v \leftarrow P_{A||C||\mathsf{len}(A)||\mathsf{len}(C)}(\{h_v^i\})$
  $\tau \leftarrow \tau_c \oplus \tau_v \oplus \mathsf{AES.Enc}(k_c + k_v, IV_c)$
  return $(C, \tau)$ to $\mathcal{C}$
  return $C$ to $\mathcal{V}$
```

Algorithm 2 2PC-AES-GCM Decrypt

```
Require: $k = k_c + k_v, IV_c, IV_v, \{h_c^i\}_{i\in[n]}, \{h_v^i\}_{i\in[n]}$
Require: $\mathcal{C}$ inputs a set of $n$ masks $\{b_i\}$ and secret parameters sp for a computationally binding and perfect hiding commitment scheme $\Gamma'$, and $\mathcal{V}$ inputs the corresponding commitments $\{d_i\}$ that were sent by $\mathcal{C}$, generated using $\Gamma'$, and ephemeral challenges $\{t_i'\}$.
Require: $\mathcal{C}$ and $\mathcal{V}$ jointly input a set of ciphertext blocks $C_1, \ldots, C_n$ and a tag $\tau(A, C, k, IV)$.
Require: $IV_c$ and $IV_v$ must not have been supplied for encryption previously.
Ensure: $\mathcal{C}$ learns $(M_1, \ldots, M_n)$.
Ensure: $\mathcal{V}$ learns $(E_1, \ldots, E_n)$.
  if $\exists\ i$ s.t. $0 \leftarrow \Gamma'.\mathsf{Open}(\mathsf{sp}', d_i, t_i', b_i)$
      return Error                    ▷ Commitment checks failed.
  end if then
  if $IV_c \neq IV_v$ then
      return Error                         ▷ The IVs must match.
  end if
  $\tau_c' \leftarrow P_{A||C||\mathsf{len}(A)||\mathsf{len}(C)}(\{h_c^i\})$
  $\tau_v' \leftarrow P_{A||C||\mathsf{len}(A)||\mathsf{len}(C)}(\{h_v^i\})$
  $\tau' = \tau_c' \oplus \tau_v' \oplus \mathsf{AES.Enc}(k_c + k_v, IV_c)$
  if $\tau' \neq \tau$ then
      return Error                              ▷ Invalid tag
  end if
  $(M_i = C_i \oplus \mathsf{AES.Enc}(k_c + k_v, IV_c + i))_{i\in[n]}$
  $(E_i = \mathsf{AES.Enc}(k_c + k_v, IV_c + i)_i \oplus b_i)_{i\in[n]}$
  return $(M_i)_{i\in[n]}$ to $\mathcal{C}$
  return $(E_i)_{i\in[n]}$ to $\mathcal{V}$
```

Lemma 6 (Malicious Client). *2PC-AES-GCM is secure in the* *presence of a malicious adversary that controls $\mathcal{C}$.*

Proof. Let S be a PPT simulator for the encryption functionality, that simply returns samples C' from the domain of $\mathsf{AES.Enc}$, and $\tau_c \leftarrow\$ \{0,1\}^t$, and returns (C, τ_c) to $\mathcal{C}$. We ultimately argue that the real-world outputs of 2PC-AES-GCM are indistinguishable from this.

Let $\mathsf{S}_{\mathsf{AES}}$ be a simulator for the ideal 2PC evaluation of $\mathsf{AES.Enc}$, and let S_P be a simulator for the ideal evaluation of P. It first sends m to $\mathsf{S}_{\mathsf{AES}}$ and learns $C = (C_1, \ldots, C_n)$. Then it sends C to S_P (along with A) and learns τ. It returns (C, τ) to $\mathcal{C}$. To see that this is indistinguishable from the real-world, we can trivially construct a hybrid argument from the real-world protocol that relies on two steps, replacing real garbled circuit evaluation of each functionality with ideal-world simulation, and argue security based on the maliciously-secure 2PC garbled circuit approach that we use (Section III).

Finally, based on the assumption that AES is a pseudorandom permutation, we can construct a final hybrid step, that replaces $\mathsf{AES.Enc}$ with a random value in the domain.[15]

The case of decryption is much simpler since the client only learns the message if they submit valid inputs to S (by the AEAD security guarantees of AES-GCM). This can be established using the same simulators $\mathsf{S}_{\mathsf{AES}}$ and S_P defined above. □

Lemma 7 (Malicious Verifier)**.** *2PC-AES-GCM is secure in the presence of a malicious adversary that controls $\mathcal{V}$.*

Proof. The proof for a malicious $\mathcal{V}$ follows the same structure as in the case of $\mathcal{C}$, but note that $\mathcal{V}$ is strictly less powerful, because the $\mathcal{V}$ does not submit a message to be encrypted. □

We briefly note that PageSigner follows a slightly different approach than this for computing tags: we decided not to follow their approach, as a "back-of-an-envelope" calculation suggests that it is strictly slower than the aforementioned approach. We discuss this in more detail in the full version of our work [30].

B. Commitment Scheme

We prove that the commitment scheme for AES-GCM ciphertexts presented in Section V-A is a perfectly hiding and computationally binding commitment scheme. Concretely, this means showing that the 2PC's Algorithm 2 produces computationally binding and perfectly hiding commitments E_i to $e_i = \mathsf{AES.Enc}(k, IV + i)$. Since the primary application of this work is to prove facts about traffic received from the server, we focus on only the AES Decrypt algorithm. If we wanted to prove facts about client-encrypted traffic, Algorithm 1 would have to be updated to also include commitments. As before, we implicitly assume that the 2PC evaluations of the polynomial P and the AES functionality are secure with respect to malicious adversaries.

First, let Γ' be a perfectly hiding, and computationally binding commitment scheme, generating commitments $d \in \{0,1\}^\lambda$ for arbitrary $x \in \{0,1\}^*$. Let $\mathcal{K}$ and $\mathcal{X}$ be the key and ciphertext spaces for AES-GCM, respectively. Then, let $\mathsf{sp}' \leftarrow \Gamma'.\mathsf{Gen}(1^\lambda)$, and $d_i = \Gamma'.\mathsf{Commit}(\mathsf{sp}', b_i)$ for some $b_i \leftarrow\$ \mathcal{X}$.

Lemma 8 (AES-GCM Commitment security)**.** *The algorithm 2PC-AES-GCM Decrypt, when instantiated with Γ', produces computationally binding and perfectly hiding commitments to decryptions of AES-GCM ciphertexts.*

Proof. We first formally describe the AES-GCM commitment (Γ_{AES}) scheme for any of the i^{th} message blocks, using the following functionality (applying the framework described in Section A-A).

- $\mathsf{sp} = (b_i, k_c)$, and assume that the receiver holds the commitment $d_i \leftarrow \Gamma'.\mathsf{Commit}(\mathsf{sp}', b_i)$.
- $E_i \leftarrow \Gamma_{\mathsf{AES}}.\mathsf{Commit}(\mathsf{sp}, M_i)$: Runs the 2PC-AES-GCM-Commit algorithm to generate commitments $(C_i, E_i = e_i + b_i)$, where C_i is the i^{th} received ciphertext, and $e_i = \mathsf{AES.Enc}(k_c + k_v, IV_c + i)$.
- $(k_v, t'_i) \leftarrow \Gamma_{\mathsf{AES}}.\mathsf{Challenge}(C_i, E_i)$: reveals $\mathcal{V}$'s key share, k_v, to the commitment sender, along with a challenge t'_i for Γ'.
- $\hat{b} \leftarrow\$ \Gamma_{\mathsf{AES}}.\mathsf{Open}(\mathsf{sp}, (C_i, E_i), k_v, M_i)$: First, checks that $1 \leftarrow \Gamma'.\mathsf{Open}(\mathsf{sp}', d_i, t'_i, b_i)$. Then, computes $b_i \oplus E_i$ to reveal e_i, and then returns 1 iff $M' \leftarrow C_i \oplus e_i$ satisfies $M' = M_i$.

To argue perfect hiding, notice that b_i is not revealed to $\mathcal{V}$ during the execution of 2PC-AES-GCM-Commit. As Γ' is a perfectly hiding commitment scheme, we may simply replace b_i with a uniformly random value r_i in the range of b_i, which in turn makes $E_i = r_i \oplus e_i$ a one-time pad encryption of e_i. By the properties of the one-time pad, we have that the scheme is therefore also a perfectly hiding commitment to $e_i = \mathsf{AES.Enc}(k_c + k_v, IV + i)$.

To argue computational binding, we first ensure that the masks b_i generated by the client are consistent with their input to the 2PC-AES-GCM-Commit algorithm by explicitly checking that they correspond to the verifier-known commitments. To argue security henceforth, we consider two possible events. In the first event, we consider a PPT adversary $\mathcal{B}'$ that can generate valid openings of d_i to $b' \neq b_i$ for Γ'. The advantage of $\mathcal{B}'$ is clearly bounded by the computational binding security of Γ'. In the second event, we assume that no such $\mathcal{B}'$ exists, and instead we consider a PPT adversary $\mathcal{B}$ that finds M', such that $M' = E_i \oplus b_i \oplus C_i$ for $M' \neq M_i$. Since the only free variable in this equation is $e_i = \mathsf{AES.Enc}(k_c + k_v, IV_c + i)$, this would require $\mathcal{B}$ to find $k' \neq k_c \oplus k_v$ such that $\mathsf{AES.Enc}(k', IV_c + i) = e_i$. Clearly, by the IND-CCA security of AES-GCM, finding such a $k' \in \mathcal{K}$ is computationally infeasible. Note that the lack of key-committing security does not play a role here: the adversary would need to freely manipulate the value of e_i to launch such an attack, which is impossible under the assumption that b_i is fixed.

□

[15]This only holds if the IV is a nonce, see [2, §B.2].