Oblivious TLS via Multi-Party Computation
Full Version

Damiano Abram[1], Ivan Damgård[1], Peter Scholl[1], and Sven Trieflinger[2]

[1] Aarhus University
{damiano.abram, ivan, peter.scholl}@cs.au.dk
[2] Robert Bosch GmbH
Sven.Trieflinger@de.bosch.com

Abstract. In this paper, we describe Oblivious TLS: an MPC protocol that we prove UC secure against a majority of actively corrupted parties. The protocol securely implements TLS 1.3. Thus, any party P who runs TLS can communicate securely with a set of servers running Oblivious TLS; P does not need to modify anything, or even be aware that MPC is used.
Applications of this include communication between servers who offer MPC services and clients, to allow the clients to easily and securely provide inputs or receive outputs. Also, an organization could use Oblivious TLS to improve in-house security while seamlessly connecting to external parties.
Our protocol runs in the preprocessing model, and we did a preliminary non-optimized implementation of the on-line phase. In this version, the hand-shake completes in about 1 second. Performance of the record protocol depends, of course, on the encryption scheme used. We designed an MPC friendly scheme which achieved a throughput of about 300 KB/sec. Based on implementation results from other work, the standard AES-GCM can be expected to be as fast, although our implementation did not do as well.

1 Introduction

Secure multi-party computation (MPC) allows a group of parties to jointly evaluate a function on private inputs, ensuring that no party learns anything more than what can be deduced from the output of the function. Developments in recent years have shown that MPC can be practical for a range of use-cases, and is starting to see real-world deployments. While the classic scenario for MPC involves a set of parties who each have a private input, more recent applications also focus on the setting where a set of *MPC servers*, called an *MPC engine*, are distributively performing a computation on inputs uploaded by clients and known to none of the servers. In this client-server setting, the computation may be outsourced by external parties who initially provided the inputs, or the servers may be part of a larger system which delegated a private computation to them. As in the regular MPC setting, there must be some intrinsic motivation or incentive for the parties operating the MPC servers not to collude. For instance, the

servers can be hosted by independent organisations which have interest in protecting the confidentiality of the clients' data against the other parties, perhaps because some form of representation is implemented.

The set of clients can be dynamic in such systems and clients should not need to participate in the actual MPC protocol, instead they should be able to send private input to the protocol using standardized algorithms, which is exactly what Oblivious TLS allows. This creates a more flexible overall system that more closely resembles the cloud-based IT system deployments that are common today. For instance, in [9], Damgård et al. describe a concrete instantiation of such a system. They propose a credit rating system that enables banks to benchmark their customers' confidential performance data against a large representative set of confidential performance data from a consultancy house. The authors anticipate that the MPC servers would be run by the consultancy house and the Danish Bankers Association in a commercial setting. Individual banks as clients learn nothing but the computed benchmarking score.

Security is typically maintained as long as not too many of the servers are corrupted; for instance, in the *dishonest majority* setting it is common to allow up to $n - 1$ out of n servers to be corrupted, while the *honest majority setting* relaxes this by assuming that more than half of the servers are honest.

The Transport Layer Security protocol (TLS) is the leading standard for secure communication over the Internet. TLS allows two parties, or *endpoints*, to first run a handshake protocol to establish a common key, and secondly, in the record layer protocol, to securely and authentically transmit information using the key. The latest version of the protocol is TLS 1.3, which has seen major design changes to address vulnerabilities in previous versions.

Contributions. In this work, based on a master's thesis [2], we study the problem of obliviously running one or more endpoints of the TLS 1.3 protocol, *inside an MPC engine*. We refer to this scheme as *Oblivious TLS*. The protocol allows the engine to securely communicate with any endpoint of the Internet that runs TLS, in a completely oblivious manner: the other endpoint does not need to be modified, nor even be aware of the fact that it is interacting with a multi-party computation (and likewise, the *second* endpoint may also be an Oblivious TLS instance, unbeknownst to the first).

The possibilities created by Oblivious TLS are manifold and potentially groundbreaking: Oblivious TLS facilitates the integration of MPC-based components into today's complex IT systems. For example, distributed key management systems based on MPC can be interfaced without time-consuming and sometimes infeasible client-side modifications. Workloads that, despite the impressive performance gains seen in recent years, are still outside the realm of the possible using MPC today, could be securely outsourced to external services protected by comparatively low overhead Trusted Execution Environments by seamlessly integrating with TLS-based remote attestation mechanisms. Another fascinating possibility is the use of Oblivious TLS in conjunction with *Distributed Autonomous Organizations* (DAO) that ensure the confidentiality of

data through the use of MPC. In the future, Oblivious TLS may enable DAOs to obliviously use external services to, for example, autonomously manage cloud resources required to conduct their business.

The MPC engine itself can be instantiated with a large class of standard, modern MPC protocols based on secret-sharing with arithmetic operations. We focus on instantiating this with actively secure MPC protocols based on information-theoretic MACs with security against a dishonest majority, such as SPDZ [11,10] and related protocols [23], however, our techniques are also applicable to other settings and honest majority protocols.

TLS 1.3 is notoriously complex, and running this inside MPC presents several technical challenges. We first give an overview of some of these challenges below, and then describe some further motivation for the problem of running TLS in MPC.

Multi-Party Diffie-Hellman. For the handshake protocol, we chose to run elliptic-curve Diffie-Hellman, currently the most popular key exchange method. Doing this inside MPC requires an exponentiation between a known public key and a secret exponent, where the output must remain secret. Moreover, the shared key (an elliptic curve point) must be represented in a suitable manner in the MPC engine to allow for further private computations. We present a new method for doing this based on *doubly-authenticated points*, namely, a way of reliably generating random elliptic curve points that are secret-shared both in a standard finite field MPC representation, and simultaneously in a specialized shared elliptic curve representation. This allows for efficient conversions between the two representations, and may be of independent interest.

We also present a more efficient variant, which avoids the use of doubly-authenticated points. This comes with the slight downside that we do not manage to securely realise the same key exchange functionality, since a corrupted MPC server in the oblivious endpoint can force the derived key to be shifted by an arbitrary amount. In practice, however, we argue that this weaker version of the functionality suffices to run TLS, since the shift to the key is completely harmless, unless the adversary already happened to know the private key of the other endpoint.

Threshold Signing. To authenticate the endpoints, we additionally need to run a threshold signing protocol. Here, the message to be signed (based on the TLS transcript) is public, so the only information secret-shared in the MPC engine is the signing key and signature randomness. For this, we use EdDSA Schnorr-based signatures, which allow a simple threshold protocol without any expensive MPC operations.

Record Layer Protocol. For the record layer, we need to run authenticated encryption inside MPC. For this, we present an approach based on the standard AES-GCM construction, and a more specialized approach based on a custom

MPC-friendly AEAD scheme. AES-GCM is quite well-suited to MPC, because of the linear structure of its Galois field MACs. The second approach is much more efficient, however, since it avoids doing any AES operations inside MPC. This comes at the cost of a small modification to the TLS specification, which also requires the endpoint to know the number of MPC servers involved in the computation.

Motivation and Related Work. As MPC becomes more widespread, it is natural to think not only about designing new and improved MPC protocols, but also about how these can be integrated into existing infrastructure. In particular, an MPC engine will typically not be a standalone piece, but rather a secure component of a larger system.

Whenever some private data passes in or out of the MPC component, this has to be done in a secure manner. With typical MPC protocols based on some form of secret sharing, the natural solution is to simply have the external process secret-share inputs to the MPC servers, and receive shares of outputs to be reconstructed. When active security is required, this is less straightforward since shares can be tampered with, although there are known methods and protocols that allow receiving inputs from, or sending outputs to, an external client in an authentic manner [9].

A drawback of these solutions is that they tend to be tailored to specific MPC protocols, meaning that all the clients and components of the system must be aware of the fact that MPC is taking place. This firstly has the potential security concern that it reveals that an MPC protocol is being carried out in the first place, and secondly, requires highly specialized software to implement.

In [16], this motivated the study of symmetric primitives such as PRFs, which are *MPC-friendly* (either by design, or by chance), meaning that they can be evaluated inside an MPC engine relatively cheaply. This was later extended to build MPC-friendly modes of operation for block ciphers [25].

While these works address the problem of encrypting data inside an MPC engine, they do not immediately lead to solutions for securely communicating with an external party, without either resorting to protocol-specific methods or other assumptions like pre-shared keys. DISE [3] also studied distributed forms of symmetric encryption including authenticated encryption. However, in their setting the message is always known to one party, which is not the case for us.

Finally, the recent and independent work DECO [27] presents a protocol that allows a TLS client to prove provenance of TLS data to a third party. Their solution is essentially based on a 2-party execution of a TLS client, with a very similar approach to the one described in this paper. Oblivious TLS is however applicable to both client and server sides of the connection and generalises to the multiparty case. Moreover, the multiparty Diffie-Hellman procedure presented in this paper is completely actively secure, whereas the solution proposed in [27] allows for influence of the adversary that may cause a Handshake failure.

Roadmap. We begin with some preliminaries in Section 2, where we explain notation, give an overview of TLS 1.3, and the MPC building blocks we use. In Section 3, we outline our solution, describing the general idea and presenting the main steps of the protocol. Section 4 covers the handshake layer of Oblivious TLS, where we focus on the method for generating doubly-authenticated points, the elliptic-curve Diffie-Hellman protocol and the signature generation. In Section 5, we describe the record layer. Then, in Section 6, we discuss security and performance of Oblivious TLS.

2 Notation and Preliminaries

We denote the finite field with p elements by $\mathbb{F}_p$, where p is a prime or prime power. Its multiplicative group is $\mathbb{F}_p^\times$. Sometimes, when the cardinality is not important for the discussion, we simply write $\mathbb{F}$. When dealing with groups, we represent the cyclic subgroup generated by an element g with $\langle g \rangle$.

For any string a, $\mathrm{len}(a)$ denotes the length of a, whereas $\mathrm{Trunc}(a, c)$ denotes the substring of the first c elements of a.

When dealing with bit sequences, we identify the sets $\{0,1\}^k$, $\mathbb{F}_2^k$ and $\mathbb{F}_{2^k}$ as different representations of the finite field with 2^k elements. For this reason, when multiplying two elements $a, b \in \{0,1\}^k$, we mean multiplication in $\mathbb{F}_{2^k}$. The set $\{0,1\}^*$ instead represents $\bigcup_{i=0}^{\infty} \{0,1\}^i$.

The symbol $[m]$ indicates the set $\{1, 2, \ldots, m\}$. Whenever we write $a \leftarrow b$ to assign the value of b to a, and similarly, write $a \xleftarrow{\$} S$, where S is a set, to mean that a is randomly sampled from S. Finally, λ denotes the security parameter and $\mathbb{P}$ represents a probability measure.

2.1 An Overview of TLS 1.3

TLS 1.3 [24] is the latest version of TLS, one of the most common protocols for secure communications over the Internet. The procedure is composed of two subprotocols: the Handshake and the Record layer. The goal of the first one is to negotiate secure symmetric keys between the two endpoints. The second one instead uses the bargained keys to protect the communications.

The Record Layer. Security is enforced by an AEAD (authenticated encryption with additional data), an encryption algorithm that guarantees privacy and integrity of the plaintext as well as the integrity of the header of the Record layer fragments. In order to encrypt and decrypt, an AEAD needs a different nonce for every fragment. These are deterministically derived from an initial vector (IV) and do not need to be kept secret. Indeed, security is guaranteed as long as the key remains private. The Record layer of TLS 1.3 provides for three types of fragments: Handshake messages, alerts and application data. The first type consists of all the information concerning the standard management of the TLS connection. Alerts are used to notify unexpected and potentially malicious events, whereas application data refers to the actual communication between the two endpoints (i.e. all the information for which they decided to use TLS 1.3).

The Handshake. We use the term "client" to denote the endpoint that initiates the connection, whereas the term "server" indicates the other endpoint. Furthermore, we define the transcript as the concatenation of all the messages exchanged on the connection until the analysed moment. The Handshake can be split into two phases: the key exchange phase, whose goal is to establish the keys and negotiate the cryptographic algorithms used in the connection, and the authentication phase, which has the objective of authenticating the endpoints and provide key confirmation.

The key exchange phase. The protocol is started by the client sending a ClientHello to the server. This is a message containing a 32-byte nonce (protection against replay attacks), information concerning the cryptographic algorithms supported by the client (signatures, AEADs, Diffie-Hellman groups) and at least one Diffie-Hellman public key. The server replies with a ServerHello, which contains another 32-byte nonce, the cryptographic algorithms selected among those offered by the client and one Diffie-Hellman public key. When the initial exchange is concluded, the two endpoints perform a Diffie-Hellman key exchange using the keys specified in the Hello messages. A key derivation function is then applied on the result. The operation takes as input also the transcript. At the end, the parties obtain two AEAD keys (handshake keys), as well as the related IVs. From that moment, the Record layer protects the client-to-server flow (resp. the serve-to-client flow) using the first key (resp. using the second key). The endpoints obtain also two MAC keys. After that, the server could provide further information concerning the management of the connection through the EncryptedExtensions message.

The authentication phase. Starting from the server, the endpoints exchange their certificates (ServerCertificate and ClientCertificate) and a signature on the transcript using the key specified on them (ServerCertificateVerify and ClientCertificateVerify). Finally, they send an HMAC on the transcript using the MAC keys obtained from the key derivation (ServerFinished and ClientFinished). In particular, the client uses the first key, the server uses the second one. In general, only the server is required to authenticate itself, the client is only required to send the ClientFinished. Anyway, the server may impose the authentication of the client through a CertificateRequest message. Clearly, the endpoints verify all the signatures and the MAC supplied by the other endpoint. If any check fails, the connection is closed.

Derivation of the keys. At the end of the Handshake, the endpoints feed the new messages of the transcript into the key derivation function.[3] At the end, the parties obtain two new AEAD keys (application keys), as well as the related new IVs. From that moment, the Record layer protects the client-to-server flow (resp. the serve-to-client flow) with the first new key (resp. with the second new key). The old keys are never used again.

[3] The key derivation function maintains an internal state, therefore all its outputs depend on the Diffie-Hellman secret and the Hello messages.

As it was proven in [12], all the values output by the Handshake are computationally independent. Furthermore, slight modifications in the transcripts input in the key derivation function would lead to completely different and unpredictable outputs.

Additional features. Clearly, what we presented in this section is not an exhaustive description of TLS 1.3. The protocol features many procedures that we did not mention, for instance, the key update mechanism, the use of pre-shared keys, 0-RTT, post-Handshake authentication, Hello-Retry-Requests, new session tickets and exporters. For further information on these topics, we refer to [24].

2.2 Multiparty Computation Protocols

Multiparty computation (MPC) deals with techniques that allow a set of parties (sometimes called an MPC engine) to jointly perform computations with security guarantees against external attackers as well as against dishonest parties.

Throughout the work we assume there is a fixed set of n parties, denoted $P_1, \ldots, P_n$. We want to prove security against an active adversary that can corrupt up to $n-1$ parties, in the static corruption model, so that the set of (indices of) honest parties $\mathcal{H} := \{i \in [n] \mid P_i \text{ is honest}\}$ is fixed and non-empty. We denote the set of corrupted parties $[n] \setminus \mathcal{H}$ with $\mathcal{C}$. Our security proofs are expressed in the universal composability (UC) framework [7].

Authenticated secret sharing. We use protocols based on additive secret-sharing schemes over finite fields, specifically, large prime fields, large binary fields and $\mathbb{F}_2$. We say that $x \in \mathbb{F}$ is secret-shared if every party P_i holds a random share $x_i \in \mathbb{F}$ such that $\sum_{i \in [n]} x_i = x$. As long as at least one party keeps its share secret, nobody learns anything about the value of x. If all parties collaborate, x can be reconstructed by revealing all the shares. This operation is called opening. To prevent corrupted parties from opening incorrect values, protocols typically augment the shares with information-theoretic MACs as in the SPDZ protocol [4,11]. Then, whenever secret-shared values are opened, the MACs can be checked and any tampering is detected with overwhelming probability.

Secret sharing over elliptic curve groups. Additive secret sharing, as presented above, can be performed over any finite group. Suppose now that E is an elliptic curve. Let G be one of its points with prime order q. In [8], the authors showed that the authenticated secret-sharing scheme of SPDZ over $\mathbb{F}_q$ induces an authenticated secret-sharing scheme over $\langle G \rangle$ which uses the same MAC key. Given a public value $a \in \mathbb{F}_q$ and a secret-shared point $[[Q]] \in \langle G \rangle$, this allows the parties to obtain shares $[[aQ]]$ (over E) without any communication between the parties. Moreover, given a secret-shared $[[a']] \in \mathbb{F}_q$ and a public point $Q' \in \langle G \rangle$, it is possible to non-interactively obtain $[[a'Q']]$ over E.

Arithmetic black box functionality. We work in the arithmetic black box model, which abstracts away the underlying details of secret-sharing by an ideal functionality. The functionality has separate commands for receiving inputs from the parties, performing certain arithmetic operations, and delivering outputs. We write $[[x]]$ to denote that a value $x \in \mathbb{F}$ is stored by the functionality under some public identifier known to all parties.

The specific functionality we use is $\mathcal{F}_{\text{MPC}}$, given in Fig. 16 in Appendix A. It supports computations on different fields, as well as over an elliptic curve as described above. It can also handle conversions between values stored in $\mathbb{F}_2$ and $\mathbb{F}_p$. These operations can be instantiated using protocols such as SPDZ [11] (for computations over large fields), TinyOT [23] or multi-party garbled circuits [17] (for computations over $\mathbb{F}_2$). Conversions between $\mathbb{F}_p$ and $\mathbb{F}_2$ can be done using so-called preprocessed doubly-authenticated bits [26]. For complete details on how the $\mathcal{F}_{\text{MPC}}$ functionality can be instantiated, we refer the reader to Section 4.3.

3 Overview of the Solution

Oblivious TLS is a protocol that allows an n-party engine to communicate with a TLS 1.3 endpoint, preserving privacy and correctness of the transmissions against up to $n-1$ corrupted parties. Effectiveness and security are guaranteed independently of the role played by the MPC engine in the TLS connection. For concreteness, however, in this paper we assume Oblivious TLS is adopted at the server side, which we expect to be the most common scenario.

The communicating party. We assume that only one of the parties manages the communication with the client. Supposing this is party P_1, then whenever the MPC engine has to send a message, P_1 is the entity that physically performs the operation. Moreover, when the client sends a message to the engine, P_1 receives it and shares it with the other parties. Clearly, P_1 can always perform a Denial-of-Service attack, by simply dropping the incoming or outgoing communications.

Handshake modes. The goal of our work was to design the simplest protocol that allowed a set of parties to communicate with a TLS 1.3 endpoint. For this reason, we focused our attention on Diffie-Hellman-based Handshakes without the use of pre-shared keys (see [24, Section 2]). We believe that Oblivious TLS can be extended to other Handshake modes. However, it might be the case that the use of pre-shared keys decreases the efficiency of the whole protocol as the key derivation would become more complicated. Since this is an expensive part of Oblivious TLS, opening a new connection might be preferable to resuming an older session.

Privacy of metadata. Oblivious TLS does not preserve privacy of Handshake messages and alerts against the corrupted parties of the MPC engine, but only against external attackers. This is because the derived handshake keys are revealed to the MPC engine, and the alerts are immediately opened upon receipt.

This choice allows a more efficient management of alerts and handshake messages, including verification of signatures and computation of transcripts. On the other hand, if an attacker corrupts any party of the engine, it gains access to the metadata of the connection. We do not believe this to be a huge concern, however, since this type of targeted attack is not typically feasible for, say, a mass surveillance adversary who aims to harvest metadata.

Handshake

The Handshake of Oblivious TLS is a multiparty execution of its original version. In particular, the messages exchanged between the client and the MPC engine are the same as in a traditional TLS 1.3 connection. However, additional security properties are guaranteed, specifically, the protocol protects the privacy of the application keys against up to $n - 1$ corrupted parties and ensures the authenticity of the multiparty endpoint. Both objectives are achieved using multiparty public keys, i.e. key pairs where the private counterpart is secret-shared. We now outline the main steps of the protocol.

Initialisation. To set up an Oblivious TLS server, the parties generate an EdDSA key using Π_{Sign} (see Section 4.4) and request a Certificate Authority to issue a certificate that binds the public key to the identity of the MPC engine. The private counterpart is secret-shared, therefore, its value remains secret as long as at least one party is honest. The key will be used to guarantee the authenticity of the communications. The MPC engine also generates a random seed s for a PRG (this can be done using commitment schemes). Every random value inserted in the Handshake messages must be generated using s and the selected PRG.

ClientHello and ServerHello. The two messages are generated following the specification of TLS 1.3. However the 32-byte nonces must be generated using the seed s and the selected PRG. Moreover, the messages must contain a not-necessarily-fresh DH public key which was generated using Π_{DH} or Π_{weakerDH} (see Sections 4.1 and 4.2). The private counterpart of such key is secret-shared, therefore, its value is known to nobody as long as at least one party is honest.

Cryptographic computations - Part I. After sending the Hello messages, the parties perform a multiparty Diffie-Hellman key exchange using Π_{DH} or Π_{weakerDH}, obtaining a secret-shared output. Then, the key derivation function is applied to the result using the MPC techniques described in Section 4. At the end, the Handshake keys[4] and IVs are opened. The MAC keys for ServerFinished and ClientFinished as well as the internal state of the key derivation function are instead kept in shared form.

[4] Using the notation of [24, Section 7.1], *client_handshake_traffic_secret* and *server_handshake_traffic_secret* **must not** be opened.

EncryptedExtensions, CertificateRequest and Certificates. These messages are generated and checked as described in the specification of TLS 1.3. Observe that their encryption and decryption can be computed locally by each party. Indeed, the handshake keys and IVs have been opened in the previous step. Clearly, the parties must send the certificate of the MPC engine.

ServerCertificateVerify and ClientCertificateVerify. Observe that the transcript of the connection is known to all the parties of the engine. Therefore, the verification of ClientCertificateVerify can be performed locally. The signature in ServerCertificateVerify is instead generated using the MPC protocol Π_{Sign} (see Section 4.4). Clearly, in order to do that, all the parties must agree on the transcript. In particular, they must check that P_1 generated a fresh nonce using s and the DH key was generated by the whole MPC engine. Since the EdDSA private key is shared, only with the collaboration of the whole engine, it is possible to generate a signature.

ServerFinished and ClientFinished. The generation of the HMAC in ServerFinished is performed by applying MPC algorithms to the corresponding secret-shared MAC key (see Section 4). The verification of the HMAC in ClientFinished instead does not require the use of any multiparty protocol. Indeed, the client MAC key can be opened just after the reception of the message. Each party can then check the MAC locally. It is fundamental that the MAC key is opened after the reception of ClientFinished. Otherwise, the protocol would not guarantee explicit authentication. In any case, opening the MAC keys for the verification does not affect security. Indeed, the opened MAC key is never used afterwards and its value does not leak any information.

Cryptographic computations - Part II. The second part of the key derivation is performed using the MPC techniques described in Section 4. The operation takes as input the full transcript of the connection as well as the internal state of the key derivation function, which is secret-shared. At the end, the new IVs are opened. The remaining outputs must be kept in shared form. The only exception to this rule occurs when our MPC-friendly AEAD is adopted (see Section 5.2). In that case, the i-th PRF key is opened to the i-th party.

Key Update. TLS 1.3 describes a key update scheme based on the key derivation function of the Handshake (see [24, Section 7.2]). Again, we can perform the operation using the MPC techniques presented in Section 4. At the end, the new IVs are opened. The remaining outputs must be kept in shared form (the only exception occurs when our MPC-friendly AEAD is adopted, the case is managed as described above).

Record Layer

The Record layer of Oblivious TLS is essentially an adaptation of the original protocol to secret-shared keys. As a consequence, the changes do not affect the

Handshake messages. Indeed, in that case, the keys as well as the nonces are known to every party. When we switch to the application keys, instead, only the nonces are known. For compatibility, the fragment partition, the additional data, the nonce generation and the padding are performed as in TLS 1.3. Encryptions and decryptions are instead executed using multiparty operations (see Section 5). Specifically, the encryption outputs a non-shared ciphertext taking as input a secret-shared plaintext, a secret-shared key and cleartext nonce and additional data. The decryption outputs either $\perp$ (in case of a tampered ciphertext) or a secret-shared plaintext and takes as input a secret-shared key and non-shared nonce, ciphertext and additional data. Upon decryption, the fragment type (which is encoded in the padding) is checked. In case of Alerts and Handshake messages, the plaintext is opened and handled according to TLS.

4 Handshake Operations

The Handshake of TLS 1.3 is based on Diffie-Hellman key exchange. Given the public key of the client, the parties running Oblivious TLS should be able to compute a secret-sharing of the exchanged secret. Notice that if any party P_i learns the exchanged secret, Oblivious TLS would completely lose its purpose as P_i could compute the symmetric keys and communicate with the client without any restriction. In order to design a multiparty Diffie-Hellman protocol, it is necessary for the parties to have a secret-shared private key. Clearly, the public key does need to be kept secret.

We chose to focus on Diffie-Hellman over elliptic curves, as it is the most popular version of the protocol and allows us to work over smaller finite fields than traditional DH. Specifically, we use the curve *Curve25519* of [5], although with minor changes the protocol could also use other curves of TLS 1.3.

In this section, we will present an actively secure protocol for Diffie-Hellman as well as a more efficient variant that allows some limited influence on the computation to the adversary. The downside of this solution is that it does not permit to directly reduce the security of Oblivious TLS to the security proof of TLS 1.3 [12] without introducing new cryptographic assumptions.

Diffie-Hellman notation. For the whole section, we assume to work with an elliptic curves E of equation

$$Y^2 = X^3 + AX^2 + BX + C$$

over a prime field of odd cardinality p. Furthermore, we assume the Diffie-Hellman group to be $\langle G \rangle$ where $G \in E$ has prime order q such that $q^2 \nmid |E|$. We denote the identity element of the group with ∞. Remember that this is the only non-affine point of the group. In this section, we use the notation $[[\,\cdot\,]]_q$, $[[\,\cdot\,]]_p$ and $[[\,\cdot\,]]_E$ to denote secret-sharings over $\mathbb{F}_q$, $\mathbb{F}_p$ and $\langle G \rangle$ respectively (modelled as values in $\mathcal{F}_{\text{MPC}}$).

For clarity, we assume the elliptic curve Diffie-Hellman key exchange to be the algorithm that, on input a secret key $s \in \mathbb{F}_q^\times$ and a point $Q \in \langle G \rangle \setminus \{\infty\}$, outputs

the x-coordinate of sQ. Actually, among all the elliptic curves supported by TLS 1.3, this description applies only to *Curve25519* and *Curve448*. The output of the other algorithms usually depends on both the coordinates of sQ.

Computations over elliptic curves. Recall that given two affine points (x_0, y_0) and (x_1, y_1) of the curve E such that $x_0 \neq x_1$, their sum (x_3, y_3) is computed as follows

$$m \leftarrow \frac{y_1 - y_0}{x_1 - x_0}, \qquad x_3 \leftarrow m^2 - A - x_0 - x_1, \qquad y_3 \leftarrow m(x_1 - x_3) - y_1 \tag{1}$$

We also recall that given an affine point (x_0, y_0) of E, its opposite is $(x_0, -y_0)$. As a consequence, two points P and Q of the curve have the same x-coordinate if and only if $P = Q$ or $P = -Q$.

Actually, there exist multiple ways to compute the addition between elliptic curve points. In traditional computation (i.e. non-multiparty computation), alternative coordinate systems are usually preferred as they permit to perform operations over elliptic curves without divisions. However, for secret-sharing based protocols like SPDZ the cost of a division is roughly twice the cost of a multiplication. All the division-free methods known so far need at least 10 multiplications to perform additions, so in our case, affine coordinates are still the best solution.

Key derivation, HMAC and key updates. After having performed the Diffie-Hellman key exchange, the obtained secret is input into a key derivation function which outputs multiple symmetric keys. In particular, TLS 1.3 uses the HKDF scheme of [22] which is based on hash functions (concretely, *SHA256* or *SHA384*). Since both the exchanged secret and the derived symmetric keys must remain private, the key derivation must be performed in MPC.

Before computing the hash function, we convert the secret from a $[[\cdot]]_p$ sharing into a $[[\cdot]]_2$ sharing, so we can compute the hash function as a binary circuit, using e.g. a garbled circuit-based protocol [17]. Alternatively, we could use a customized MPC-friendly hash function, however, this is non-standard and not supported by endpoints on the Internet.

The same approach can be used also to compute the IVs and the actual encryption keys of the AEAD (see [24, Section 7.3]), the HMAC keys and the HMACs used in *ClientFinished* and *ServerFinished* (see [24, Section 4.4.4]) and the key updates (see [24, Section 7.2]).

Signatures. The last cryptographic operation that the MPC engine needs to perform in the Handshake is the generation and verification of signatures. Since the transcript is known to all the parties of the engine, signatures can always be verified locally. Signing instead is more complex, indeed, the signature must be issued only with the approval of all the parties. We therefore use a threshold Schnorr-style protocol based on EdDSA signatures, given in Section 4.4. Since the message being signed is public, we can do this step without any expensive MPC operations.

4.1 Diffie-Hellman

DaPoint. The proposed protocol needs a particular preprocessing phase Π_{daPoint}, which is described in Figures 1 and 2. The description uses $\mathcal{F}_{\text{MPC}}$ and $\mathcal{F}_{\text{Rand}}$ as resources. The latter is a simple functionality that outputs a random permutation to all the parties.

The protocol Π_{daPoint} has the purpose of generating N doubly-authenticated-point (daPoint) tuples, i.e. random triples of the form $\big([[R]]_E, [[u]]_p, [[v]]_p\big)$ such that $R \in \langle G \rangle \setminus \{\infty\}$ and (u, v) are the affine coordinates of R. The algorithm is based on a cut-and-choose style bucketing technique [23].

It is possible to prove that Π_{daPoint} securely implements the functionality $\mathcal{F}_{\text{daPoint}}$ described in Figure 3.

Theorem 1. *Assuming that*

$$\omega := N\binom{M + N \cdot l}{l}^{-1} \quad \textit{and} \quad \omega' := \frac{M + N \cdot l}{q}$$

are negligible functions in the security parameter, $\Pi_{daPoint}$ securely implements $\mathcal{F}_{daPoint}$ in the $(\mathcal{F}_{MPC}, \mathcal{F}_{Rand})$-hybrid model.

The proof of theorem 1 can be found in Appendix B.1. We point out that if the order of the additions in step 7 of Π_{daPoint} is changed, the protocol is probably still secure but our proof does not apply anymore.

Complexity of daPoint. If we implement $\mathcal{F}_{\text{MPC}}$ using SPDZ, the execution of Π_{daPoint} takes $10+4(n-1)$ rounds. Each of the generated tuples has the following cost: $2nl - 1$ multiplicative triples in $\mathbb{F}_p$, $nl - 1$ division tuples in $\mathbb{F}_p$, $3nl - 1$ squaring couples in $\mathbb{F}_p$, $2n(l + M/N)$ input masks in $\mathbb{F}_p$, $n(l + M/N)$ input masks over $\mathbb{F}_q$ and the communication of

$$\big(3nM/N + 2M/N + 5l + 11nl - 9\big) \cdot \log(p) + nM/N + l - 1$$

bits for every party.

Multiparty Diffie-Hellman. Given the functionality $\mathcal{F}_{\text{daPoint}}$, it is possible to construct a multiparty protocol for elliptic curve Diffie-Hellman as described in Figure 5. The following theorem shows that Π_{DH} securely implements the functionality $\mathcal{F}_{\text{DH}}$ presented in Figure 4.

Theorem 2. *Assuming q^{-1} to be a negligible sequence in the security parameter, the protocol Π_{DH} securely implements the functionality $\mathcal{F}_{DH}$ in the $\mathcal{F}_{daPoint}$-hybrid model.*

The proof of theorem 2 can be found in Appendix B.2.

Complexity of Diffie-Hellman. If we implement $\mathcal{F}_{\text{MPC}}$ using SPDZ, the protocol Π_{DH} takes 4 rounds, 1 division tuple over $\mathbb{F}_p$, 1 squaring couple over $\mathbb{F}_p$, 1 daPoint tuple and the communication of $5\log(p) + 1$ bits for every party. The key generation requires instead just 1 random shared element of $\mathbb{F}_q$ and the communication of $\log(p) + 1$ bits for every party.

Π_{daPoint}

Let $M, N, l \in \mathbb{N}$ be three security-parameter-dependent values with $M, l \geq 2$.

MPC

The parties can issue queries to $\mathcal{F}_{\text{MPC}}$ but they cannot access the internal values (i.e. everything except the output) of the procedure DaPoint.

DaPoint

On input $(\text{daPoint}, (\text{id}_{i,1}, \text{id}_{i,2}, \text{id}_{i,3})_{i\in[N]})$ the parties compute the following steps:

1. For each $i \in [n]$, the parties generate $M + Nl$ random elements $[[z_{i,1}]]_q, [[z_{i,2}]]_q, \ldots, [[z_{i,M+Nl}]]_q$ in $\mathbb{F}_q$ such that $z_{i,j}$ is known only to P_i for each $j \in [M + Nl]$. This operation is performed using $\mathcal{F}_{\text{MPC}}$.
2. For each $i \in [n]$ and $j \in [M + Nl]$, the parties compute $[[Z_{i,j}]]_E \leftarrow [[z_{i,j}]]_q G$ using $\mathcal{F}_{\text{MPC}}$ and P_i computes $Z_{i,j}$ locally.
3. For each $i \in [n]$ and $j \in [M + Nl]$, party P_i computes $(x_{i,j}, y_{i,j})$, the affine coordinates of $Z_{i,j}$. If this is not possible since $Z_{i,j} = \infty$, the protocol aborts. Otherwise, P_i inputs $x_{i,j}$ and $y_{i,j}$ in $\mathcal{F}_{\text{MPC}}$ with domain $\mathbb{F}_p$.
4. The parties sample a random permutation ψ of $[M + Nl]$ using $\mathcal{F}_{\text{Rand}}$.
5. For each $i \in [n]$ and $j \in [M+Nl]\setminus[Nl]$, the parties open $[[Z_{i,\psi(j)}]]_E$, $[[x_{i,\psi(j)}]]_p$ and $[[y_{i,\psi(j)}]]_p$. If the affine coordinates of the former do not coincide with the latter, the protocol aborts.
6. For each $(i, j) \in [n] \times [Nl]$, the parties set $[[s_{i,\psi(j)}]]_p \leftarrow [[x_{i,\psi(j)}]]_p^2$ and open

$$t_{i,\psi(j)} \leftarrow [[y_{i,\psi(j)}]]_p^2 - [[x_{i,\psi(j)}]]_p \cdot [[s_{i,\psi(j)}]]_p - A \cdot [[s_{i,\psi(j)}]]_p - B \cdot [[x_{i,\psi(j)}]]_p - C.$$

   If any of the $t_{i,\psi(j)}$'s is different from zero, the protocol aborts.
7. For each $j \in [Nl]$, the parties set $[[R_{\psi(j)}]]_E \leftarrow \sum_{i\in[n]} [[Z_{i,\psi(j)}]]_E$ and $[[x_{\psi(j)}]]_p \leftarrow [[x_{1,\psi(j)}]]_p$, $[[y_{\psi(j)}]]_p \leftarrow [[y_{1,\psi(j)}]]_p$. Then, for $i \in [n] \setminus \{1\}$,

$$\begin{aligned}
[[m]]_p &\leftarrow \frac{[[y_{\psi(j)}]]_p - [[y_{i,\psi(j)}]]_p}{[[x_{\psi(j)}]]_p - [[x_{i,\psi(j)}]]_p} \\
[[x_{\psi(j)}]]_p &\leftarrow [[m]]_p^2 - A - [[x_{\psi(j)}]]_p - [[x_{i,\psi(j)}]]_p \\
[[y_{\psi(j)}]]_p &\leftarrow [[m]]_p \cdot ([[x_{i,\psi(j)}]]_p - [[x_{\psi(j)}]]_p) - [[y_{i,\psi(j)}]]_p.
\end{aligned}$$

   If for any i, m cannot be computed due to a zero denominator, the protocol aborts.
8. For each $(i, j) \in [N] \times [l]$, let $[[R_{i,j}]]_E := [[R_{\psi((i-1)l+j)}]]_E$ and

$$[[u_{i,j}]]_p := [[x_{\psi((i-1)l+j)}]]_p, \quad [[v_{i,j}]]_p := [[y_{\psi((i-1)l+j)}]]_p.$$

   The sequence $\mathcal{B}_i := (\psi((i - 1)l + j - 1))_{j\in[l]}$ is called the i-th bucket. This is equivalent to splitting the first Nl elements of the permuted sequence into blocks of l elements called buckets.

Fig. 1. The daPoint Protocol - Part 1

9. For each $i \in [N]$ and $j \in \{2, 3, \ldots, l\}$, the parties compute and open

$$W_{i,j} \leftarrow [[R_{i,1}]]_E + [[R_{i,j}]]_E$$
$$[[m]]_p \leftarrow \frac{[[v_{i,1}]]_p - [[v_{i,j}]]_p}{[[u_{i,1}]]_p - [[u_{i,j}]]_p}$$
$$w_{i,j} \leftarrow [[m]]_p^2 - A - [[u_{i,j}]]_p - [[u_{1,j}]]_p$$
$$w'_{i,j} \leftarrow [[m]]_p \cdot ([[u_{i,j}]]_p - [[w_{i,j}]]_p) - [[v_{i,j}]]_p$$

If for any j, m cannot be computed due to a zero denominator or the affine coordinates of $W_{i,j}$ do not coincide with $(w_{i,j}, w'_{i,j})$, the protocol aborts. Otherwise, for every $i \in [N]$, the parties store $[[R_{i,1}]]_E$, $[[u_{i,1}]]_p$ and $[[v_{i,1}]]_p$ with identities $\mathrm{id}_{i,1}$, $\mathrm{id}_{i,2}$ and $\mathrm{id}_{i,3}$.

Fig. 2. The daPoint Protocol - Part 2

$\mathcal{F}_{\mathrm{daPoint}}$

MPC

$\mathcal{F}_{\mathrm{daPoint}}$ replies to the queries as $\mathcal{F}_{\mathrm{MPC}}$ did.

daPoint

After receiving $(\mathrm{daPoint}, (\mathrm{id}_{i,1}, \mathrm{id}_{i,2}, \mathrm{id}_{i,3})_{i\in[N]})$ from every honest party and the adversary, $\mathcal{F}_{\mathrm{daPoint}}$ samples a random point R_i in $\langle G\rangle \setminus \{\infty\}$ for every $i \in [N]$. Let (u_i, v_i) be its affine coordinates. The functionality stores R_i, u_i and v_i with labels $\mathrm{id}_{i,1}$, $\mathrm{id}_{i,2}$ and $\mathrm{id}_{i,3}$.

Fig. 3. The daPoint Functionality

4.2 Diffie-Hellman With Influence of the Adversary

We now present the second version of the Diffie-Hellman protocol. We called the procedure "weaker Diffie-Hellman" in order to highlight the difference with the protocol described in the previous section. Anyway, we will argue that weaker Diffie-Hellman is sufficiently secure for Oblivious TLS.

ECSum. The general idea of the protocol is similar to the one presented in the Section 4.1. We still need a preprocessing phase Π_{ECSum} which is described in Figure 6 using $\mathcal{F}_{\mathrm{MPC}}$ as a resource. The purpose of Π_{ECSum} is to generate random tuples of the form $(R_1, R_2, \ldots, R_n, [[x]]_p, [[y]]_p)$ such that, for each $i \in [n]$, R_i is an affine point of $\langle G\rangle$ known only to party P_i and $([[x]]_p, [[y]]_p)$ is a secret-sharing of the affine coordinates of $R := \sum_{i\in[n]} R_i$.

As formalised by the following theorem, it is possible to prove that Π_{ECSum} securely implements the functionality $\mathcal{F}_{\mathrm{ECSum}}$ described in Figure 7.

Theorem 3. *Assuming q^{-1} to be a negligible sequence in the security parameter, the protocol Π_{ECSum} securely implements the functionality $\mathcal{F}_{ECSum}$ in the $\mathcal{F}_{MPC}$-hybrid model.*

$\mathcal{F}_{\text{DH}}$

MPC

$\mathcal{F}_{\text{DH}}$ replies to the queries as $\mathcal{F}_{\text{daPoint}}$ did.

Key Generation

After receiving $(\text{KeyGen}, \text{id})$ from every honest party and the adversary, $\mathcal{F}_{\text{DH}}$ samples a random value $s \xleftarrow{\$} \mathbb{F}_q^{\times}$ and computes $S \leftarrow sG$. Then, it passes S to the adversary and waits for a reply. If the answer is OK, $\mathcal{F}_{\text{DH}}$ outputs S to every honest party and stores s with label id. Otherwise, it aborts.

Diffie-Hellman

After receiving $(\text{DH}, \text{id}_1, Q, \text{id}_2)$ from the adversary and every honest party, $\mathcal{F}_{\text{DH}}$ retrieves the private key s of label id_1. If $Q = \infty$ or $qQ \neq \infty$, the functionality does nothing. Otherwise, it computes e, the x coordinate of sQ, and stores it with identity id_2.

Fig. 4. The Diffie-Hellman Functionality

Π_{DH}

MPC

The parties can issue queries to $\mathcal{F}_{\text{daPoint}}$ but they cannot access the private keys and the internal values of the key exchange procedure.

Key Generation

On input $(\text{KeyGen}, \text{id})$ the parties perform the following steps

1. Sample a random secret value $[[s]]_q \in \mathbb{F}_q$ and set $[[S]]_E \leftarrow [[s]]_q G$.
2. Call $\mathcal{F}_{\text{daPoint}}$ to open $[[S]]_E$. If $S = \infty$, the protocol restarts.
3. Store the secret key $[[s]]_q$ with label id and output the public key S.

Diffie-Hellman

On input $(\text{DH}, \text{id}_1, Q, \text{id}_2)$ the parties perform the following steps

1. They retrieve the private key $[[s]]_q$ with label id_1. Such key is stored in $\mathcal{F}_{\text{daPoint}}$. If $Q = \infty$ or $qQ \neq \infty$, the protocol stops.
2. They compute $[[Z]]_E \leftarrow [[s]]_q Q$ using $\mathcal{F}_{\text{daPoint}}$.
3. They call $\mathcal{F}_{\text{daPoint}}$ to obtain a random daPoint tuple $([[R]]_E, [[x]]_p, [[y]]_p)$.
4. They compute and open $W \leftarrow [[Z]]_E - [[R]]_E$.
5. Let (u, v) be the affine coordinates of W. If $W = \infty$, the final output is $[[x]]_p$. Otherwise, using $\mathcal{F}_{\text{daPoint}}$, the parties compute the output

$$[[e]]_p \leftarrow \left(\frac{[[y]]_p - v}{[[x]]_p - u}\right)^2 - A - [[x]]_p - u.$$

   In case of zero denominator, the protocol aborts. The value of $[[e]]_p$ is stored with label id_2.

Fig. 5. The Diffie-Hellman Protocol

Π_{ECSum}

MPC

The parties can issue queries to $\mathcal{F}_{\text{MPC}}$ but they cannot access the internal values (i.e. everything except the output) of the procedure ECSum.

ECSum

On input $(\text{ECSum}, \text{id}_1, \text{id}_2)$ the parties compute the following steps:

1. Each party P_i selects a random point $R_i \in \langle G \rangle \setminus \{\infty\}$ and inputs the two affine coordinates of the point (x_i, y_i) in $\mathcal{F}_{\text{MPC}}$.
2. For each $i \in [n]$, the parties compute $[[s_i]]_p \leftarrow [[x_i]]_p^2$ and they open
$$t_i \leftarrow [[y_i]]_p^2 - [[x_i]]_p \cdot [[s_i]]_p - A \cdot [[s_i]]_p - B \cdot [[x_i]]_p - C.$$
If any of the t_i's is different from zero, the protocol aborts.
3. The parties set $[[x]]_p \leftarrow [[x_1]]_p$, $[[y]]_p \leftarrow [[y_1]]_p$ and later, for $i \in [n] \setminus \{1\}$,
$$\begin{aligned} [[m]]_p &\leftarrow \frac{[[y]]_p - [[y_i]]_p}{[[x]]_p - [[x_i]]_p} \\ [[x]]_p &\leftarrow [[m]]_p^2 - A - [[x]]_p - [[x_i]]_p \\ [[y]]_p &\leftarrow [[m]]_p \cdot ([[x_i]]_p - [[x]]_p) - [[y_i]]_p. \end{aligned}$$
If for any i, m cannot be computed due to a zero denominator, the protocol aborts. Otherwise, the parties store $[[x]]_p$ and $[[y]]_p$ with labels id_1 and id_2.
4. Each party P_i outputs the point R_i.

Fig. 6. The EC-Sum Protocol

$\mathcal{F}_{\text{ECSum}}$

MPC

$\mathcal{F}_{\text{ECSum}}$ replies to the queries as $\mathcal{F}_{\text{MPC}}$ did.

ECSum

On input $(\text{ECSum}, \text{id}_1, \text{id}_2)$ from each honest party and the adversary, $\mathcal{F}_{\text{ECSum}}$ waits for a point $U \in E$ from the adversary. Then, it keeps sampling random points R_i in $\langle G \rangle$ for every $i \in \mathcal{H}$ until $R := U + \sum_{i \in \mathcal{H}} R_i \neq \infty$. Finally, it stores the coordinates of R with labels id_1 and id_2 and outputs R_i to P_i for each $i \in \mathcal{H}$.

Fig. 7. The EC-Sum Functionality

The proof of theorem 3 can be found in Appendix C.1. We point out that if the order of the additions in step 3 of Π_{ECSum} is changed, the procedure is probably still secure but the proof that we are going to present does not apply anymore.

Complexity of ECSum. If we implement $\mathcal{F}_{\text{MPC}}$ using SPDZ, the execution of Π_{ECSum} takes $4n-2$ rounds, $2n-1$ multiplicative triples in $\mathbb{F}_p$, $3n-1$ squaring couples in $\mathbb{F}_p$, $n-1$ division tuples in $\mathbb{F}_p$, $2n$ input masks in $\mathbb{F}_p$ and the communication of $(10n-4)\log(p)$ bits for each party.

"Weaker" DH. Using Π_{ECSum} as a subprotocol, we present a more efficient version of the elliptic curve Diffie-Hellman, called Π_{weakerDH}, in Figure 9. It uses $\mathcal{F}_{\text{ECSum}}$, as well as a key set-up functionality $\mathcal{F}_{\text{Key}}$ (see Figure 8), which can be implemented by having each P_i broadcast S_i, and run a zero-knowledge proof of knowledge of the secret s_i.

$\mathcal{F}_{\text{Key}}$

Key

On input (Key) from each party, $\mathcal{F}_{\text{Key}}$ samples $s_i \xleftarrow{\$} \mathbb{F}_q$ for each $i \in \mathcal{H}$ and computes $S_i \leftarrow s_i G$. When the adversary provides a pair (s_j, S_j) for every $j \in \mathcal{C}$, the functionality answers with $\{S_i\}_{i\in\mathcal{H}}$ and waits for a reply. If the adversary sends OK, $\mathcal{F}_{\text{Key}}$ checks that $S_j = s_j G$ for each $j \in \mathcal{C}$. In such case, it outputs $(s_i, S_1, S_2, \ldots, S_n)$ to P_i $\forall i \in \mathcal{H}$. Otherwise, it sends $\perp$ to each honest party.

Fig. 8. The functionality $\mathcal{F}_{\text{Key}}$

As we already mentioned, Π_{weakerDH} does not securely implement the functionality $\mathcal{F}_{\text{DH}}$ from the previous section (Figure 4), but a slightly weaker functionality $\mathcal{F}_{\text{weakerDH}}$ (Figure 10). The difference is that now the adversary may "shift" the exchanged secret by an error point Q_ϵ. Moreover, it permits to test whether the exchanged secret $sQ + Q_\epsilon$ coincides with some point Q_τ chosen by the adversary. Note that when using the functionality for TLS, this does not increase the capabilities of the attacker, since in TLS, it is always possible to perform such a test by applying the key derivation on Q_τ and trying to decrypt a ciphertext transmitted by the endpoints.

Theorem 4. *Assuming q^{-1} to be a negligible sequence in the security parameter λ, the protocol $\Pi_{weakerDH}$ securely implements the functionality $\mathcal{F}_{weakerDH}$ in the $(\mathcal{F}_{ECSum}, \mathcal{F}_{Key})$-hybrid model.*

The proof of theorem 4 can be found in Appendix C.2.

Complexity of "weaker" DH. If we implement $\mathcal{F}_{\text{MPC}}$ using SPDZ, the Diffie-Hellman procedure in Π_{weakerDH} usually takes 4 rounds, 1 squaring couple in

Π_{weakerDH}

MPC

The parties can issue queries to $\mathcal{F}_{\text{ECSum}}$ but they cannot access the internal values of the key exchange procedure.

Key Generation

On input $(\text{KeyGen}, \text{id})$, the parties send (Key) to $\mathcal{F}_{\text{Key}}$. If $\mathcal{F}_{\text{Key}}$ replies with $(s_i, S_1, S_2, \ldots, S_n)$, P_i outputs $S \leftarrow \sum_{j\in[n]} S_j$ and stores s_i with label id. Otherwise, it aborts.

Diffie-Hellman

On input $(\text{DH}, \text{id}_1, Q, \text{id}_2)$, the parties perform the following steps

1. Each party P_i checks that $qQ = \infty$ and $Q \neq \infty$. If this is not the case, it stops. Otherwise, it retrieves the share s_i of the private key with label id_1.
2. Using ECSum, P_i obtains a random point R_i and shares of $[[x]]_p, [[y]]_p$.
3. Each party P_i broadcasts $W_i \leftarrow a_i Q - R_i$. If there exists $i \in [n]$ such that $W_i \notin E$, the protocol aborts. Otherwise, the parties set $W \leftarrow \sum_{i\in[n]} W_i$.
4. Let (u, v) be the affine coordinates of W. If $W = \infty$, the final output is $[[x]]_p$. Otherwise, using $\mathcal{F}_{\text{ECSum}}$, the parties compute the output

$$[[e]]_p \leftarrow \left(\frac{[[y]]_p - v}{[[x]]_p - u}\right)^2 - A - [[x]]_p - u.$$

   If the division fails due to a zero denominator, the protocol aborts. The value of $[[e]]_p$ is stored with label id_2.

Fig. 9. The weaker Diffie-Hellman Protocol

$\mathcal{F}_{\text{weakerDH}}$

MPC

$\mathcal{F}_{\text{weakerDH}}$ replies to the queries as $\mathcal{F}_{\text{ECSum}}$ did.

Key Generation

After receiving $(\text{KeyGen}, \text{id})$ from each party, $\mathcal{F}_{\text{weakerDH}}$ samples a random value $s \xleftarrow{\$} \mathbb{F}_q$ and computes $S \leftarrow sG$. Then, it sends S to the adversary and waits for a reply. If the answer is OK, $\mathcal{F}_{\text{weakerDH}}$ outputs S to each honest party and stores s with label id. Otherwise, it aborts.

Diffie-Hellman

After receiving $(\text{DH}, \text{id}_1, Q, \text{id}_2)$ from every party, $\mathcal{F}_{\text{weakerDH}}$ checks whether $qQ = \infty$ and $Q \neq \infty$. If this is not the case, it stops. Otherwise, it retrieves the private key s of label id_1. Then, it waits for two point Q_ϵ and Q_τ from the adversary and computes $Z \leftarrow Q_\epsilon + sQ$. If $Z = \infty$ or $Z = Q_\tau$, $\mathcal{F}_{\text{weakerDH}}$ sends OK to the adversary and aborts. Otherwise, it computes e, the x coordinate of Z, and stores it with identity id_2.

Fig. 10. The weaker Diffie-Hellman Functionality

$\mathbb{F}_p$, 1 division tuple in $\mathbb{F}_p$, 1 ECSum tuple and the communication of $5\log(p)+1$ bits for each party.

Notice that from a complexity point of view, the improvement of Π_{weakerDH} over Π_{DH} is due to the lower cost of ECSum tuples compared to daPoint tuples.

4.3 Instantiating $\mathcal{F}_{\text{MPC}}$ on the Required Fields.

In concrete situations, we cannot choose the elliptic curve used by the Diffie-Hellman algorithm. As a matter of fact, TLS 1.3 supports only 5 secure curves. Although, it may be possible to find other secure curves, very few endpoints of the Internet would support them. For this reason, the choice of the fields $\mathbb{F}_p$ and $\mathbb{F}_q$ used by $\mathcal{F}_{\text{MPC}}$ is very restricted. If $\mathcal{F}_{\text{MPC}}$ is implemented using SPDZ, the Offline phase (i.e. the expensive preprocessing phase which is necessary to perform multiplications and inputs) must be instantiated on these fields.

HE-based Offline phase. There exist several protocols that can be used for the Offline phase. Some of them (Overdrive [20] and the original Offline phase of SPDZ [11]) are based on homomorphic encryption. Unfortunately, these solutions come with strong constraints on the fields on which they can be instantiated. In particular, $\mathbb{F}$ must be a prime field and there must exist a suitable $m \in \mathbb{N}$ such that $2^m \mid |\mathbb{F}| - 1$. None of the fields we are interested in satisfy this property. Actually, these requirements can be relaxed a little bit. It is indeed possible to instantiate these protocols also on fields $\mathbb{F}$ such that there exists a sufficiently large prime p that divides $|\mathbb{F}| - 1$. In concrete applications, p should be close to 2^{14}. In these cases, the parameters of the homomorphic encryption scheme must be modified in order to meet the security and correctness requirements (see [11, Appendix C], [15, Appendices A and B] and [20, Section 3]). In general, with equal security guarantees, this leads to lower efficiency.

OT-based Offline phase. Another option is to use an Offline phase based on Oblivious Transfer such as MASCOT [19]. This solution is not as efficient as Overdrive, however, the only requirement on the field is that its cardinality is sufficiently close to a power of 2. This condition is satisfied by both $\mathbb{F}_p$ and $\mathbb{F}_q$ for most the elliptic curves proposed by TLS 1.3.

Ring-LPN-based Offline phase. The last option is to use the Ring-LPN-based 2-party Offline phase of [6]. This solution is the most efficient known so far, especially from a communicational point of view. In general, the protocol has the same requirement on the field as the homomorphic-encryption-based solutions. This is due to a matter of efficiency, as the property in question permits to use the Fast Fourier Transform to enhance the speed of the algorithm. However, it is possible to relax the condition and use any field of cardinality sufficiently close to a power of 2. Clearly, the Fast Fourier Transform cannot be used anymore in these cases.

4.4 Signature Generation

The authentication of the TLS connection is essentially based on signatures. Since the identity of the MPC engine consists in the union of all its parties, it is necessary for the private key to be secret-shared, otherwise, an attacker may issue new signatures without having control of all the parties.

In TLS 1.3, the endpoints sign the transcript of the Handshake. Since the latter is known to all the members of the MPC engine, it is sufficient that we design a multiparty protocol that on input a secret-shared key $[[a]]$ and a cleartext message m, outputs a cleartext signature $s \leftarrow \mathrm{Sign}(a, m)$. Clearly, the signing algorithm should be supported by TLS 1.3. We decided to base our protocol on EdDSA. Indeed, Schnorr signatures have interesting homomorphic properties that suit our context.

$\mathcal{F}_{\mathrm{Sign}}$

Let G be the base point of the curve and let q be its order.
Initialization. Upon receiving (Init) from every party, $\mathcal{F}_{\mathrm{Sign}}$ generates a random pair (a, A) such that $A = aG$. Then, it sends A to the adversary and waits for a reply. If the answer is OK, the functionality outputs A to every honest party and stores (a, A). Otherwise, it aborts.
Sign: Upon receiving (Sign, m) from every party, where m is in $\{0,1\}^*$, $\mathcal{F}_{\mathrm{Sign}}$ generates a signature (r, R) of m using the stored key. Then, it sends (s, R, m) to the adversary. If the answer is OK, $\mathcal{F}_{\mathrm{Sign}}$ outputs (s, R) to every honest party.
Abort: On input (Abort) from the adversary, the functionality aborts.

Fig. 11. The functionality $\mathcal{F}_{\mathrm{Sign}}$

Schnorr signatures. We briefly recall how Schnorr signatures are generated. Let $(\langle G\rangle, +)$ be an elliptic curve group of prime order q and suppose that the discrete logarithm problem is hard over $\langle G\rangle$. Let $H : \{0,1\}^* \longrightarrow \mathbb{F}_q$ be a hash function. A private key is a random element $a \in \mathbb{F}_q$, whereas its public counterpart is defined to be $A := aG$. A signature (R, s) of a message $m \in \{0,1\}^*$ is generated as follows

$$r \xleftarrow{\$} \mathbb{F}_q, \qquad R \leftarrow rG, \qquad s \leftarrow (r + H(R, A, m) \cdot a) \bmod q$$

The signature can be verified by checking whether $sG = R + H(R, A, m)A$.

Multiparty signature. Using the functionality $\mathcal{F}_{\mathrm{Key}}$ as a resource (see Figure 8), the parties can generate EdDSA signatures using the protocol Π_{Sign} described in Figure 12. The proof of the following theorem can be found in Appendix D.

Theorem 5. *The protocol Π_{Sign} securely implements $\mathcal{F}_{Sign}$ in the $\mathcal{F}_{Key}$-hybrid model.*

Π_{Sign}

Let G be the base point of the curve and let q be its order.

Initialization. Party P_i sends (Key) to $\mathcal{F}_{\text{KEY}}$. If $\mathcal{F}_{\text{KEY}}$ replies with $(a_i, A_1, A_2, \ldots, A_n)$, P_i outputs $A \leftarrow \sum_{j\in[n]} A_j$ and stores (A, a_i). Otherwise, it aborts.

Sign. Let m be in $\{0,1\}^*$, every party P_i performs the following operations.

1. It sends (Key) to $\mathcal{F}_{\text{KEY}}$. If $\mathcal{F}_{\text{KEY}}$ replies with $(r_i, R_1, R_2, \ldots, R_n)$, P_i computes $R \leftarrow \sum_{j\in[n]} R_j$. Otherwise, it aborts.
2. P_i computes and broadcasts $s_i \leftarrow (r_i + H(R, A, m) \cdot a_i) \bmod q$.
3. P_i waits for s_j from every other party P_j and computes $s \leftarrow \sum_{i\in[n]} s_i \bmod q$. Then, it checks that $sG = R + H(R, A, m) \cdot A$. If this is not true, it aborts. Otherwise, it outputs (R, s).

Fig. 12. The protocol Π_{Sign}

5 Record Layer Operations

In the Record layer for Oblivious TLS, we need secure protocols that given an AEAD scheme $(\mathcal{E}, \mathcal{D})$ and a secret-shared symmetric key $[[K]]$, allow performing the following operations

- **Encrypt.** On input a cleartext nonce N, cleartext associated data A and a secret-shared plaintext $[[X]]$, output a cleartext string $C = \mathcal{E}(K, N, A, X)$.
- **Decrypt.** On input a cleartext nonce N, cleartext associated data A and a non-shared ciphertext C, output $\perp$ if and only if $\perp = \mathcal{D}(K, N, A, C)$. Otherwise, output a secret-shared value $[[X]]$ where $X = \mathcal{D}(K, N, A, C)$.

The secret-sharing scheme used in this high-level description strongly depends on the AEAD. Observe that in practical situations, there might be a mismatch between the secret-sharing scheme used by the application on top of TLS and the secret-sharing scheme used by the AEAD. In such cases, we assume that suitable conversions were already performed.

Padding. In TLS 1.3, the plaintexts always have a padding. Whereas its application is a simple operation, the removal can be a bit complex. Indeed, using multiparty computation, we need to discover the position of the last non-zero byte and open it. Its value encodes the fragment type (see [24, Section 5.1]). In the case of an alert or Handshake data (key update requests, post Handshake authentication or new session tickets), the plaintext is simply opened and handle according TLS 1.3. In the case of application data, the first part of the plaintext (up to the second last non-zero value) is kept in shared form and is handled following the instructions of the application on top of TLS.

Supported AEADs. Oblivious TLS supports two different AEAD schemes. The first one is AES-GCM, one of the most popular encryption algorithms. The second one is instead a novel AEAD specifically designed by us for Oblivious TLS,

and avoids all evaluation of block ciphers inside MPC. The efficiency of the MPC friendly AEAD is considerably better than AES-GCM, however, the downside is that a custom algorithm is generally not supported by TLS clients. Both solutions rely on MACs to guarantee integrity. For this reason, the associated MAC keys must always be kept in shared form, otherwise a corrupted party would be able to tamper with the communications.

5.1 AES-GCM

We decided to adopt AES-GCM as it seemed to allow the most efficient MPC execution among all the AEADs suggested by TLS 1.3 (see [24, Section B.4]).

Overview of AES-GCM. We briefly recall how the cipher works (for details, see [13]). Let k be the key and let N be a nonce. Let $\mathrm{AES}(k, x)$ denote the encryption of x under the key k using the AES block cipher. The algorithm defines the MAC key $H := \mathrm{AES}(k, O)$ where O is the 128-bit string entirely made of zeros. Figure 13 describes the encryption procedure. The value C_0 is usually called the MAC of the AEAD. Decryptions are performed in a similar way: at the beginning the MAC is regenerated from the ciphertext and the result is compared with the MAC received from the client. If the check fails, the algorithm outputs $\perp$, otherwise the plaintext is retrieved by reversing the operations of the encryption.

1. The plaintext is split into 128-bit blocks $X_1, X_2, \ldots, X_L$. Do the same on the associated data to get $A_1, A_2, \ldots, A_{L'}$.
2. From N, $L+1$ 128-bit nonces $N_0, N_1, N_2, \ldots, N_L$ are derived.
3. Set $C_i \leftarrow X_i \oplus \mathrm{AES}(k, N_i)$ for every $i \in [L]$.
4. Let S be an encoding of L and L' as a 128-bit string.
5. Set $M \leftarrow \bigoplus_{i=1}^{L'} A_i \cdot H^i \oplus \bigoplus_{i=1}^{L} C_i \cdot H^{i+L'} \oplus S \cdot H^{L+L'+1}$.
6. Set $C_0 \leftarrow M \oplus \mathrm{AES}(k, N_0)$ and output $C_0, C_1, \ldots, C_L$.

Fig. 13. AES-GCM encryption

Multiparty AES Evaluation. To run AES-GCM inside MPC, we need many evaluations of AES on cleartext inputs derived from the nonce, under a secret-shared key and with secret-shared output. We consider two methods for evaluating AES:

- The secret-sharing based AES evaluation of [18]. This solution might be preferable when the parties can communicate over fast networks.
- A multi-party garbled circuit protocol such as [17]. This involves evaluating AES as a binary circuit, but obtains a constant round complexity so may be preferable over slow networks.

After the AES evaluations, the parties get $[[C_i]]$ from $[[X_i]]$ (in the encryption) or $[[X_i]]$ from C_i (in the decryption) for every $i \in [L]$. In encryption, $[[C_i]]$ is opened for every $i \in [L]$.

The MAC generation. It remains to explain how C_0 is generated (or checked in the case of a decryption). Suppose now that we have a secret-sharing of $[[H^i]]$ over $\mathbb{F}_{2^{128}}$ for every $i \in [L+L'+1]$. It is possible to obtain a secret-sharing of the MAC $[[C_0]]$ without communication between the parties. Indeed, the additional data as well as $C_1, C_2, \ldots, C_L$ and S are cleartext information, therefore

$$[[C_0]] = [[\mathrm{AES}(k, N_0)]] \oplus \bigoplus_{i=1}^{L'} A_i \cdot [[H^i]] \oplus \bigoplus_{i=1}^{L} C_i \cdot [[H^{L'+i}]] \oplus S \cdot [[H^{L+L'+1}]].$$

The MAC must be opened only in the case of an encryption, otherwise H might be leaked. When decrypting, the parties must check $[[C_0]]$ for equality with the first 128-bit block of the ciphertext received from the client. The operation must leak no information besides the output bit.

Once and for all operations. Observe that some operations do not have to be repeated for every encryption or decryption. Specifically, for every symmetric key, the AES key scheduling, the computation of the secret-shared MAC key $[[H]]$ and its powers can be performed only once. Clearly, it is sufficient to compute the first T powers where T is an upper bound on $L+L'+1$. In TLS 1.3, $L+L'+1$ is at most 1026, but depending on the application, it may be smaller.

5.2 An MPC-friendly AEAD

To improve efficiency, we designed a distributed AEAD scheme that avoids evaluations of PRFs or block ciphers inside MPC, removing the most expensive part of AES-GCM. Note that distributed AEAD was also considered in [3], however, in their setting the message being encrypted was known to a single party, which is not the case here.

Some disadvantages. Clearly, our MPC-friendly AEAD is not included in the cipher suites supported by TLS 1.3 by default, hindering the usability of the scheme. However, the specification permits the adoption of new cipher suites for private use (see [24, Section 11]). The efficiency of the construction in MPC comes with another drawback. Specifically, the number of keys used in the AEAD depends on the number of parties in the MPC engine. That means that such information has to be somehow communicated to the client (e.g. in the Hello messages). Since, the MPC execution is secure if and only if the keys are independent, a high number of parties also means a complicated and expensive key derivation.

The basic idea. Let λ be a security parameter and suppose that our plaintexts are shared over a field $\mathbb{F}_p$. Let

$$F : \{0,1\}^\lambda \times \{0,1\}^{l_1} \longrightarrow \mathbb{F}_p^{l_2}$$

be a PRF. We define a second PRF

$$\begin{aligned} F^n &: \left(\{0,1\}^\lambda\right)^n \times \{0,1\}^{l_1} \longrightarrow \mathbb{F}_p^{l_2} \\ F^n(k_1, k_2, \ldots, k_n, x) &:= \sum_{i \in [n]} F(k_i, x). \end{aligned}$$

Suppose now that the key derivation distributes a random k_i to party P_i for each $i \in [n]$. Given l_2 secret-shared elements $[[y_1]], [[y_2]], \ldots, [[y_{l_2}]]$, the parties can compute $\mathbf{y} + F^n(k_1, k_2, \ldots, k_n, x)$ where $\mathbf{y} = (y_1, y_2, \ldots, y_{l_2})$ as follows:

1. Every P_i sets $\mathbf{z_i} \leftarrow (y_{1,i}, y_{2,i}, \ldots, y_{l_2,i})$ where $y_{j,i}$ is the i-th share of $[[y_j]]$.
2. Every party P_i broadcasts $\mathbf{c_i} \leftarrow \mathbf{z_i} + F(k_i, x)$.
3. The parties compute $\mathbf{c} \leftarrow \mathbf{c_1} + \mathbf{c_2} + \cdots + \mathbf{c_n}$.

In this way, we obtain an encryption of y without leaking any information, however, a dishonest party is able to insert an additive error in the computation. Actually, that is not a problem. Indeed, the adversary can always intercept a ciphertext travelling through the network and add an arbitrary error. The security of the communication is still guaranteed as long as the MAC of the AEAD is correctly computed.

MAC-then-Encrypt and the PolyMAC function. Since there is no assurance that the ciphertext is correctly generated, any Encrypt-then-MAC AEAD cannot be used. For this reason, we decided to design an OTP-based MAC-then-Encrypt scheme (MtE) using the following MAC function. We denote the resulting AEAD with $\mathrm{MtE}(F^n, \mathrm{PolyMAC})$.

Definition 1 (PolyMAC). *Let L_1 and L_2 be two security-parameter-dependent values. Let Ecd be an injective function from $\{0, 1, \ldots, L_2\}$ to $\mathbb{F}_p^\times$. Let $k := (y, z) \in \mathbb{F}_p^2$ be a key and suppose the message to authenticate is*

$$m := ((a_1, a_2, \ldots, a_r), (x_1, x_2, \ldots, x_l)) \in \mathbb{F}_p^r \times \mathbb{F}_p^l$$

with $r \leq L_1$ and $l \leq L_2$. We define PolyMAC as

$$\mathit{PolyMAC}(k, m) := \mathit{Ecd}(l) \cdot y^{l+r+1} + \sum_{i=1}^{r} a_i \cdot y^{l+r+1-i} + \sum_{i=1}^{l} x_i \cdot y^{l+1-i} + z.$$

Clearly, when PolyMAC is plugged in the AEAD, $(a_1, a_2, \ldots, a_r)$ becomes the additional data, whereas $(x_1, x_2, \ldots, x_l)$ is the plaintext.

Theorem 6. *If*

$$\frac{L_1 + L_2 + 1}{p}$$

is negligible in the security parameter, MtE(F^n, PolyMAC) is a secure AEAD for every $n \in \mathbb{N}^{>0}$.

The proof of theorem 6 can be found in Appendix E. Observe that in many multiparty computation protocols, p is $O(2^\lambda)$. Therefore, if we choose L_1 and L_2 to be linear in the security parameter, we obtain a secure AEAD.

MPC execution of the AEAD. Performing encryptions and decryptions on secret-shared data with the new AEAD is very easy and cheap. First of all, notice that a secret-sharing of the MAC $[[M]]$ can be computed with active security. Indeed, given a secret-shared MAC key $([[y]], [[z]])$, cleartext additional data $(a_i)_{i\in[r]}$ and secret-shared plaintext $\left([[x_i]]\right)_{i\in[l]}$,

$$[[M]] := \mathrm{Ecd}(l) \cdot [[y^{l+r+1}]] + \sum_{i=1}^{r} a_i \cdot [[y^{l+r+1-i}]] + \sum_{i=1}^{l} [[x_i]] \cdot [[y^{l+1-i}]] + [[z]].$$

Encryptions can then be performed as explained at the beginning of this section. Decryptions are instead a little trickier. Indeed, in order to obtain an authenticated secret-shared plaintext, each party P_i has to input $F(k_i, x)$ in $\mathcal{F}_{\mathrm{MPC}}$. The rest of the operations are rather straightforward. In any case, we stress that $[[M]]$ must be checked without opening it. Clearly, in both encryption and decryption, the adversary can insert an additive error. Anyway, even an usual attacker controlling the network had this possibility, therefore, security is guaranteed by the MAC. The protocol is thoroughly described in Figure 14.

Complexity of the MPC-friendly AEAD. Observe that just as for AES-GCM, we do not need to compute the powers of $[[y]]$ every time that an encryption or a decryption is performed. We can just perform the operation once, for instance, immediately after the keys are derived.

The total complexity of encryption is l multiplicative triples and the communication of $n(3l + 1)$ elements in $\mathbb{F}_p$. The complexity of a decryption instead is l multiplicative triples, the communication of $n(3l + 1)$ elements in $\mathbb{F}_p$, a comparison and $n(l + 1)$ input masks. Considering that this is the cost of the communication of l elements in $\mathbb{F}$, the protocol is significantly cheap. We highlight that if some elements of the plaintext are public (e.g. because of the padding) we could spare some multiplicative triples in the generation of the MAC.

Secure communication between MPC clusters. The new AEAD permits to communicate between two MPC engines too. Security is guaranteed as long as the adversary does not know at least one of the PRF keys. Therefore, the parameter n and the key distribution must be properly studied to suit the adversarial model. In particular, if the adversary can corrupt parties of both engines and

Context. Each party P_i owns a random key $k_i \in \{0,1\}^\lambda$. Moreover, the parties hold two authenticated secret-shared elements $[[y]]$ and $[[z]]$ which are uniformly distributed in $\mathbb{F}_p$. Finally, we assume that the parties have secret-sharings of the first $L_1 + L_2 + 1$ powers of $[[y]]$.

Encryption

Input. Cleartext additional data $(a_1, a_2, \ldots, a_{l'}) \in \mathbb{F}_p^{l'}$, secret-shared plaintext $([[x_1]], [[x_2]], \ldots, [[x_l]]) \in \mathbb{F}_p^l$ and a cleartext nonce $r \in \{0,1\}^{l_1}$.

1. The parties compute
$$[[t]] \leftarrow \mathrm{Ecd}(l) \cdot [[y^{l+r+1}]] + \sum_{i=1}^{r} a_i \cdot [[y^{l+r+1-i}]] + \sum_{i=1}^{l} [[x_i]] \cdot [[y^{l+1-i}]] + [[z]]$$
2. Each party P_i sets $\mathbf{v_i} \leftarrow (x_{1,i}, x_{2,i}, \ldots, x_{l,i}, t_i)$ where $x_{j,i}$ denotes P_i's share of $[[x_j]]$ for every $j \in [l]$.
3. Each party P_i computes $\mathbf{w_i} \leftarrow \mathrm{Trunc}(F(k_i, r), l+1)$.
4. Each party P_i broadcasts $\mathbf{c_i} \leftarrow \mathbf{w_i} + \mathbf{v_i}$.
5. The parties output $\mathbf{c} \leftarrow \mathbf{c_1} + \mathbf{c_2} + \cdots + \mathbf{c_n}$.

Decryption

Input. Cleartext additional data $(a_1, a_2, \ldots, a_{l'}) \in \mathbb{F}_p^{l'}$, non-shared ciphertext $(c_1, c_2, \ldots, c_{l+1}) \in \mathbb{F}_p^{l+1}$ and a cleartext nonce $r \in \{0,1\}^{l_1}$.

1. Each party P_i computes $\mathbf{w_i} \leftarrow F(k_i, r)$.
2. Each party P_i inputs the first $l+1$ elements of $\mathbf{w_i}$ in $\mathcal{F}_{\mathrm{MPC}}$. Let $[[w_{i,j}]]$ be a secret sharing of the j-th element of $\mathbf{w_i}$.
3. The parties compute $[[x_j]] \leftarrow c_j - \sum_{i \in [n]} [[w_{i,j}]]$ for every $j \in [l]$. Then, they set $[[t']] \leftarrow c_{l+1} - \sum_{i \in [n]} [[w_{i,l+1}]]$.
4. The parties compute
$$[[t]] \leftarrow \mathrm{Ecd}(l) \cdot [[y^{l+r+1}]] + \sum_{i=1}^{r} a_i \cdot [[y^{l+r+1-i}]] + \sum_{i=1}^{l} [[x_i]] \cdot [[y^{l+1-i}]] + [[z]]$$
5. The parties compute $b \leftarrow \mathrm{Compare}([[t]], [[t']])$. If $b = 0$, they output $\perp$, otherwise they output the identities of $[[x_1]], [[x_2]], \ldots, [[x_l]]$.

Fig. 14. Multiparty protocol for $\mathrm{MtE}(F^n, \mathrm{PolyMAC})$.

security has to be guaranteed as long as none of the endpoints is completely corrupt, n can become as big as $n_1 \cdot n_2$ where n_1 and n_2 are the cardinalities of the two MPC engines.[5]

6 Oblivious TLS

In this section, we analyse the security of Oblivious TLS. At the end, we present some general results on performance.

6.1 Security

The Multi-Stage Key Exchange Model. We want to prove the security of Oblivious TLS in the Multi-Stage Key Exchange Security Model [12]. The adversary interacts with several endpoints running the protocol and has complete control over the network. In particular, it is allowed to intercept, drop and inject communications. Moreover, it has the ability to corrupt endpoints and request the leakage of established keys. The adversary wins when it succeeds in breaking particular authentication properties (Match Security) or can distinguish a tested key from a random string of the same length (Multi-Stage Security). Since the model was used to prove the security of TLS 1.3 [12], our intention is to reduce the security of Oblivious TLS to the proof of TLS 1.3.

Adaptations to Oblivious TLS. The main difference between our protocol and the traditional version of TLS 1.3 is the use of MPC, therefore, every endpoint of our multi-stage model is actually an MPC engine. We regard them as atomic entities. Observe that we can model the actual MPC protocols for Diffie-Hellman, key derivation, signature generation and encryption and decryption with the corresponding functionalities. We allow the multi-stage adversary the same influence on the multiparty procedures as if it controlled the corrupted parties of the engine. The corruption of an MPC engine in the model corresponds in practice to the corruption of all the associated parties.

We obtain a model that is almost identical to the security model of TLS 1.3. The only differences are the following:

- The Handshake keys are leaked to the multi-stage adversary whenever there exists at least one corrupted party.
- The MAC keys used in ClientFinished and ServerFinished are leaked to the multi-stage adversary upon reception of the corresponding messages.
- The IVs are leaked to the adversary.
- If Π_{weakerDH} is adopted, the adversary is able to shift the Diffie-Hellman secret by an arbitrary elliptic curve point Q_ϵ.
- The adversary has the ability to make the engines abort.

[5] Each party of the first engine knows exactly n_2 keys, which must be distributed among all the n_2 parties of the second engine, and vice versa.

Actually, in this model, it would be trivial for the adversary to win. Indeed, it would be sufficient to test a handshake key of an engine with a corrupted party (distinguishing it from random is straightforward as the key is leaked). To fix this problem, we allow the adversary to test handshake keys only if both the endpoints of the connection are completely honest (i.e. the corresponding engines have no corrupted parties).

Security of Π_{DH}-based Oblivious TLS. In [12], the authors proved that in TLS 1.3, the application keys remain secure and authenticated even if the encryption keys are leaked. Moreover, they proved that the MAC keys used in ClientFinished and ServerFinished are computationally independent of all the other keys of the protocol. Since they are used for the last time in the verification of the MACs, the Handshake remains secure as long as they are leaked after the reception of ClientFinished and ServerFinished. Even the IVs are independent of the keys (see [24, Section 7.3]). Therefore, their knowledge does not leak any additional information. Moreover, the security of an AEAD is guaranteed as long as the keys are kept secret. As a consequence, the security proof of [12] applies to Oblivious TLS too, after minor modifications. That is enough to infer the security of our protocol.

Security of $\Pi_{weakerDH}$-based Oblivious TLS. When Π_{weakerDH} is used, the adversary is able to "shift" the Diffie-Hellman exchanged secret by an arbitrary elliptic curve point Q_ϵ. Usually, this results in a failure of the Handshake. Indeed, the MPC engine and client derive different keys, not being able to decrypt the incoming messages nor checking the MACs in ClientFinished and ServerFinished. The only exception occurs when the client is another MPC engine using Oblivious TLS and the corrupted parties insert the same error Q_ϵ.

It is possible to prove the security of the protocol following the guideline [6] of the original security proof of TLS 1.3 [12] and substituting the PRF-Oracle-Diffie-Hellman assumption (PRF-ODH) (see [12, Section 2.4]) with a new cryptographic assumption, which we are going to describe in the following paragraph.

The Shifted PRF Oracle DH assumption. The PRF-Oracle-Diffie-Hellman assumption (PRF-ODH), on which the security of TLS 1.3 is based, does not allow us to model the influence of the corrupted parties in $\mathcal{F}_{\text{weakerDH}}$. Therefore, when Π_{weakerDH} is adopted, the security proof of Oblivious TLS has to rely on the following.

Definition 2 (Shifted ODH assumption). *Let $\mathbb{G} := \{(G_\lambda, g_\lambda)\}_{\lambda\in\mathbb{N}}$ be a sequence of security-parameter-dependent groups and let $g_\lambda \in G_\lambda$ for every $\lambda \in \mathbb{N}$. Moreover, let $\mathcal{F} := \{F_\lambda : G_\lambda \times \{0,1\}^* \longrightarrow \{0,1\}^\lambda\}_{\lambda\in\mathbb{N}}$ be a sequence of functions.*

Let $\mathcal{A}$ be a PPT adversary and consider the game $\mathcal{G}^{\textit{Shifted-ODH}}_{\mathcal{F},\mathbb{G},\mathcal{A}}(\lambda)$ described in Figure 15. We define the Shifted-ODH advantage of $\mathcal{A}$ against $(\mathcal{F},\mathbb{G})$ as

$$Adv^{\textit{Shifted-ODH}}_{\mathcal{F},\mathbb{G},\mathcal{A}}(\lambda) := \left|\mathbb{P}\big(\mathcal{G}^{\textit{Shifted-ODH}}_{\mathcal{F},\mathbb{G},\mathcal{A}}(\lambda) = 1\big) - \frac{1}{2}\right|$$

[6] Observe that to achieve Match security, it is necessary to add Q_ϵ to sid_i for every stage i.

$\mathcal{G}^{\text{Shifted-ODH}}_{\mathcal{F},\mathbb{G},\mathcal{A}}(\lambda)$

Setup: The PPT adversary $\mathcal{A}$ is activated with a sequence of λ bits set to 1. Afterwards, the challenger samples a random bit b and two random elements $u, v \xleftarrow{\$} [q_\lambda - 1]$ where q_λ is the order of g_λ. Then, it sends g_λ^u and g_λ^v to $\mathcal{A}$.
Challenge: The adversary replies with a challenge $(x^*, h) \in \{0,1\}^* \times G_\lambda$. The challenger computes $y_1 \leftarrow F_\lambda(g_\lambda^{uv} \cdot h, x^*)$ and samples y_0 uniformly in $\{0,1\}^\lambda$. Then, it replies to the adversary with y_b.
Query: The adversary may issue queries of the form $(z, x, w) \in G_\lambda \times \{0,1\}^* \times G_\lambda$ such that $(z, x, w) \neq (g_\lambda^u, x^*, h)$, which the challenger answers with $F_\lambda(z^v \cdot w, x)$.
Guess: The adversary outputs a bit b'. The final output is 1 if and only if $b' = b$.

Fig. 15. The Shifted Oracle Diffie-Hellman game

We say that the Shifted-ODH assumption holds for $(\mathcal{F}, \mathbb{G})$ if for every PPT adversary $\mathcal{A}$ the advantage $Adv^{Shifted\text{-}ODH}_{\mathcal{F},\mathbb{G},\mathcal{A}}(\lambda)$ is negligible in λ.

Essentially the Shifted ODH assumption models a Diffie-Hellman key exchange in which the adversary shifts the exchanged secret by and arbitrary element of the group and an entropy extractor is applied to the result. The assumption holds if the adversary is not able to distinguish between the real output and a random sequence of the same length even if it has access to results of other key exchanges. In our opinion, the assumption is reasonable as long as the Diffie-Hellman problem is "hard" over $\mathbb{G}$ and $\mathcal{F}$ "hides" the correlation between different inputs. Indeed, if the adversary cannot distinguish between g^{uv} and a random element of the group, then, it cannot distinguish between $g^{uv} \cdot h$ and a random element of the coset $\langle g \rangle \cdot h$, which has the same entropy. The condition on $\mathcal{F}$ is usually satisfied when $\mathcal{F}$ is based on hash functions. This property holds in Oblivious TLS (see [24, Section 7.1]).

6.2 Performance

To estimate the performance of Oblivious TLS, we carried out some benchmarks using the SCALE-MAMBA library[7], based on implementation of the main components in the handshake and record layers, which we believe to be the bottleneck. This does not give a full, standards-compliant implementation of Oblivious TLS, but rather, is intended to obtain some estimates of its expected performance.

We tested the online phase of the resulting MPC protocol, assuming the necessary input-independent preprocessing (multiplication triples etc.) has been generated. Based on the results, we can expect a Handshake to take around 1 or 2 seconds. We also tested the throughput of the different multiparty AEADs we considered. The MPC-friendly AEAD showed interesting outcomes with a

[7] `https://github.com/KULeuven-COSIC/SCALE-MAMBA`

throughput of around 300 KB/s. The Garbled-Circuit-based version of AES-GCM, instead, proved itself to be rather inefficient (around 1 KB/s). We expect that with the alternative AES evaluation method based on [18] (which is not available in SCALE-MAMBA), we could achieve throughputs up to 3 MB/s for AES-GCM, even exceeding our MPC-friendly AEAD. The drawback of this is that the preprocessing material is much more expensive to generate, and also the round complexity is higher. For further information on performance, see Appendix F.

Acknowledgements

We would like to thank Roberto Zunino for suggestions and comments on Damiano Abram's master's thesis. The work of Sven Trieflinger and Damiano Abram was funded by Robert Bosch GmbH. Ivan Damgård was supported by the European Research Council (ERC) under the European Unions's Horizon 2020 research and innovation programme under grant agreement No 669255 (MPCPRO). Peter Scholl was supported by a starting grant from the Aarhus University Research Foundation.

References

1. Scale-mamba software, `https://homes.esat.kuleuven.be/~nsmart/SCALE`
2. Abram, D.: Oblivious TLS. Master's thesis, Università degli Studi di Trento (March 2020)
3. Agrawal, S., Mohassel, P., Mukherjee, P., Rindal, P.: DiSE: Distributed symmetric-key encryption. In: ACM CCS 2018. ACM Press (Oct 2018)
4. Bendlin, R., Damgård, I., Orlandi, C., Zakarias, S.: Semi-homomorphic encryption and multiparty computation. In: EUROCRYPT 2011. LNCS, Springer, Heidelberg (May 2011)
5. Bernstein, D.J.: Curve25519: New Diffie-Hellman speed records. In: PKC 2006. LNCS, Springer, Heidelberg (Apr 2006)
6. Boyle, E., Couteau, G., Gilboa, N., Ishai, Y., Kohl, L., Scholl, P.: Efficient pseudorandom correlation generators from ring-LPN. In: CRYPTO 2020, Part II. LNCS, Springer, Heidelberg (Aug 2020)
7. Canetti, R.: Universally composable security: A new paradigm for cryptographic protocols. In: 42nd FOCS. IEEE Computer Society Press (Oct 2001)
8. Dalskov, A.P.K., Orlandi, C., Keller, M., Shrishak, K., Shulman, H.: Securing DNSSEC keys via threshold ECDSA from generic MPC. In: ESORICS 2020, Part II. LNCS, Springer, Heidelberg (Sep 2020)
9. Damgård, I., Damgård, K., Nielsen, K., Nordholt, P.S., Toft, T.: Confidential benchmarking based on multiparty computation. In: FC 2016. LNCS, Springer, Heidelberg (Feb 2016)
10. Damgård, I., Keller, M., Larraia, E., Pastro, V., Scholl, P., Smart, N.P.: Practical covertly secure MPC for dishonest majority - or: Breaking the SPDZ limits. In: ESORICS 2013. LNCS, Springer, Heidelberg (Sep 2013)
11. Damgård, I., Pastro, V., Smart, N.P., Zakarias, S.: Multiparty computation from somewhat homomorphic encryption. In: CRYPTO 2012. LNCS, Springer, Heidelberg (Aug 2012)

12. Dowling, B., Fischlin, M., Günther, F., Stebila, D.: A Cryptographic Analysis of the TLS 1.3 Handshake Protocol. Cryptology ePrint Archive, Report 2020/1044 (2020)
13. Dworkin, M.: Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC. Tech. rep. (2007)
14. Furukawa, J., Lindell, Y., Nof, A., Weinstein, O.: High-throughput secure three-party computation for malicious adversaries and an honest majority. In: EUROCRYPT 2017, Part II. LNCS, Springer, Heidelberg (Apr / May 2017)
15. Gentry, C., Halevi, S., Smart, N.P.: Homomorphic evaluation of the AES circuit. In: CRYPTO 2012. LNCS, Springer, Heidelberg (Aug 2012)
16. Grassi, L., Rechberger, C., Rotaru, D., Scholl, P., Smart, N.P.: MPC-friendly symmetric key primitives. In: ACM CCS 2016. ACM Press (Oct 2016)
17. Hazay, C., Scholl, P., Soria-Vazquez, E.: Low cost constant round MPC combining BMR and oblivious transfer. In: ASIACRYPT 2017, Part I. LNCS, Springer, Heidelberg (Dec 2017)
18. Keller, M., Orsini, E., Rotaru, D., Scholl, P., Soria-Vazquez, E., Vivek, S.: Faster secure multi-party computation of AES and DES using lookup tables. In: ACNS 17. LNCS, Springer, Heidelberg (Jul 2017)
19. Keller, M., Orsini, E., Scholl, P.: MASCOT: Faster malicious arithmetic secure computation with oblivious transfer. In: ACM CCS 2016. ACM Press (Oct 2016)
20. Keller, M., Pastro, V., Rotaru, D.: Overdrive: Making SPDZ great again. In: EUROCRYPT 2018, Part III. LNCS, Springer, Heidelberg (Apr / May 2018)
21. Krawczyk, H.: The order of encryption and authentication for protecting communications (or: How secure is SSL?). In: CRYPTO 2001. LNCS, Springer, Heidelberg (Aug 2001)
22. Krawczyk, H.: Cryptographic extraction and key derivation: The HKDF scheme. In: CRYPTO 2010. LNCS, Springer, Heidelberg (Aug 2010)
23. Nielsen, J.B., Nordholt, P.S., Orlandi, C., Burra, S.S.: A new approach to practical active-secure two-party computation. In: CRYPTO 2012. LNCS, Springer, Heidelberg (Aug 2012)
24. Rescorla, E.: The Transport Layer Security (TLS) Protocol Version 1.3. RFC 8446 (2018)
25. Rotaru, D., Smart, N.P., Stam, M.: Modes of operation suitable for computing on encrypted data. IACR Trans. Symm. Cryptol. (3) (2017)
26. Rotaru, D., Smart, N.P., Tanguy, T., Vercauteren, F., Wood, T.: Actively secure setup for SPDZ. Cryptology ePrint Archive, Report 2019/1300 (2019), `https://eprint.iacr.org/2019/1300`
27. Zhang, F., Maram, D., Malvai, H., Goldfeder, S., Juels, A.: DECO: Liberating Web Data Using Decentralized Oracles for TLS. In: Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security (2020)

A Security Model and Resources

We are going to prove the security of our protocols in the UC model of Canetti [7]. We assume to deal with an active secure adversary that can corrupt up to $n-1$ parties. The set of dishonest parties is decided by the adversary at the beginning and it can never be changed afterwards. We also assume that the parties are connected by point to point secure channels and they have access to a broadcast

medium. When any functionality aborts, we assume that it communicated the abortion to every honest party as well as to the adversary before halting.

Let S be a security parameter dependent set of finite fields with the following properties

- $\mathbb{F}_2 \in S$.
- There exists a unique $k \in \mathbb{N}$ such that $\mathbb{F}_{2^k} \in S$.
- $S \setminus \{\mathbb{F}_2, \mathbb{F}_{2^k}\}$ contains only prime fields of characteristic sufficiently close to a power of 2.
- The value of $\chi := \max\{|\mathbb{F}|^{-1} \mid \mathbb{F} \in S \setminus \{\mathbb{F}_2\}\}$ is negligible in the security parameter.

Moreover, let (E, G) be a security parameter dependent pair with the following properties

- E is an elliptic curve defined over a prime field in $S \setminus \{\mathbb{F}_2\}$.
- G is an affine point of E of prime order $q \neq 2$ such that $q^2 \nmid |E|$.
- $\mathbb{F}_q \in S$.

Observe that if $q^2 \nmid |E|$, it is easy to verify if a point $Q \in E$ belongs to $\langle G \rangle$. As a matter of fact, it is sufficient to check whether $qQ = \infty$ (∞ denotes the identity element of the group). Notice indeed that $\langle G \rangle$ is the only subgroup of order q in E.

We can assume that the parties have access to the functionality $\mathcal{F}_{\text{MPC}}$ described in Figure 16. Observe that we abused the notation and in $\mathcal{F}_{\text{MPC}}$ we used $[[x]]$ to denote the identity of x. We also assume that the identities are generated deterministically and the parties can derive the identity of the output of every procedure without any interaction.

B Security Proofs of Π_{DH}

In this appendix, we will denote the number of honest parties by H. Moreover, we will assume $\mathcal{H}$ to be $\{h_i \mid i \in [H]\}$ with $h_1 < h_2 < \cdots < h_H$. For every $i \in H$, we also denote the number of corrupted parties between the i-th and the $i+1$-th honest party by c_i. Specifically, $c_i := h_{i+1} - h_i$ for every $i \in [H-1]$, whereas $c_H := n - h_H$.

We indicate the complement of a set S as $S^{\complement}$, whereas $\mathcal{P}(S)$ denotes the power set. Moreover, given a finite set U, a probability distribution μ over the measurable space $(U, \mathcal{P}(U))$ and a function $F : U \to V$, we represent with μ_F the distribution induced by μ and F over V.

We assume to deal with an elliptic curve $E : Y^2 = X^3 + AX^2 + BX + C$ over the field $\mathbb{F}_p$. Moreover, we suppose that the base point point of the Diffie-Hellman algorithm is an affine point G of E and we assume that its order is an odd prime number q. Before proving the security of the multiparty Diffie-Hellman protocols, we define the following operation.

$\mathcal{F}_{\text{MPC}}$

Input. On input $(\text{Input}, \mathbb{F}, i, x)$ from P_i and $(\text{Input}, \mathbb{F}, i)$ from all the other parties, the functionality checks that $\mathbb{F} \in S$ and $x \in \mathbb{F}$. If this is the case, it stores x with identity $[[x]]$ and domain $\mathbb{F}$.

Polynomial. On input $(\text{Poly}, \mathbb{F}, f, m, [[x_1]], [[x_2]], \ldots, [[x_m]])$ from every party, the functionality checks that $\mathbb{F} \in S$, $f \in \mathbb{F}[X_1, X_2, \ldots, X_m]$ and the domain of $[[x_i]]$ is $\mathbb{F}$ for every $i \in [m]$. If this is the case, it stores $y \leftarrow f(x_1, x_2, \ldots, x_m)$ with domain $\mathbb{F}$ and identity $[[y]]$.

Division. On input $(\text{Div}, \mathbb{F}, [[x_1]], [[x_2]])$ from every party, the functionality checks that $\mathbb{F} \in S$ and the domain of $[[x_i]]$ is $\mathbb{F}$ for every $i \in [2]$. If this is the case and $x_2 \neq 0$, it stores $y \leftarrow x_1/x_2$ with domain $\mathbb{F}$ and identity $[[y]]$. Then, it outputs OK to the adversary. If instead $x_2 = 0$, it outputs ZD to every party.

Random Input. On input $(\text{RandInput}, \mathbb{F}, i)$ from every party, the functionality checks that $\mathbb{F} \in S$. If this is the case, it stores $y \xleftarrow{\$} \mathbb{F}$ with domain $\mathbb{F}$ and identity $[[y]]$ and outputs y to party P_i.

Random. On input $(\text{Rand}, \mathbb{F})$ from every party, the functionality checks that $\mathbb{F} \in S$. If this is the case, it stores $y \xleftarrow{\$} \mathbb{F}$ with domain $\mathbb{F}$ and identity $[[y]]$.

EC Addition. On input $(\text{ECAdd}, [[Q_1]], [[Q_2]])$ from every party, the functionality checks that the domain of $[[Q_1]]$ and $[[Q_2]]$ is E. If this is the case, it stores $Q_3 \leftarrow Q_1 + Q_2$ with domain E and identity $[[Q_3]]$.

EC Multiplication. On input $(\text{ECMult}, [[Q_1]], a)$ from every party, the functionality checks that the domain of $[[Q_1]]$ is E and $a \in \mathbb{F}_q$ where q is the order of G. If this is the case, it stores $Q_2 \leftarrow a \cdot Q_1$ with domain E and identity $[[Q_2]]$.

EC Embedding. On input $(\text{ECEmbed}, [[a]], Q)$ from every party, the functionality checks that $Q \in \langle G \rangle$ and the domain of $[[a]]$ is $\mathbb{F}_q$ where q is the order of G. If this is the case, it stores $Q_2 \leftarrow a \cdot Q$ with domain E and identity $[[Q_2]]$.

ToBits. On input $(\text{ToBits}, [[x]])$ from every party, the functionality computes $b_0, b_1, \ldots, b_l$ such that $x = \sum_{i=0}^{l} b_i \cdot 2^i$ (or $x = \bigoplus_{i=0}^{l} b_i \cdot \alpha^i$ if the domain of $[[x]]$ is $\mathbb{F}_{2^k} \cong \mathbb{F}_2[\alpha]$). Then, it stores b_i with domain $\mathbb{F}_2$ and identity $[[b_i]]$ for every $i \in \{0, 1, \ldots, l\}$.

ToField. On input $(\text{ToField}, [[x]], \mathbb{F})$ from every party, the functionality checks that $\mathbb{F} \in S$ and the domain of $[[x]]$ is $\mathbb{F}_2$. If this is the case, it stores $y \leftarrow x$ with domain $\mathbb{F}$ and identity $[[y]]$.

Output. On input $(\text{Output}, [[x]])$ from every party, the functionality sends x to the adversary and waits for a reply. If the answer is OK, the functionality outputs x to every honest party. Otherwise, it aborts.

Abort. On input (Abort) from the adversary, the functionality aborts.

Fig. 16. The multiparty computation functionality

Definition 3. *Let $\boxplus$ be the operation defined as follows*

$$\boxplus : E \cup \{\perp\} \times E \longrightarrow E \cup \{\perp\}$$

$$Q_1 \boxplus Q_2 := \begin{cases} \perp & \text{if } Q_1 = \infty \text{ or } Q_2 = \infty \\ \perp & \text{if } Q_1 = \perp \\ \perp & \text{if } Q_1 \text{ are affine points with the same } x \text{ coordinate} \\ Q_1 + Q_2 & \text{otherwise} \end{cases}$$

Observe that $\boxplus$ represents the result of formula (1) (see Section 4). As a matter of fact, the operation succeeds if and only if the two addends are affine points with different x coordinates. Notice that the associative property does not hold for $\boxplus$.

B.1 Security of $\Pi_{\mathbf{daPoint}}$

We start with a lemma that is used in the security proofs of both Π_{daPoint} and Π_{ECSum}.

Lemma 1. *Let $C_i \in E \setminus \{\infty\}$ for every $i \in \mathcal{C}$ and let $T \in E$. Assume that*

$$B := (((C_1 \boxplus C_2) \boxplus C_3) \boxplus \dots) \boxplus C_{h_1 - 1} \neq \perp.$$

Supposing q sufficiently big, there exists a sequence of points $(R_i)_{i \in [n]}$ such that

$$\begin{cases} (((R_1 \boxplus R_2) \boxplus R_3) \boxplus \dots) \boxplus R_n = T, & \\ R_i \in \langle G \rangle & \forall i \in \mathcal{H} \\ R_i = C_i & \forall i \in \mathcal{C} \end{cases} \tag{2}$$

if only if one of the following holds

- *$H > 1$, $T \in \sum_{i \in \mathcal{C}} C_i + \langle G \rangle$ and*

$$T \notin \Omega := \left\{ \sum_{i=\iota}^{c_H} C_{h_H + i} \mid \iota \in [c_H] \right\} \cup \left\{ \sum_{i=\iota}^{c_H} C_{h_H + i} + C_{h_H + \iota} \mid \iota \in [c_H] \right\} \cup \left\{ \infty \right\}$$

- *$H = 1$, $T \in \sum_{i \in \mathcal{C}} C_i + \langle G \rangle$ and*

$$T \notin \Omega' := \Omega \cup \left\{ \sum_{i \in \mathcal{C}} C_i, \sum_{i \in \mathcal{C}} C_i + B \right\}$$

Moreover, if the above condition is satisfied,

- *if $H > 1$, there exist at least*

$$\prod_{i=1}^{H-1} (q - 3 - 2c_i) - 2(q-1)^{H-2}$$

different sequences $(R_i)_{i \in [n]}$ for which (2) holds.

– *if $H = 1$, there exists exactly one sequence $(R_i)_{i\in[n]}$ for which (2) holds.*

Proof. We start by observing that $Q_1 \boxplus Q_2 \neq \infty$ for every $Q_1, Q_2 \in \langle G\rangle$. As a matter of fact, $Q_1 + Q_2 = \infty$ if and only if $Q_1 = -Q_2$ and therefore the two points would have the same x coordinate. In such case, $Q_1 \boxplus Q_2 = \perp$.

For each sequence $(R_i)_{i\in[n]}$, we define $T_0 := \infty$ and $T_i := T_{i-1} + R_i$ for every $i \in [n]$. Observe that the following relation holds

$$(((R_1 \boxplus R_2) \boxplus R_3) \boxplus \dots) \boxplus R_n \neq \perp \tag{3}$$

if and only if $R_i \neq \infty$ and $R_i \neq \pm T_{i-1}$ for every $i \in [n]$. Moreover, if $R_i = C_i$ for every $i \in \mathcal{C}$, it is necessary and sufficient that, for every $i \in [H]$,

B1. $R_{h_i} \neq \infty$
B2. $R_{h_i} \neq \pm T_{h_i-1}$
B3. $R_{h_i} \neq \pm C_{h_i+\iota} - T_{h_i-1} - \sum_{j=1}^{\iota-1} C_{h_i+j}$ for every $\iota \in [c_i]$.

Indeed, if the last inequality was not satisfied for any $\iota \in [c_i]$, we would have $R_{h_i+\iota} = C_{h_i+\iota} = \pm T_{h_i+\iota-1}$. We can conclude that we have at least $q - 3 - 2c_i$ possible ways to choose R_{h_i} in order for (3) to hold.

Observe that if inequality (3) is satisfied,

$$(((R_1 \boxplus R_2) \boxplus R_3) \boxplus \dots) \boxplus R_n = \sum_{i\in[n]} R_i.$$

Therefore, if we want to satisfy (2),

$$R_{h_H} = T - T_{h_H-1} - \sum_{i=1}^{c_H} C_{h_H+i} =: W - T_{h_H-1}. \tag{4}$$

If $T \notin \sum_{i\in\mathcal{C}} C_i + \langle G\rangle$, $W - T_{h_H-1}$ would not belong to $\langle G\rangle$ and therefore (4) as well as (2) would not hold. Furthermore, (4) might be incompatible with the other conditions that R_{h_H} must satisfy in order for inequality (3) to hold. Namely:

C1. From B1, $R_{h_H} = W - T_{h_H-1} \neq \infty$.
This condition does not hold if and only if $T_{h_H-1} = W$.
C2. From B2, $R_{h_H} = W - T_{h_H-1} \neq T_{h_H-1}$.
This condition does not hold if and only if $2T_{h_H-1} = W$.
C3. From B2, $R_{h_H} = W - T_{h_H-1} \neq -T_{h_H-1}$.
This conditions does not hold if and only if $T = \sum_{i=1}^{c_H} C_{h_H+i}$.
C4. From B3, $R_{h_H} = W - T_{h_H-1} \neq \pm C_{h_H+\iota} - T_{h_H-1} - \sum_{i=1}^{\iota-1} C_{h_H+i}$ for every $\iota \in [c_H]$. This condition does not hold if and only if

$$T = \sum_{i=1}^{c_H} C_{h_H+i} - \sum_{i=1}^{\iota-1} C_{h_H+i} \pm C_{h_H+\iota} \quad \text{for any } \iota \in [c_H].$$

Observe that $T \in \Omega$ if and only if either C3 or C4 does not hold. Moreover, if $H = 1$, $T \in \Omega' \setminus \Omega$ if and only if either C1 or C2 is not satisfied (observe that in this case, $T_{h_H-1} = B$). If $H = 1$ and $T \notin \Omega'$, we know that C1, C2, C3, C4 are satisfied, and therefore $R_{h_H} = W - B$ is the only choice that make equality (2) hold.

If instead $H > 1$, we can try to generate the sequence $(R_i)_{i\in[n]}$ by choosing for $j = 1, 2, \ldots, H-1$, a random R_{h_j} that satisfies B1, B2 and B3. Then, we set $R_{h_H} \leftarrow W - T_{h_H-1}$. Clearly, $R_i = C_i$ for every $i \in \mathcal{C}$. In this way, we obtain $\prod_{i=1}^{H-1}(q - 3 - 2c_i)$ different sequences. Observe anyway, that not all of them satisfy (2). As a matter of fact, C1 and C2 might not hold for some of them.

Let W' be the only point of

$$\sum_{\substack{i\in\mathcal{C}\\ i<h_H}} C_i + \langle G\rangle$$

such that $2W' = W$ (observe that W' is unique since the order of $\langle G\rangle$ is an odd prime). It may be possible that W' does not exist. We now count the problematic sequences. Notice that all of them are contained in the set

$$\Big\{(R_i)_{i\in[n]} \mid R_i = C_i \quad \forall i \in \mathcal{C},$$
$$R_{h_i} \in \langle G\rangle \setminus \{\infty\} \quad \forall i \in [H-2],$$
$$R_{h_{H-1}} + \sum_{i=1}^{c_{H-1}} C_{h_{H-1}+i} + T_{h_{H-1}-1} \in \{W, W'\},$$
$$R_{h_H} = W - T_{h_H-1}\Big\}$$

This set has at most $2(q-1)^{H-2}$ elements. Therefore, we have proven that if $T \notin \Omega$, there exist at least $\prod_{i=1}^{H-1}(q - 3 - 2c_i) - 2(q-1)^{H-2}$ sequences that satisfy equation (2). □

We start by proving the following lemma.

Lemma 2. *Assume that*

$$\omega' := \frac{M + N \cdot l}{q}$$

is a negligible function in the security parameter $\lambda \in \mathbb{N}$. If all the parties behave honestly in $\Pi_{daPoint}$, the probability that the protocol aborts is negligible.

Proof. Since all the parties behave honestly, an abortion occurs if and only if a zero denominator is found or if there exist $i \in [n]$ and $j \in [M + Nl]$ such that $Z_{i,j} = \infty$. Let E_1 be the event in which there exist no $i \in [n]$ and $j \in [M + Nl]$ such that $Z_{i,j} = \infty$. The probability of E_1 is

$$\left(\frac{q-1}{q}\right)^{n(M+Nl)}$$

We now work conditioned on E_1. Fix a $k \in [Nl]$ and let $j := \psi(k)$. Let $T_0 = \infty$ and for every $i \in [n]$ set $T_i := T_{i-1} + Z_{i,j}$. The computation of R_j fails due to a zero denominator if and only if $Z_{i,j} = \pm T_{i-1}$ for some $i \in [n] \setminus \{1\}$.

Claim. Assuming that no zero denominator occurs, $T_i \in \langle G \rangle \setminus \{\infty\}$ for every $i \in [n]$. Therefore, $T_i \neq -T_i$ for each $i \in [n]$.

This is trivially true for $i = 1$. For $i > 1$, observe that T_i is the sum of elements of $\langle G \rangle$ and therefore it belongs to $\langle G \rangle$ too. Moreover, if $T_i = \infty$, then $Z_{i,j} = -T_{i-1}$ which would cause a zero denominator in the previous step of the computation. If $T_i = -T_i$, $2T_i = \infty$. Since all the elements of $\langle G \rangle \setminus \{\infty\}$ have order $q \neq 2$, this condition cannot hold.

We now try to count the sequences $(Z_{i,j})_{i \in [n]}$ that do not cause zero denominators. If we start from $i = 1$ and we gradually select $Z_{i,j} \in \langle G \rangle \setminus \{\infty, T_{i-1}, -T_{i-1}\}$, we understand that there exist $(q-1)(q-3)^{n-1}$ sequences that do not cause zero denominators.

Let E_2 be the event in which no zero denominator is found computing $R_{\psi(k)}$ for $k \in [Nl]$. Conditioned on E_1, the probability of E_2 is

$$\mathbb{P}(E_2 \mid E_1) = \frac{\Big((q-1)(q-3)^{n-1}\Big)^{Nl}}{(q-1)^{nNl}} = \left(\frac{q-3}{q-1}\right)^{(n-1)Nl}$$

Conditioned on $E_1 \cap E_2$, the protocol aborts only if there exist $i \in [N]$ and $j \in [l] \setminus \{1\}$ such that $R_{i,1} = \pm R_{i,j}$. Let E_3 be the event in which such situation never occurs.

Claim. Conditioned on $E_1 \cap E_2$, $R_{i,j}$ is uniformly distributed in $\langle G \rangle \setminus \{\infty\}$ for every $i \in [N]$ and $j \in [l]$.

That means that the probability of E_3 conditioned on $E_1 \cap E_2$ is

$$\left(\frac{q-3}{q-1}\right)^{N(l-1)}$$

We now prove the claim. Fix $i \in [N]$ and $j \in [l]$. Let $\iota := \psi((i-1) \cdot l + j)$ and notice that $R_{i,j} = R_\iota$. As we did before, define $T_0 := \infty$ and $T_k := T_{k-1} + Z_{k,\iota}$. We start observing that $R_\iota \in \langle G \rangle \setminus \{\infty\}$. This follows from the first claim of this proof and the fact that $R_\iota = T_n$.

We prove the claim by induction on n. Let T be an element of $\langle G \rangle \setminus \{\infty\}$. We count the number of sequences $(Z_{k,\iota})_{k \in [n]}$ that satisfy E_1 and E_2 and such that $\sum_{k \in [n]} Z_{k,\iota} = T$. Let this number be $N_{n,T}$.

For $n = 1$, the claim is trivial. Observe that $N_{1,T} = 1$ for every T, indeed, $Z_{1,\iota} = T$ is the only option.

Suppose the claim is true for $n = h - 1$, we prove it for $n = h$. Let T be in $\langle G \rangle \setminus \{\infty\}$. Starting from $k = 1$ until $n - 1$, we gradually select $Z_{k,\iota}$ in $\langle G \rangle \setminus \{\infty, T_{k-1}, -T_{k-1}\}$. For $k = 1$, we have $q - 1$ possible choices as $T_0 = -T_0 = \infty$. For $k > 1$, we have instead $q - 3$ choices (this is a consequence of the

first claim of this proof). For $Z_{n,\iota}$ there is at most one possibility i.e. $T - T_{n-1}$. Observe that sometimes there is no way to choose $Z_{n,\iota}$, indeed $T - T_{n-1}$ might be in $\{\infty, T_{n-1}, -T_{n-1}\}$. Actually, $T - T_{n-1} \neq -T_{n-1}$ otherwise $T = \infty$. Therefore, the problematic cases are $T_{n-1} = T$ and $T_{n-1} = T/2$ (the division by 2 is well defined in $\langle G \rangle$ since q is coprime with 2). Observe that $T \neq \infty$ implies that T and $T/2$ are distinct.

Therefore, the number of sequences of $(Z_{k,\iota})_{k\in[h]}$ such that $E_1 \cap E_2$ holds and $\sum_{k\in[h]} Z_{k,\iota} = T$ is

$$N_{h,T} = (q-1)\cdot(q-3)^{h-2} - N_{h-1,T} - N_{h-1,T/2}$$

As a matter of fact, we just have to count all the possible sequences of the first $n-1$ points and subtract those whose sum is T or $T/2$. By inductive hypothesis, $N_{h-1,T}$ and $N_{h-1,T/2}$ are independent of T, therefore, $N_{h,T}$ is independent of T too. This is sufficient to prove the claim.

In conclusion, the probability of aborting is

$$\begin{aligned}
&\mathbb{P}(E_1^{\complement}) + \left(\mathbb{P}(E_2^{\complement} \mid E_1) + \mathbb{P}(E_3^{\complement} \mid E_1 \cap E_2)\cdot\mathbb{P}(E_2 \mid E_1)\right)\cdot\mathbb{P}(E_1) \leq \\
\leq &\mathbb{P}(E_1^{\complement}) + \mathbb{P}(E_2^{\complement} \mid E_1) + \mathbb{P}(E_3^{\complement} \mid E_1 \cap E_2) = \\
= &1 - \left(\frac{q-1}{q}\right)^{n(M+Nl)} + 1 - \left(\frac{q-3}{q-1}\right)^{(n-1)Nl} + 1 - \left(\frac{q-3}{q-1}\right)^{N(l-1)} \leq \\
\leq &\frac{n(M+Nl)}{q} + \frac{2(n-1)Nl}{q-1} + \frac{2N(l-1)}{q-1}
\end{aligned}$$

Observe that the last term is negligible if ω' is negligible. □

We present here the proof of theorem 1.

Proof. Consider the simulator $\mathcal{S}_{\text{daPoint}}$ described in Figure 17.

$\mathcal{S}_{\text{daPoint}}$

MPC
The simulator checks that the identities specified in the queries correspond to a stored value. If this is the case, it simply forwards the communications between the adversary and the functionality.

daPoint
The simulator runs the protocol with the adversary. If an abortion occurs, the simulator sends (Abort) to the functionality. Otherwise, it sends $(\text{daPoint}, (\text{id}_{i,1}, \text{id}_{i,2}, \text{id}_{i,3})_{i\in[N]})$ to the functionality.

Fig. 17. The daPoint Simulator

Observe that the protocol aborts with the same distribution of the simulation. In particular, if a zero denominator occurs or any party P_i (even a corrupted

one) claims that there exists $j \in [M + Nl]$ such that $Z_{i,j} = \infty$, the protocol always aborts and the simulation is perfect. The same reasoning applies when there exist $i \in [n]$ and $j \in [M + Nl]$ such that $(x_{i,j}, y_{i,j}) \notin E$. Indeed, if the incorrect point is opened in step 5, the protocol aborts. Moreover, in the lucky case in which the point passes the check, the protocol aborts in the following step when the equation of the curve $Y^2 = X^3 + AX^2 + BX + C$ is checked on all the non-opened points. Therefore, consider the event E_1 in which none of the previous cases occurs. Let $\mathcal{Y}$ be the event in which the protocol succeeds and observe that $\mathbb{P}(\mathcal{Y} \mid E_1^{\complement}) = 0$.

Now consider the protocol execution and define

$$\begin{aligned} R_j &:= \sum_{i \in [n]} Z_{i,j} && \text{for } j \in [M + Nl] \\ Z'_{i,j} &:= (x_{i,j}, y_{i,j}) && \text{for } i \in [n] \text{ and } j \in [M + Nl] \\ R'_j &:= \sum_{i \in [n]} Z'_{i,j} && \text{for } j \in [M + Nl]. \end{aligned}$$

We start by showing that if there exists any $j \in [M + Nl]$ such that $R'_j \neq R_j$ conditioned on E_1, then the protocol aborts with overwhelming probability (i.e. the probability of not aborting is negligible). Let $S := \{j \in [M + Nl] \mid R'_j \neq R_j\}$.

The first observation is that if $R'_j \neq R_j$, there exists at least one $i \in [n]$ such that $Z'_{i,j} \neq Z_{i,j}$. Therefore, if there exist more than Nl elements in S, the protocol aborts with probability 1 at step 5.

For each $i \in [N]$ and $j \in [l]$, let $f(i, j) := \psi((i - 1)l + j)$.

The second observation is that, conditioned on E_1, the protocol does not abort only if every bucket is either contained in S or in $S^{\complement}$. Indeed, if there exist $j, k \in \mathcal{B}_i$ such that $R'_j = R_j$ and $R'_k \neq R_k$, then, there exists $\iota \in \mathcal{B}_i$ such that $R_{f(i,1)} + R_\iota \neq R'_{f(i,1)} + R'_\iota$. In particular,

$$\iota = \begin{cases} k & \text{if } R_{f(i,1)} = R'_{f(i,1)} \\ j & \text{if } R_{f(i,1)} \neq R'_{f(i,1)} \end{cases}$$

We have shown that if $|S| > Nl$ or $l \nmid |S|$, the probability of an abortion is 1. We now analyse what is the probability of aborting in the other cases, i.e. when $|S| = rl$ with $0 < r \leq N$.

We consider the possible permutations ψ that would make the protocol succeed conditioned on E_1. Let their set be Σ. We can represent each permutation as a sequence of $M + Nl$ non-repeated numbers in $[M + Nl]$. The j-th number of the sequence represents the image of j. The i-th bucket is the sequence of elements from position $(i-1)l+1$ to il. The permutations that cause no abortion have to send all the elements of S in r out of the first N buckets. There are $\binom{N}{r}$ ways of choosing these buckets, $(rl)!$ ways of permuting the elements in S and $(M + Nl - rl)!$ ways of permuting the remaining elements. Therefore, the

probability of picking any of the permutations that cause no abortion is

$$\mathbb{P}(\psi \in \Sigma \mid |S| = rl, E_1) \leq \frac{\binom{N}{r} \cdot (rl)! \cdot (M + Nl - rl)!}{(M + Nl)!} = \binom{N}{r} \binom{M + Nl}{rl}^{-1}$$

As it was proven in [14, Section 5], for every $r \in [N]$

$$\binom{N}{r} \binom{M + Nl}{rl}^{-1} \leq N \cdot \binom{M + Nl}{l}^{-1} = \omega$$

As a consequence, $\mathbb{P}(\mathcal{Y} \mid S \neq \emptyset, E_1) \leq \omega(\lambda)$ and therefore, the protocol succeeds with negligible probability when $S \neq \infty$. In conclusion, if we prove that no adversary can distinguish the output of the protocol from the output of the functionality when $S = \emptyset$ and E_1 holds, the theorem is proven.

Recall lemma 1, and observe that the points of the adversary all belong to $\langle G \rangle$, therefore, we do not need to care about cosets. For every $j \in [M+Nl]$, define Ω_j to be the set of values $T \in \langle G \rangle$ that R_j cannot assume without causing a zero denominator in step 7. In general, we have a different Ω_j for every $j \in [M+Nl]$. Observe that $|\Omega_j| \leq 2c_H + 3$.

Consider now a bucket $\mathcal{B}_i$. Conditioned on the knowledge of the values $\{W_{i,j} \mid j = 2, 3, \ldots, l\}$, the adversary knows that

- $R_{f(i,1)} \notin \Omega_{f(i,1)}$
- $W_{i,j} - R_{f(i,1)} = R_{f(i,j)} \notin \Omega_{f(i,j)}$ for $j = 2, 3, \ldots, l$.
- $R_{f(i,1)} \neq W_{i,j}/2$ for $j = 2, 3, \ldots, l$ (observe that the division by 2 is well defined since the order of $\langle G \rangle$ is an odd prime). If this condition was not satisfied, a zero denominator would occur when $R_{f(i,1)}$ and $R_{f(i,j)}$ are added.

In other words, $R_{f(i,1)}$ can assume at least $\Lambda := q - l(2c_H + 4) + 1$ different values. By lemma 1, given any of such values T for $R_{f(i,1)}$, we have at least

$$\rho := \begin{cases} \left(\prod_{j=1}^{H-1}(q - 3 - 2c_j) - 2(q-1)^{H-2}\right)^l & \text{if } H > 1 \\ 1 & \text{otherwise} \end{cases}$$

ways to choose $(Z_{k,f(i,j)})_{k \in \mathcal{H}, j \in [l]}$ that respect the view of the adversary and such that $R_{f(i,1)} = T$.

We now show that the distribution μ of $(Z_{k,f(i,j)})_{k \in [n], j \in [l]}$ in the protocol is indistinguishable from the uniform distribution ν over the set

$$\begin{aligned} \Xi_1 := \Big\{ (Q_{k,f(i,j)})_{k \in [n], j \in [l]} \mid\ & Q_{k,f(i,j)} = R_{k,f(i,j)} \quad \forall (k,j) \in \mathcal{C} \times [l], \\ & Q_{k,f(i,j)} \in \langle G \rangle \quad \forall (k,j) \in \mathcal{H} \times [l], \\ & \sum_{k=1}^{n} Q_{k,f(i,1)} + \sum_{k=1}^{n} Q_{k,f(i,j)} = W_{i,j} \quad \forall j \in [l] \Big\}. \end{aligned}$$

Define Ξ_2 to be set of sequences $(Z_{k,f(i,j)})_{k \in [n], j \in [l]}$ that respect the view of the adversary and observe that μ is the uniform distribution over Ξ_2. Since

$\Xi_2 \subseteq \Xi_1$, the statistical distance between μ and ν is

$$d(\mu,\nu) = \frac{|\Xi_1| - |\Xi_2|}{|\Xi_1|} = \frac{q^{l(H-1)+1} - |\Xi_2|}{q^{l(H-1)+1}}$$

Suppose that $H > 1$. In such case,

$$d(\mu,\nu) = \frac{q^{l(H-1)+1} - |\Xi_2|}{q^{l(H-1)+1}} \leq \frac{q^{l(H-1)+1} - \Lambda \cdot \rho}{q^{l(H-1)+1}} =$$

$$= \frac{q^{l(H-1)+1} - (q - l(2c_H + 4) + 1)\Big(\prod_{j=1}^{H-1}(q - 3 - 2c_j) - 2(q-1)^{H-2}\Big)^l}{q^{l(H-1)+1}}$$

Observe that the numerator of the expression has degree $l(H-1)$ in q, whereas the denominator has degree $l(H-1)+1$ in q. The leading coefficient of the numerator is

$$l(2c_H + 4) - 1 + l\big(\sum_{j=1}^{H-1}(3 + 2c_j) + 2\big) = l(\sum_{j=1}^{H} 2c_j + 3H + 3) - 1$$

$$\leq l(2n + H + 3) \leq 3l(n+1).$$

Therefore, if $\omega'(\lambda)$ is a negligible sequence in the security parameter, no PPT adversary can distinguish between μ and ν.

If $H = 1$ instead, observe that $|\Xi_2| \geq \Lambda = q - l(2c_H + 4) + 1$. Therefore,

$$d(\mu,\nu) = \frac{q - |\Xi_2|}{q} \leq \frac{q - q + l(2c_H + 4) - 1}{q} \leq \frac{l(2n+4) - 1}{q}$$

Again, if $\omega'(\lambda)$ is a negligible sequence in the security parameter, no PPT adversary can distinguish between μ and ν.

Now, let F be the function that given a sequence $(U_{k,f(i,j)})_{k\in[n],j\in[l]}$ of points in E, outputs $\sum_{k\in[n]} U_{k,f(i,1)}$. By well known results,

$$d(\mu_F, \nu_F) \leq d(\mu,\nu).$$

Observe that μ_F is the distribution of the output of the i-th bucket of the protocol conditioned on the view of the adversary. On the other hand, ν_F is the uniform distribution over $\langle G\rangle$.

Let ν' be the uniform distribution over $\langle G\rangle \setminus \{\infty\}$. Observe that for every $i \in [N]$, this is the distribution of the i-th output of the functionality. The statistical distance between ν' and ν_F is $1/q$ which is negligible. Since we have

$$d(\mu_F, \nu') \leq d(\mu_F, \nu_F) + d(\nu_F, \nu')$$

and the right-hand side is the sum of two negligible functions, we have proven that the statistical distance between μ_F and ν' is negligible and therefore, for every $i \in [N]$, no PPT adversary is able to distinguish the i-th output of the functionality from the i-th output of the protocol. The theorem follows from lemma 2 and from the fact that the N outputs are i.i.d. in both the protocol and the functionality. □

B.2 Security of Π_{DH}

Proof. Consider the simulator in Figure 18. We show that no adversary is able to distinguish between Π_{DH} and the composition of $\mathcal{F}_{\mathrm{DH}}$ and $\mathcal{S}_{\mathrm{DH}}$.

$\mathcal{S}_{\mathrm{DH}}$

MPC
The simulator checks that the identities specified in the queries correspond to a stored value. If this is the case, it simply forwards the communications between the adversary and the functionality.
Key Generation
The simulator sends $(\mathrm{KeyGen}, \mathrm{id})$ to the functionality and runs the protocol with the adversary. When the simulation obtains a non-∞ public key, $\mathcal{S}_{\mathrm{DH}}$ forwards the point received from the functionality to the adversary. The simulator sends the reply of the adversary to $\mathcal{F}_{\mathrm{DH}}$.
Diffie-Hellman
The simulator sends $(\mathrm{DH}, \mathrm{id}_1, Q, \mathrm{id}_2)$ to the functionality. When W is opened, the $\mathcal{S}_{\mathrm{DH}}$ sends a random element in $\langle G \rangle$ to the adversary and waits for an answer. If the reply is not OK, the simulator sends (Abort) to the functionality. $\mathcal{S}_{\mathrm{DH}}$ never sends ZD to the adversary.

Fig. 18. The Diffie-Hellman Simulator

First of all observe that all the queries available also in $\mathcal{F}_{\mathrm{daPoint}}$ are perfectly simulated. Therefore, we focus our analysis on the key generation and key exchange procedures.

The key generation procedure is perfectly simulated. As a matter of fact, the simulation restarts the procedure with the same probability as the original protocol. Observe that no information is leaked and the adversary has no influence on the protocol apart from the ability of making it abort when the opening is performed. Furthermore, the simulation aborts if and only if the functionality does. The output distribution is exactly the same. Indeed, the protocol outputs $([[s]]_q, sG)$ where $sG \neq \infty$. Such event occurs if and only if $s \in \{1, 2, \ldots, q-1\}$. Now, it is straightforward to see that this is the same distribution as the output of the functionality.

In the key exchange procedure, both protocol and functionality check that the client's public key Q belongs to the proper set i.e. $\langle G \rangle \setminus \{\infty\}$. This is done by checking that $qQ = \infty$. In such case, the order of Q divides q. Since q is a prime, the only possibilities are that the order is either q or 1. The first case is excluded by the fact that we checked that $Q \neq \infty$. Now, at the beginning, we explicitly required that $q^2 \nmid |E|$. That means that there exists only one subgroup of E of order q which must coincide with $\langle G \rangle$. This is sufficient to conclude that $qQ = \infty$ implies $Q \in \langle G \rangle$. We have just proven that the operation in step 2 is well defined.

The distributions of the element W communicated to the adversary in the protocol and the simulation are slightly different. The first one is indeed

$$\mu(W) = \begin{cases} \frac{1}{q-1} & \text{if } sQ - W \neq \infty \\ 0 & \text{otherwise} \end{cases}$$

The second one is instead the uniform distribution ν over $\langle G \rangle$.

The statistical distance between μ and ν is negligible. Indeed

$$d(\mu, \nu) = \frac{1}{2} \cdot \left((q-1) \cdot \left(\frac{1}{q-1} - \frac{1}{q} \right) + \frac{1}{q} \right) = \frac{1}{q}$$

Therefore, no PPT adversary is able to distinguish between μ and ν.

The only remaining possibility for the adversary to distinguish between protocol and simulation is the distribution of the zero denominator event. Indeed, that never happens in the simulation. We obtain a zero denominator if and only if $W = Z - R$ and R have the same x coordinate (i.e. x=u). That happens if and only if $Z - R = R$ or $Z - R = -R$.

The second case occurs if and only if $Z = sQ = \infty$. Since we know that Q has order exactly q and $s \in \{1, 2, \ldots, q-1\}$, this event cannot arise.

The relation $Z - R = R$ holds instead if and only if $Z = sQ = 2R$. We know that sQ is an element of $\langle G \rangle \setminus \{\infty\}$. Since $\langle G \rangle$ has no element of order 2, the point $(s/2)Q \in \langle G \rangle \setminus \{\infty\}$ is well defined. Remember that R is sampled uniformly in the same domain by $\mathcal{F}_{\text{daPoint}}$, therefore the probability that the collision occurs is $1/(q-1)$.

When Π_{DH} succeeds, the distribution of its output is exactly the same as in the simulation. This is sufficient to prove the security of Π_{DH}. □

C Security Proofs of Π_{weakerDH}

In this appendix, we present the security proofs of the protocols described in Section 4.2.

C.1 Security of Π_{ECSum}

Proof. We start by analysing the probability of an abortion when all the parties are honest. This can occur if and only if a zero denominator is found. Consider the execution of the protocol Π_{ECSum} and remember that R_i is sampled uniformly in $\langle G \rangle \setminus \{\infty\}$ for each $i \in [n]$.

We define $T_0 := \infty$. Then, for $i = 1, 2, \ldots, n$, we define $T_i := T_{i-1} + R_i$. A zero denominator occurs if and only if $R_i = \pm T_{i-1}$ for some $i \in [n]$. For $i = 1$, this condition cannot be satisfied. For $i > 1$ instead, the event arises with probability at most $2/(q-1)$. In conclusion, the protocol aborts with probability smaller than $2(n-1)/(q-1)$. If q^{-1} is negligible in the security parameter, the probability of an abortion is negligible.

$\mathcal{S}_{\text{ECSum}}$

MPC

The simulator checks that the identities specified in the queries correspond to a stored value. If this is the case, it simply forwards the communications between the adversary and the functionality.

ECSum

The simulator runs the protocol with the adversary. Let $\{R_i \mid i \in \mathcal{C}\}$ be the points input by the corrupted parties. If an abortion occurs, the simulator sends (Abort) to the functionality. Otherwise, it sends $(\text{ECSum}, \text{id}_1, \text{id}_2)$ to $\mathcal{F}_{\text{ECSum}}$ as well as $\sum_{i \in \mathcal{C}} R_i$.

Fig. 19. The EC-Sum Simulator

Consider now the simulator $\mathcal{S}_{\text{ECSum}}$ described in Figure 19.

Observe that the procedures of $\mathcal{F}_{\text{MPC}}$ are perfectly simulated, whereas EC-Sum aborts with the same distribution as the protocol. What we need to check is that the distribution of the output of the functionality is indistinguishable from the distribution of the output of Π_{ECSum}.

First of all, observe that if the adversary inserts a point that does not belong to the curve, the protocol aborts in step 2. In this case, the indistinguishability is trivial, we therefore assume that this event does not happen. For the same reason, we can also suppose that no zero denominator occurs.

Recall lemma 1 (see Appendix B.1). There exist at least $\Lambda := q - 2c_H - 3$ different values $R := (x, y)$ that the output can assume without causing any zero denominator. Moreover, if $H > 1$, there are at least

$$\rho := \prod_{i=1}^{H-1} (q - 3 - 2c_i) - 2(q-1)^{H-2}$$

different choices for $(R_i)_{i \in \mathcal{H}}$ that satisfy the view of the adversary and such that $\sum_{i=1}^{n} R_i = R$. If instead $H = 1$, there exists exactly one choice for $(R_i)_{i \in \mathcal{H}}$ in order to satisfy the view of the adversary and have $\sum_{i=1}^{n} R_i = R$.

Let μ be the distribution of the output of the protocol conditioned on the view of the adversary. Let ν be the uniform distribution over the set

$$\Xi_1 := \Big\{ (Q_i)_{i \in [n]} \mid Q_i = R_i \ \ \forall i \in \mathcal{C}, \quad Q_i \in \langle G \rangle \ \ \forall i \in \mathcal{H} \Big\}.$$

We show that μ and ν are indistinguishable distributions.

Observe that μ is the uniform distribution over the set Ξ_2 of all possible sequences $(R_i)_{i \in \mathcal{H}}$ that satisfy the view of the adversary. Since $\Xi_2 \subseteq \Xi_1$, the statistical distance between μ and ν is

$$d(\mu, \nu) = \frac{|\Xi_1| - |\Xi_2|}{|\Xi_1|} = \frac{q^H - |\Xi_2|}{q^H}$$

When $H > 1$, we have

$$d(\mu,\nu) \leq \frac{q^H - \Lambda \cdot \rho}{q^H} = \frac{q^H - (q - 2c_H - 3) \cdot \left(\prod_{i=1}^{H-1}(q - 3 - 2c_i) - 2(q-1)^{H-2}\right)}{q^H}$$

Observe that the numerator of the last expression has degree $H-1$ in q, whereas the denominator has degree H in q. Moreover, the leading coefficient of the numerator is

$$2c_H + 3 + \sum_{i=1}^{H-1}(3 + 2c_i) + 2 = \sum_{i=1}^{H} 2c_i + 3H + 2 \leq 3n + 2$$

Therefore, if q^{-1} is negligible in the security parameter λ, no PPT adversary can distinguish between μ and ν when $H > 1$.

If instead $H = 1$, $|\Xi_2| \geq \Lambda = q - 2c_H - 3$, therefore,

$$d(\mu,\nu) = \frac{q - |\Xi_2|}{q} \leq \frac{q - q + 2c_H + 3}{q} = \frac{2c_H + 3}{q} \leq \frac{2n+3}{q}$$

Again, if q^{-1} is negligible in the security parameter λ, no PPT adversary can distinguish between μ and ν when $H = 1$.

Now, let F be the function that given a sequence $(U_i)_{i\in[n]}$ of points in E, outputs $(U_{h_1}, U_{h_2}, \ldots, U_{h_H})$. By well known results,

$$d(\mu_F, \nu_F) \leq d(\mu,\nu).$$

Observe that μ_F is the distribution of the output of Π_{ECSum} conditioned on the view of the adversary. On the other hand, ν_F is the uniform distribution over $\langle G\rangle^H$.

If $\sum_{i\in\mathcal{C}} R_i \notin \langle G\rangle$, ν_F coincides with the distribution of the output of the functionality (it is impossible for the functionality to obtain $R = \infty$). Therefore, in such case, the security of Π_{ECSum} is proven. If instead $\sum_{i\in\mathcal{C}} R_i \in \langle G\rangle$, the output of the functionality is uniformly distributed over

$$\left\{(R_{h_1}, R_{h_2}, \ldots, R_{h_H}) \in \langle G\rangle^H \mid \sum_{i\in[n]} R_i \neq \infty\right\}.$$

Let ν' be its distribution. The statistical distance between ν' and ν_F is

$$d(\nu', \nu_F) = \frac{q^H - q^{H-1}(q-1)}{q^H} = \frac{1}{q}$$

which is negligible. Since we have

$$d(\mu_F, \nu') \leq d(\mu_F, \nu_F) + d(\nu_F, \nu')$$

and the right-hand side is the sum of two negligible functions, we have proven that the statistical distance between μ_F and ν' is negligible. We conclude that no PPT adversary is able to distinguish the output of the functionality from the output of the protocol. Since the stored point is $\sum_{i\in[n]} R_i$ in both the protocol and the simulation, this terminates the proof. □

C.2 Security of Π_{weakerDH}

Proof. Consider the simulator $\mathcal{S}_{\text{weakerDH}}$ described in Figure 20. Observe that the MPC procedures are perfectly simulated.

$\mathcal{S}_{\text{weakerDH}}$

Let $\iota \in \mathcal{H}$.

MPC

The simulator checks that the identities specified in the queries correspond to a stored value. If this is the case, it simply forwards the communications between the adversary and the functionality.

Key Generation

1. The simulator sends $(\text{KeyGen}, \text{id})$ to $\mathcal{F}_{\text{weakerDH}}$ on behalf of every corrupted party P_j.
2. Let S be the point communicated by the functionality. The simulator samples a random integer $s_i \xleftarrow{\$} \mathbb{F}_q$ and sets $S_i \leftarrow s_i G$ for each $i \in \mathcal{H} \setminus \{\iota\}$.
3. When the adversary sends $\{(s_j, S_j)\}_{j \in \mathcal{C}}$, the simulator checks whether $S_j = s_j G$ for every $j \in \mathcal{C}$. In such case, it sets $S_\iota \leftarrow S - \sum_{i \in [n] \setminus \{\iota\}} S_i$. Otherwise, it samples $s_\iota \xleftarrow{\$} \mathbb{F}_q$ and sets $S_\iota \leftarrow s_\iota G$.
4. The simulator sends $\{S_i\}_{i \in \mathcal{H}}$ to the adversary. If the answer is OK, the simulator forwards the message to the functionality if and only if $S_j = s_j G$ for every $j \in \mathcal{C}$. Otherwise, it sends (Abort).

Diffie-Hellman The simulator checks whether $qQ = \infty$ and $Q \neq \infty$. If this is not the case, it stops. Otherwise, it runs the protocol with the adversary simulating $\mathcal{F}_{\text{ECSum}}$ with an internal copy of the resource. Let U be the point communicated by the adversary in this occasion. The simulator models the points broadcast by the honest parties with random points $(W_i)_{i \in \mathcal{H}}$ in $\langle G \rangle$.
When every corrupted party P_j has broadcast a point W'_j, the simulator checks that $W'_j \in E$ for each $j \in \mathcal{C}$. It this is not the case, it sends (Abort) to the functionality. Otherwise, it sends to the functionality

$$Q_\epsilon \leftarrow \sum_{j \in \mathcal{C}} (W'_j - s_j Q) + U \quad \text{and} \quad Q_\tau \leftarrow 2 \sum_{i \in \mathcal{H}} W_i + 2 \sum_{j \in \mathcal{C}} W'_j.$$

If $\mathcal{F}_{\text{weakerDH}}$ aborts, the simulator sends ZD to the adversary.

Fig. 20. The simulator $\mathcal{S}_{\text{weakerDH}}$

We start by analysing the key generation procedure. The distribution of the points $\{S_i\}_{i \in \mathcal{H}}$ received by the adversary is the same in both protocol and simulation. Indeed, in both cases, the points $\{S_i\}_{i \in \mathcal{H}}$ are independent and uniformly distributed in $\langle G \rangle$. The fact is trivially true for the protocol. In the case of the simulation, we have to consider two possibilities. If there exists $j \in \mathcal{C}$ such that $S_j \neq s_j G$, the fact is again trivial. In the other case, the points $\{S_i\}_{i \in \mathcal{H} \setminus \{\iota\}}$ are independent and uniformly distributed in $\langle G \rangle$. Moreover, $\sum_{i \in [n] \setminus \{\iota\}} S_i$ is a point

of $\langle G\rangle$. Since S is independent of everything else and it is uniformly distributed in $\langle G\rangle$, the point $S_\iota = S - \sum_{i\in[n]\setminus\{\iota\}} S_i$ is independent of $\{S_i\}_{i\in\mathcal{H}\setminus\{\iota\}}$ and it is uniformly distributed in $\langle G\rangle$.

The distribution of the output of the key generation is the same in both protocol and simulation. As a matter of fact, if the adversary does not provide the correct private keys to $\mathcal{F}_{\text{KEY}}$, both the situations abort. When no abortion occurs, the output of the honest parties is the sum of the points $S_1, S_2, \ldots, S_n$ in both the protocol and the simulation.

Consider now the execution of Diffie-Hellman. The distribution of the points broadcast by the honest parties may be slightly different in the simulation. Specifically, if we denote with U the point supplied by the adversary to $\mathcal{F}_{\text{ECSum}}$, in the protocol the points $(W_i)_{i\in\mathcal{H}}$ are uniformly distribute over the set

$$\Big\{(W_{h_1}, W_{h_2}, \ldots, W_{h_H}) \in \langle G\rangle^H \mid U + \sum_{i\in\mathcal{H}}(s_i Q - W_i) \neq \infty\Big\},$$

whereas in the simulation they are uniform in $\langle G\rangle^H$. Anyway, the statistical distance between the two distributions is at most $1/q$, which is negligible. Therefore, no PPT adversary can distinguish the protocol from the simulation by simply looking at them.

In both protocol and simulation, let W'_j be the point broadcast by the corrupted party P_j and let s be the private key (i.e. the discrete logarithm of S). Furthermore, we set $Q_\epsilon := \sum_{j\in\mathcal{C}}(W'_j - s_j Q) + U$. Let $R := (x, y)$ be the point stored by $\mathcal{F}_{\text{ECSum}}$. The Diffie-Hellman procedure in Π_{weakerDH} fails due to e zero denominator if and only if

$$\begin{aligned} R = \pm W &= \pm\Big(\sum_{i\in\mathcal{C}} W'_i + \sum_{i\in\mathcal{H}} W_i\Big) = \\ &= \pm\Big(\sum_{i\in\mathcal{C}}(W'_i - s_i Q) + U + \sum_{i\in[n]} s_i Q - U - \sum_{i\in\mathcal{H}} R_i\Big) = \pm\Big(Q_\epsilon + sQ - R\Big). \end{aligned}$$

In other words, $W = -R$ if and only if $sQ + Q_\epsilon = \infty$. Moreover, $R = W$ if and only if $2R = sQ + Q_\epsilon$ and therefore, if and only if $2W = sQ + Q_\epsilon$. Observe that the Diffie-Hellman procedure of the simulation fails in the same way when any of these conditions holds.

The only thing that we have to check in order to finish is that the results stored with identity id_2 in the protocol and in the simulation coincide. This is trivially true. Indeed, in the protocol the output is the x coordinate of

$$\begin{aligned} W + R &= \sum_{i\in\mathcal{H}} W_i + \sum_{j\in\mathcal{C}} W'_j + R = \sum_{i\in\mathcal{H}}(W_i + R_i) + \sum_{j\in\mathcal{C}} W'_j + U = \\ &= \sum_{i\in\mathcal{H}} s_i Q + \sum_{j\in\mathcal{C}} W'_j + U = sQ + \sum_{j\in\mathcal{C}}(W'_j - s_j Q) + U = \\ &= sQ + Q_\epsilon. \end{aligned}$$

□

D Security Proof of Π_{Sign}

We now prove the security of the protocol Π_{Sign} described in Section 4.4.

Proof. Consider the simulator $\mathcal{S}_{\text{Sign}}$ described in Figure 21 and let the symbol $\overset{q}{\equiv}$ denote the congruence relation modulo q.

$\mathcal{S}_{\text{Sign}}$

Let G be the base point of the curve and let q be its order. Let P_ι be an honest party.

Initialization:

1. When the simulator receives (Key) from a corrupted party P_j, the simulator sends (Init) to $\mathcal{F}_{\text{Sign}}$ on behalf of P_j.
2. Let A be the point communicated by the functionality. The simulator samples a random integer $a_i \xleftarrow{\$} \langle q \rangle$ and sets $A_i \leftarrow a_i G$ for each $i \in \mathcal{H} \setminus \{\iota\}$.
3. When the adversary sends $(a_j, A_j)_{j \in \mathcal{C}}$, the simulator checks whether $A_j = a_j G$ for every $j \in \mathcal{C}$. In such case, it sets $A_\iota \leftarrow A - \sum_{i \in [n] \setminus \{\iota\}} A_i$. Otherwise, it samples $a_\iota \xleftarrow{\$} \langle q_\lambda \rangle$ and sets $A_\iota \leftarrow a_\iota G$.
4. The simulator sends $(A_i)_{i \in \mathcal{H}}$ to the adversary. If the answer is OK, the simulator forwards the message to the functionality if and only if $A_j = a_j G$ for every $j \in \mathcal{C}$. Otherwise, it sends (Abort).

Sign: The simulator performs the following steps.

1. It stores the triple (s, R, m) received from the functionality.
2. It samples a random $r_i \xleftarrow{\$} \langle q \rangle$ for each $i \in \mathcal{H} \setminus \{\iota\}$.
3. It computes $R_i \leftarrow r_i G$ for each $i \in \mathcal{H} \setminus \{\iota\}$.
4. When the adversary sends $(r_j, R_j)_{j \in \mathcal{C}}$, the simulator checks whether $R_j = r_j G$ for every $j \in \mathcal{C}$. In such case, it sets $R_\iota \leftarrow R - \sum_{i \in [n] \setminus \{\iota\}} R_i$. Otherwise, it samples $r_\iota \xleftarrow{\$} \langle q \rangle$ and sets $R_\iota \leftarrow r_\iota G$.
5. The simulator sends $(R_i)_{i \in \mathcal{H}}$ to the adversary.
6. If the answer is not OK or there exists $j \in \mathcal{C}$ such that $R_j \neq r_j G$, the simulator sends (Abort) to the functionality.
7. The simulator computes $s_i \leftarrow (r_i + H(R, A, m) \cdot a_i) \bmod q$ for each $i \in [n] \setminus \{\iota\}$.
8. The simulator computes $s_\iota \leftarrow \left(s - \sum_{i \in [n] \setminus \{\iota\}} s_i\right) \bmod q$.
9. The simulator sends s_i to every corrupted party for each $i \in \mathcal{H}$.
10. When every party P_j sent a value s'_j, the simulator checks whether $\sum_{j \in \mathcal{C}} s'_j = \sum_{j \in \mathcal{C}} s_j \bmod q$. In such case, it sends OK to the functionality, otherwise, it sends (Abort).

Fig. 21. The simulator $\mathcal{S}_{\text{Sign}}$

The initialization is perfectly simulated. The fact has been already proven in Appendix 4.2. Indeed, observe that the initialization of Π_{Sign} coincides with the initialization in Π_{weakerDH}.

Consider now the simulation of the signature generation. Observe that the distribution of the points $(R_i)_{i\in\mathcal{H}}$ received by the adversary is the same in both the protocol and the simulation. As a matter of fact, in both cases, the points $(R_i)_{i\in\mathcal{H}}$ are independent and uniformly distributed in $\langle G\rangle$. The fact is trivially true for the protocol, in the simulation instead, there are two possibilities. If there exists $j \in \mathcal{C}$ such that $R_j \neq r_j G$, the fact is again obvious. Otherwise, the points $(R_i)_{i\in\mathcal{H}\setminus\{\iota\}}$ are independent and uniformly distributed in $\langle G\rangle$. Moreover, $\sum_{i\in[n]\setminus\{\iota\}} R_i$ is a point of $\langle G\rangle$. Since R is independent of everything else and it is uniformly distributed in $\langle G\rangle$, the point $R_\iota = R - \sum_{i\in[n]\setminus\{\iota\}} R_i$ is independent of $(R_i)_{i\in\mathcal{H}\setminus\{\iota\}}$ and it is uniformly distributed in $\langle G\rangle$.

Observe that if the adversary does not provide the correct private keys or it does not give the final OK to $\mathcal{F}_{\text{KEY}}$, both the situations abort. Furthermore, notice that if no abortion occurs in the simulation, the sum of the points $R_1, R_2, \ldots, R_n$ is exactly R.

The values $(s_i)_{i\in\mathcal{H}\setminus\{\iota\}}$ are perfectly simulated. This is something easy to verify. Actually, even s_ι is perfectly simulated. As a matter of fact, denoting the discrete logarithm of R with r, we have

$$\begin{aligned}
s_\iota &\stackrel{q}{\equiv} s - \sum_{\substack{i=1\\ i\neq\iota}}^{n} s_i \stackrel{q}{\equiv} r + H(R,A,m)\cdot a - \sum_{\substack{i=1\\ i\neq\iota}}^{n}(r_i + H(R,A,m)\cdot a_i) \stackrel{q}{\equiv}\\
&\stackrel{q}{\equiv} \sum_{i\in[n]}(r_i + H(R,A,m)\cdot a_i) - \sum_{\substack{i=1\\ i\neq\iota}}^{n}(r_i + H(R,A,m)\cdot a_i) \stackrel{q}{\equiv}\\
&\stackrel{q}{\equiv} r_\iota + H(R,A,m)\cdot a_\iota.
\end{aligned}$$

The last thing that we have to check in order to finish the proof is that the output distribution is the same. For both the protocol and the simulation, let s'_j be the element broadcast by the corrupted party P_j. Furthermore, let $s' := \left(\sum_{i\in\mathcal{H}} s_i + \sum_{j\in\mathcal{C}} s'_j\right) \bmod q$. The following equalities hold.

$$\begin{aligned}
s'G &= \Big(\sum_{i\in\mathcal{H}} s_i + \sum_{j\in\mathcal{C}} s'_j\Big)G = \sum_{i\in\mathcal{H}} s_i G + \sum_{j\in\mathcal{C}} s'_j G =\\
&= \sum_{i\in\mathcal{H}}(r_i + H(R,A,m)a_i)G + \sum_{j\in\mathcal{C}} s'_j G =\\
&= \sum_{i\in\mathcal{H}}(R_i + H(R,A,m)A_i) + \sum_{j\in\mathcal{C}} s'_j G =\\
&= R + H(R,A,m)A - \sum_{j\in\mathcal{C}}(R_j + H(R,A,m)A_j) + \sum_{j\in\mathcal{C}} s'_j G.
\end{aligned}$$

In other words, $s'G = R + H(R,A,m)A$ if and only if

$$\sum_{j\in\mathcal{C}} s'_j G = \sum_{j\in\mathcal{C}}(R_j + H(R,A,m)A_j).$$

If we perform the discrete logarithm on both sides of the equation, we obtain that the condition holds if and only if

$$\sum_{j\in\mathcal{C}} s'_j \overset{q}{\equiv} \sum_{j\in\mathcal{C}} (r_j + H(R,A,m)a_j) \overset{q}{\equiv} \sum_{i\in\mathcal{C}} s_j.$$

This is sufficient to understand that the simulation aborts if and only if the protocol aborts. Moreover, if the protocol does not abort, the outputs coincide. As a matter of fact, when R is fixed, there exists a unique $s \in \langle q \rangle$ that satisfies the equation $sG = R + H(R,A,m)A$. In other words, the distributions of s and s' coincide. □

E Security of the MPC-friendly AEAD

We introduce some notation. Given two strings a and b, we denote their concatenation by $a \parallel b$. Notice that the latter is different from (a,b). Indeed, it is possible to find $(a',b') \neq (a,b)$ such that $a \parallel b = a' \parallel b'$.

In order to be formal, we start by giving a definition of OTP-based MAC-then-Encrypt AEAD.

Definition 4 (OTP-based MAC-then-Encrypt). *Let*

$$F : \mathcal{K} \times \{0,1\}^{l_1} \longrightarrow \mathbb{F}_p^{l_2}$$

be a PRF and let MAC be a function

$$MAC\colon \mathcal{K}' \times \Big(\bigcup_{i=0}^{s_1} \mathbb{F}_p^i \times \bigcup_{i=0}^{s_2} \mathbb{F}_p^i\Big) \longrightarrow \mathbb{F}_p^{s_3}.$$

Assume that $s_2+s_3 \leq l_2$. We define the MAC-then-Encrypt AEAD MtE(F, MAC) as follows.

Encryption: *On input*

- *a key $(k,k') \in \mathcal{K} \times \mathcal{K}'$,*
- *a nonce $r \in \{0,1\}^{l_1}$,*
- *additional data $a \in \mathbb{F}_p^{l'}$ with $l' \leq s_1$,*
- *plaintext $x \in \mathbb{F}_p^{l}$ with $l \leq s_2$,*

compute $t \leftarrow MAC(k', (a,x))$ and $z \leftarrow F(k,r)$. Then, output

$$c \leftarrow (x \parallel t) + Trunc\big(z, len(x) + len(t)\big).$$

Decryption: *On input*

- *a key $(k,k') \in \mathcal{K} \times \mathcal{K}'$,*
- *a nonce $r \in \{0,1\}^{l_1}$,*
- *additional data $a \in \mathbb{F}_p^{l'}$ with $l' \leq s_1$,*
- *ciphertext $c \in \mathbb{F}_p^{l}$ with $s_3 \leq l \leq s_2 + s_3$,*

compute $z \leftarrow F(k, r)$. *Let* $(x \parallel t) \leftarrow c - Trunc(z, l)$ *where* $t \in \mathbb{F}_p^{s_3}$. *Afterwards, compute* $t' \leftarrow MAC(k', (a, x))$. *If* $t = t'$, *output* x, *in the remaining cases, output* $\perp$.

In [21], Krawczyk proved that an OTP-based *MAC-then-Encrypt* scheme is secure as long as the MAC is one-query resistant.

$\mathcal{G}^{\text{FORG}}_{\mathcal{M},\mathcal{A}}(\lambda)$

Setup: The challenger randomly samples $k \xleftarrow{\$} \mathcal{K}_\lambda$. Afterwards, the PPT adversary $\mathcal{A}$ is activated with a sequence of λ bits set to 1.
Query: Only once, the adversary queries a value $x \in D_\lambda$ which the challenger answers with $t \leftarrow \text{MAC}_\lambda(k, x)$.
Guess: The adversary sends a couple $(y, z) \in D_\lambda \times S_\lambda$. The challenger answers with a bit b that is set if and only if $z = \text{MAC}_\lambda(k, y)$ and $y \neq x$. The bit b is the final output of the game.

Fig. 22. The one-query forgery resistance game

Definition 5 (One-query forgery resistant MAC). *Let*

$$\mathcal{M} := \{MAC_\lambda : \mathcal{K}_\lambda \times D_\lambda \longrightarrow S_\lambda\}_{\lambda \in \mathbb{N}}$$

be a family of functions, where $\mathcal{K}_\lambda$, D_λ *and* S_λ *are generic sets. Assume that* $\mathcal{A}$ *is a PPT adversary.*

For every $\lambda \in \mathbb{N}$, *consider the game* $\mathcal{G}^{FORG}_{\mathcal{M},\mathcal{A}}(\lambda)$ *described in Figure 22. We define the one-query forgery resistance advantage of* $\mathcal{A}$ *against* $\mathcal{M}$ *as*

$$Adv^{FORG}_{\mathcal{M},\mathcal{A}}(\lambda) := \mathbb{P}\big(\mathcal{G}^{FORG}_{\mathcal{M},\mathcal{A}}(\lambda) = 1\big).$$

We say that $\mathcal{M}$ *is one-query forgery resistant if, for every PPT adversary* $\mathcal{A}$, *the advantage* $Adv^{FORG}_{\mathcal{M},\mathcal{A}}(\lambda)$ *is a negligible function in* λ.

The result of Krawczyk (see [21]) can be formalised as follows

Theorem 7. *Let* $F : \mathcal{K} \times \{0,1\}^{l_1} \longrightarrow \mathbb{F}_p^{l_2}$ *be a PRF and let MAC be a function*

$$MAC \colon \mathcal{K}' \times \Big(\bigcup_{i=0}^{s_1} \mathbb{F}_p^i \times \bigcup_{i=0}^{s_2} \mathbb{F}_p^i\Big) \longrightarrow \mathbb{F}_p^{s_3}.$$

Assume that $s_2 + s_3 \leq l_2$. *If MAC is one-query forgery resistant, then* $MtE(F, MAC)$ *is a secure AEAD.*

It is possible to prove that PolyMAC is a one-query forgery resistant MAC function. This is sufficient to conclude the security of our AEAD. The result is formalised by the following theorem.

Theorem 8. *For every PPT adversary $\mathcal{A}$, we have*

$$Adv^{FORG}_{PolyMAC,\mathcal{A}}(\lambda) \leq \frac{L_1 + L_2 + 1}{p}$$

Proof. Consider the game $\mathcal{G}^{\text{FORG}}_{\text{PolyMAC},\mathcal{A}}(\lambda)$ and let (y, z) be the MAC key selected by the challenger.

Suppose that the adversary queried the pair

$$u := ((u_1, u_2, \ldots, u_r), (u_{r+1}, u_{r+2}, \ldots, u_{r+r'})) \in \mathbb{F}_p^r \times \mathbb{F}_p^{r'}$$

with $r \leq L_1$ and $r' \leq L_2$. Moreover, assume that $t := \text{PolyMAC}\big((y, z), u\big)$ was the reply of the challenger.

Let (w, v) be the forgery attempt issued by the adversary. In particular, $w = ((w_1, w_2, \ldots, w_s), (w_{s+1}, w_{s+2}, \ldots, w_{s+s'}))$ with $s \leq L_1$ and $s' \leq L_2$, whereas $v \in \mathbb{F}_p$. We assume that w is different from u.

We define the polynomials

$$w(X) = \text{Ecd}(s') \cdot X^{s+s'+1} + w_1 \cdot X^{s+s'} + w_2 \cdot X^{s+s'-1} + \cdots + w_{s+s'} \cdot X,$$

$$u(X) = \text{Ecd}(r') \cdot X^{r+r'+1} + u_1 \cdot X^{r+r'} + u_2 \cdot X^{r+r'-1} + \cdots + u_{r+r'} \cdot X.$$

We have that

$$\text{Adv}^{\text{FORG}}_{\text{PolyMAC},\mathcal{A}}(\lambda) = \mathbb{P}\Big(\text{PolyMAC}\big((y, z), w\big) = v \,|\, \text{PolyMAC}\big((y, z), u\big) = t\Big) =$$

$$= \mathbb{P}(w(y) + z = v \,|\, u(y) + z = t) = \mathbb{P}((w - u)(y) + (v - t) = 0) =$$

$$= \mathbb{P}(y \text{ is a root of the polynomial } (w - u)(X) + (v - t)).$$

Notice that if $s + s' \neq r + r'$, the polynomial $(w - u)(X) + (v - t)$ has degree $\max\{s + s' + 1, r + r' + 1\}$, indeed the leading coefficient of $u(X)$ and $w(X)$ is always different from 0.

If $s + s' = r + r'$ but $s' \neq r'$, the degree of $(w - u)(X) + (v - t)$ is $s + s' + 1$. Otherwise, the degree is $s + s' + 1 - e > 0$ where e is the smallest index such that $w_e \neq u_e$. We know that such index exists.

We conclude that the polynomial $(w - u)(X) + (v - t)$ is always different from 0. Furthermore, its degree is less or equal to $L := L_1 + L_2 + 1$, therefore it has at most L distinct roots. That means that

$$\text{Adv}^{\text{FORG}}_{\text{PolyMAC},\mathcal{A}}(\lambda) = \mathbb{P}(y \text{ is a root of the polynomial } (w - u)(X) + (v - t)) \leq \frac{L}{p}$$

□

F Performance

In this section, we present details of the benchmarks we used to estimate the performance of Oblivious TLS. Due to its complexity, we have not implemented a fully working, standards-compliant system, but instead separately benchmarked the key subroutines in the protocol.

We modified the code of SCALE-MAMBA [1], which implements the SPDZ [11] protocol for dishonest-majority MPC of arithmetic circuits, as well as BMR garbled circuits for binary operations [17]. One limitation is that, at the time of our implementation, SCALE-MAMBA did not support arithmetic modulo arbitrary primes, which is needed for our Diffie-Hellman protocol using *Curve25519* over $\mathbb{F}_{2^{255}-19}$. We therefore performed the same computations over a different prime field of similar size, to emulate this stage of the protocol. We focused our experiments on the performance of the "online phase", assuming that the necessary preprocessing material has been generated in advance.

Handshake protocol. We analysed the latency of the main components of the handshake, using a single machine with an Intel Xeon E5-1650 CPU (6 physical cores, HyperThreading, 3.20 GHz) and 32 GB RAM. The tests were run with a 2-party MPC engine on localhost, and the results are presented in Table 1. Note that we only benchmarked Π_{weakerDH} rather than Π_{DH}, and also did not implement signature generation. We expect Π_{DH} to give similar results to Π_{weakerDH}, although, the preprocessing costs would be higher due to its use of daPoints. If authentication with signatures is needed, we expect the overhead to be minimal, since the signing protocol uses no MPC operations and has very little computation on top of plaintext signing. We also point out that we tested the key derivation without using pre-shared keys and without computing the resumption secret and the exporter secret. If these features were used, the procedure would become more complex and the computational cost might significantly increase. In conclusion, since the remaining operations are performed locally, we can expect a handshake to be performed in around 1 or 2 seconds.

Protocol	Instantiation	Latency
Π_{weakerDH}	Curve25519	$\sim 10^{-2}$ s
Key derivation	SHA-256	~ 1 s
Key update	SHA-256	~ 0.5 s
AEAD MAC powers	$\mathbb{F}_{2^{128}}$	$\sim 10^{-1}$ s
HMAC	SHA-256	$\sim 10^{-1}$ s

Table 1. Latency of the main operations in the Oblivious TLS handshake protocol

Algorithm	Instantiation	Throughput
AES-GCM (Garbled Circuits)	AES-128	~ 1 KB/s
MPC-friendly AEAD	128-bit prime field	~ 300 KB/s
	AES-based PRF	

Table 2. Throughput of AEAD schemes in Oblivious TLS

Record layer. For the record layer, we compared the performance of the two AEAD schemes we considered, namely AES-GCM and the custom, multi-party variant. These tests were run on two identical machines running on a 1Gbit Ethernet LAN with a measured throughput of 935 Mbits/s. The average latency of the network was 0.43 ms. The two machines had an Intel Core i7-8700 CPU (6 physical cores, HyperThreading, 3.20 GHz) and 32 GB RAM. The results are described in Table 2. For AES-GCM, we tested the protocol based on garbled circuits, since the lookup table-based approach of [18] is not supported in SCALE-MAMBA. The results show that our MPC-friendly AEAD is around 300x faster than using garbled circuits, which reflects the fact that the former approach avoids all evaluation of AES inside MPC. Based on the results from [18], we expect that switching to their alternative AES evaluation method, we could achieve throughputs up to 3 MB/s for AES-GCM, even exceeding the MPC-friendly AEAD. This is because [18] evaluates AES with a very lightweight online phase, which contrasts with the intensive computations and bandwidth required by garbled circuits. However, when the preprocessing phases are taken into consideration, we expect our new AEAD to behave significantly better than the AES-GCM version based on [18], which comes with a very expensive preprocessing in order to achieve the fast online phase.

We point out that in a real application, there may also be additional costs in the record layer, when accounting for the removal of padding, or converting between secret-sharing schemes, depending on the data format used by the application running on top of TLS.