The privacy of the TLS 1.3 protocol

Ghada Arfaoui[1], Xavier Bultel[2,3], Pierre-Alain Fouque[2,3], Adina Nedelcu[1,2,3], and Cristina Onete[4]

[1] Orange Labs, France
[2] IRISA, France
[3] Rennes Univ, France
[4] XLIM/CNRS 7252, France

Abstract. TLS (Transport Layer Security) is a widely deployed protocol that plays a vital role in securing Internet traffic. Given the numerous known attacks for TLS 1.2, it was imperative to change and even redesign the protocol in order to address them. In August 2018, a new version of the protocol, TLS 1.3, was standardized by the IETF (Internet Engineering Task Force). TLS 1.3 not only benefits from stronger security guarantees, but aims to protect the identities of the server and client by encrypting messages as soon as possible during the authentication. In this paper, we model the privacy guarantees of TLS 1.3 when parties execute a full handshake or use a session resumption, covering all the handshake modes of TLS. We build our privacy models on top of the one defined by Hermans *et al.* for RFIDs (Radio Frequency Identification Devices) that mostly targets authentication protocols. The enhanced models share similarities to the Bellare-Rogaway AKE (Authenticated Key Exchange) security model and consider adversaries that can compromise both types of participants in the protocol. In particular, modeling session resumption is non-trivial, given that session resumption tickets are essentially a state transmitted from one session to another and such link reveals information on the parties. On the positive side, we prove that TLS 1.3 protects the privacy of its users at least against passive adversaries, contrary to TLS 1.2, and against more powerful ones.

Keywords: privacy, TLS 1.3, AKE protocols

1 Introduction

The TLS protocol is one of the most commonly used secure-channel establishment protocols today. It ensures the security of, for example, messages exchanged over the Internet when incorporated in `https` [13,27], secure emailing, and even Voice-over-IP (VoIP) communications [29]. As other authenticated key-exchange (AKE) protocols, TLS consists of two steps: a *handshake* and subsequently, *secure message-exchange*. During the handshake, a client and a server exchange information over an insecure channel, allowing for (unilateral or bilateral) authentication and for the computation a tuple of symmetric keys. Subsequently, during the secure message-exchange step (also called the record layer for TLS), these keys are used with authenticated encryption. This process guarantees the most basic security properties of a protocol for secure-channel establishment, namely the confidentiality and authentication of the messages.

In this paper, however, we shift the focus away from the *security* of secure-channel establishment, and instead consider the *privacy* it provides.

In TLS, parties can choose to execute a full handshake each time they communicate, or they can resume a past session using session resumption. The TLS 1.3 full handshake that is most likely to be used in practice consists of a unilaterally (server-only) authenticated Diffie-Hellman-based key-exchange, which guarantees *perfect forward secrecy*, *i.e.*, compromising a party's long term keys does not affect past sessions. Once a client and a server have successfully completed a full handshake, it is possible to resume that handshake later, by using a pre-shared key and optionally an additional Diffie-Hellman element. This improves performance by avoiding authentication.

Session resumption in TLS 1.3 strongly depends on so-called *session tickets*. Upon the completion of a (full or resumed) session, the server sends the client a ticket as part of the secure session-traffic. In order to resume that session the client sends the unencrypted ticket back in a following session, and both parties use the associated pre-shared key (PSK) to compute new session keys. Unfortunately, session resumption, in its pre-shared key only mode, yields no forward secrecy.

In this paper we show that, in addition to forward secrecy, PSK-based session resumption also loses the degree of privacy guaranteed by full-mode handshakes. The same problem holds for PSK-DHE resumption.

1.1 The TLS 1.3 protocol

The history of the TLS protocol is littered with attacks against both the handshake and the record layer subprotocols [12,30,25,3,2,26,28,9,1,10,8,4,20,11,5]. The plethora of flaws discovered in the TLS 1.2 version have led the IETF to propose TLS 1.3 as the future de-facto AKE protocol to safeguard Internet connections.[5]

In many ways, the design of TLS 1.3 revolutionizes real-world authenticated key-exchange, by employing modern cryptographic mechanisms. All messages in the the protocol are encrypted using AEAD (Authenticated Encryption with Auxiliary Data). Additionally, HKDF (Hash Key Derivation Function), replaces HMAC in the key schedule. The key schedule itself is much more complex than in previous versions, respecting the paradigm of separating keys used at different layers and for different purposes. Insecure or obsolete algorithms from previous versions of TLS are no longer supported by TLS 1.3. The protocol is designed with modularity in mind, which should make it easier to implement or formally analyse. We give a more detailed description of the design elements that affect the privacy of TLS 1.3 in Appendix A.

[5] For this paper, we have relied on the August 2018 version of RFC 8446, available at `https://datatracker.ietf.org/doc/rfc8446/`.

1.2 Privacy notions for AKE

Beyond confidentiality, the notion of *user privacy* has been increasingly required for practical cryptographic countermeasures. Such requirements were exacerbated by Edward Snowden's revelations of mass surveillance attacks and large data centers storing massive amounts of user metadata [32], which co-motivated the emergence of GDPR [16] and e-Privacy [17] regulations. In addition, designers of cryptographic primitives and protocols have taken to an "encrypt as early as possible" paradigm, which formed the backbone of TLS 1.3.

At the very minimum, a privacy-protecting protocol can hide the identity of the participants (both client and servers). At the level of the protocol, this can be done by never using identifiers in plaintext; for lower layers, more complex mechanisms such as Tor might have to be put into place. But, while learning a party's identity is a complete privacy breach, partial information leakage – such as realizing whether the protocol has been run before, or linking a single, anonymous user to two distinct sessions – can also be exploited. This is the reason why modern privacy-preserving protocols aim to guarantee various flavours of *unlinkability*, rather than the weaker property of *identity-hiding*. This is also the approach we take in this paper, modelling strong adversaries and minimal restrictions on winning conditions.

Even less ambitious goals, such as identity-hiding, can be difficult to achieve in practice, for instance in the context of authentication. Krawczyk [23] noted that it is impossible to design a protocol that will protect both peers' identities from *active attacks*, since the first peer must disclose its identity to its partner before authentication can take place. As TLS 1.3 is expected to mainly run – like its predecessors – with only unilateral authentication, we may hope that it protects the identity of the server from passive and active adversaries. This would be a vast improvement with respect to previous versions of TLS, in which the server's identity is usually sent in clear together with its certificate.

In addition to the revolutionary design of its full handshake, TLS 1.3 enables session resumption in two modes: PSK and PSK-DHE. This latter mode adds freshness in session key computation. Both modes make use of the so-called session tickets. However, there seems to be no consensus about its implementation, especially that TLS 1.3 specification gives only generic guidelines about its construction. In this paper, we analyse the privacy of the different modes of TLS handshake and discuss the privacy impact of the way session tickets are constructed.

1.3 Our contributions

Our three main contributions are as follows: we formalized a game-based Left-or-Right indistinguishability definition for the properties attained by the protocol; we described a number of inevitable attacks in AKE protocols (providing for them in our model); and finally we proved the privacy properties guaranteed by the full handshake and the session-resumption mechanisms. We discuss below in more detail each of these contributions.

Our privacy model. We define the privacy of TLS as a type of unlinkability of protocol sessions. The adversary is an active Man-in-the-Middle, who can interact with protocol participants arbitrarily, akin to Bellare-Rogaway AKE adversaries [7]. However, as opposed to [7] models, in which the adversary knows whom he is interacting with, in our definitions we use the notion of *virtual identifier* taken from the RFID privacy framework of Hermans *et al.* [22]. Our adversaries will repeatedly be able to query a *drawing oracle*, which takes as input two (possibly distinct) parties of the same type (clients or servers) and outputs either the left or the right party, depending on a secret bit b. The *goal* of our adversary will be to guess b with a probability significantly larger than $\frac{1}{2}$.

A key aspect of our model is that we view resumed sessions as being linked to the previous session in which the PSK (and the ticket) is computed; we account for this by allowing clients and servers to have a *state*. The concept of tying sessions together in this way lies at the core of our model, and is one of the strongest ways in which the adversary can try to link sessions. Unfortunately, this inter-connection between the session also implies some restrictions in terms of the Left-or-Right, Corruption, and Revealing queries that the adversary is allowed to make.

A non-trivial design choice in our model concerns Left-or-Right queries. Ideally, we would like the adversary to be able to make multiple drawing queries, under reasonable restrictions (such as: one cannot make draw the same party twice without first freeing it). Unfortunately this seems impossible: during the proof the reduction to the AEAD security of the channel over which the ticket is sent would require guessing a large, combinatorial number of instances. Consequently, we have a choice of whether to define selective privacy (the adversary declares in advance which parties it will later draw), or allowing a single Draw-Party query, which is –however– adaptive. We choose the latter approach.

Trivial Attacks. Ideally, we would have liked our games to have a "clean" winning condition: the adversary would win if he managed to output a correct guess for the bit b. Unfortunately, this is not possible. Even for the full TLS 1.3 handshake an adversary can win by impersonating a client – since we consider server-only authentication. We call such an attack *trivial*, in the sense that it automatically allows the adversary to win, regardless of the design of the protocol.

Resumption brings out many more attacks against user privacy, which we detail in Section 3. The easiest way in which an attacker can break our Left-or-Right privacy notion is to choose (by using the Draw oracle) two parties such that, for some given partner, one of the two parties holds a resumption ticket with that partner, while the other does not. In that case, the adversary's strategy would be to force resumption and distinguish between the drawn parties based on whether resumption worked (the handshake runs to completion) or not (there is an abort). Thus, we must restrict the adversary's winning conditions to capture the indistinguishability between two parties that have similar resumption profiles. We note that these threats are generalizable to a wider category of protocols supporting resumption. Indeed, the attacks do not exploit features specific to TLS 1.3, but rather, weaknesses of session resumption in general.

Proving the privacy of TLS 1.3. In order to prove the privacy of TLS 1.3, we employ techniques often used in provable security for analysing AKE or other types of protocols. We reduce the problem of breaking the privacy of TLS users to either breaking down the atomic cryptographic primitives (like the signature or the authenticated encryption) or solving computational problem presumed hard (such as computational Diffie-Hellman). As long as these assumptions hold true, TLS guarantees the privacy of its users up to a certain number of intrinsic trivial attacks that we exclude from the model.

1.4 Applicability and impact

Our results are, to some degree, tailored to TLS 1.3, and to some degree more generic, covering a wider class of protocols. We point out some of the limitations below.

Protocol limitations. Analysing the privacy of complex protocols, such as TLS 1.3, is a daunting task. As a result, we only focus on some of its features, including the full handshake, session resumption in PSK and PSK-DHE mode, but not 0-RTT. The precise protocol we analyse in this paper is described in Section 2.2.

Although we strove to include as many of the protocol's privacy-preserving features as possible, some seem difficult to model, including parameter negotiation and error messages. The former can be serious privacy risks, since they can be used to profile the server's or client's behaviour, which in turn can help link sessions of the same party. Another feature we omit is the Server Name Indication (SNI) extension, which allows a single server to run TLS handshakes on behalf of multiple domains, using multiple public keys. Defining privacy in this context is tricky, since we would have to model the fact that certain servers are allowed to run handshakes for one domain, while others cannot.

Finally, we only considered one possible implementation of session tickets, which is also implemented by, for instance, WolfSSL[6]: namely, the server will encrypt the resumption master secret and a nonce within the session ticket, using a long-term symmetric key. However, we also discuss other implementations in our concluding remarks. Session tickets are non-reusable in our model.

Model limitations. Our model is also restricted to unilaterally, server-only-authenticated protocols in which the client sends the first message. However, we can trivially transform a protocol for which the server is the initiator into one in which it is the responder (by just adding a dummy message from the client, prompting the communication) without impacting the security analyses; It is only slightly more complicated to capture a mutually-authenticated handshake: we must add a winning condition that prohibits the adversary from impersonating a client.

[6] `https://www.wolfssl.com/`

We also assume that servers have a way of *a priori* distinguishing whether the handshake will be run in full, PSK, or PSK-DHE modes, thus excluding some tampering attempts by the adversary. In the TLS 1.3 protocol, no such *a priori* knowledge is needed since the server adjusts to the format of the client's first message. Finally, the mechanics of the Left-or-Right party-drawing oracle amount to a number of artificially destroyed tickets, which have no correspondence in real life.

Our attacks. In this paper we present a number of ways in which a generic adversary can link protocol sessions, both when session resumption is used (most attacks), and when it is not. These attacks exploit weaknesses which appear in TLS 1.3, but that are generalizable to larger classes of protocols. Informally speaking, it is the resumption mechanism in general introduces weakness, not the TLS 1.3 resumption in particular.

For session resumption, most of our attacks are, to some extent, parallelizable, but have a limited real-world impact. The attacks generally exploit the fact that resumed sessions imply the existence of a previous, linked session in which the ticket was forwarded. This allows an attacker to always distinguish between a party that should be able to resume a session, and a party that cannot. In the real world, even if an adversary can distinguish between such two parties, he would require auxiliary information to fully identify the parties as the sets of resuming and non-resuming parties, depending on the use case, can be prohibitively large. Nonetheless, as resumption is often used, for instance, when accessing multiple resources on the same webpage, this would still give an attacker important information about a user's access patterns. Consequently, such attacks are included in our analysis.

Privacy in isolation. In this paper, we prove the privacy of the TLS 1.3 protocol in isolation, without considering its composition with lower-layer protocols, nor other encapsulating primitives. We argue that this is still meaningful, for two main reasons. First, note that as a general rule, privacy tends to be either *preserved* or *lost*: it is much harder to "create" it. In other words, if TLS 1.3 did *not* preserve privacy, then its use – even encapsulated in privacy-preserving lower-layer protocols – would still lead to privacy breaches. In this paper, our goal was to show precisely what kind of privacy TLS 1.3 preserves. In some ways, this indicates how much privacy we can hope for, when TLS 1.3 is used as a protocol in computer networks.[7] We do note that one way to extend our result would be to verify the possibility of composing it with known results on privacy-preserving routing protocols, such as Tor. This deserves to be the subject of a separate paper.

[7] Note, however, that the reverse is unfortunately not true: even if TLS 1.3 does preserve privacy, this does not by default guarantee the privacy of its encapsulation in lower-layer protocols. This is an important limitation of our result.

1.5 Related work

To the best of our knowledge, this work proposes the first analysis of the privacy achieved by TLS 1.3. Our model has some similarities with existing work on privacy. We combine authenticated key-exchange models akin to Bellare-Rogaway [7] with game-based privacy in authentication, as defined by Hermans *et al.* [22]. This approach was previously taken by Fouque *et al.* [19] but in the context of mobile network communications (without public-key primitives and session resumption). Although we rely on both the Bellare-Rogaway definition of secure AKE and on the Hermans *et al.* notion of privacy-preserving RFID authentication, our model is a non-trivial extension of these two frameworks. The definitions in this paper are much closer to characteristics such as the unilateral server-only authentication of TLS 1.3, its complex key schedule, and the ticket-based mechanism of session resumption.

Since TLS is a network protocol, our work also touches upon the field of anonymous communication in computer networks. An early formalization of anonymous channels is provided by Hevia and Micciancio [21], who describe an adversary that is given a matrix of messages in a given protocol, and his goal is to distinguish between them. The adversary is passive and cannot corrupt parties. Their framework defines several types of privacy properties (such as sender and/or receiver unlinkability, anonymity) and shows existing reductions (either trivial, or by using techniques such as cryptography or padding). The model of Hevia and Micciancio focuses more on the number of messages, their sender, receiver, and size, and can be seen as a more global view of a network protocol when subjected to traffic analysis. Our goal here is different: we aim to describe the properties that are achieved by the TLS 1.3 protocol somewhat in isolation (thus characterizing the design of TLS 1.3, rather than the way it is used). We argue that this allows us to consider the effects of stronger adversaries, which are allowed to corrupt parties, reveal keys, and play active Man-in-the-Middle roles.

Hermans *et al.* [22] take an approach closer to ours and formalize privacy in authentication protocols, particularly in the context of RFID authentication. They build on previous work dating back to the privacy model of Vaudenay [31]. The adversary is an active Man-in-the-Middle which can interact with parties, adaptively corrupt them, and learn the result of protocol executions. A central concept of this framework is that of *virtual tag*, which is a handle meant to hide the identity of a party (namely an RFID tag) from the adversary while the latter interacts with that tag. We adopt this concept here, and follow the general design of the Left-or-Right (LoR) indistinguishability game used by Hermans *et al.*. As a result, our definition captures the unlinkability of TLS 1.3 sessions.

Our results are orthogonal to those of research on the *security* achieved by the TLS protocol, such as [14,**?**], although our model does rely on a simplification of the multistage security defined by Fischlin and Günther [18].

We assume that implementers follow best practices and ticket anti-replay measure are in place. Therefore, the Selfie attack [15] would not occur in our model.

This paper focuses on the privacy achieved by TLS 1.3 in isolation. We do not focus on its privacy when composed with lower-level protocols, a limitation which we discuss in Section 1.4. In that sense, our results are orthogonal to work which covers the privacy of anonymous networking protocols like Tor [6].

Outline of the paper. The paper is structured as follows. In Section 2, we model the TLS 1.3 protocol and introduce cryptographic assumptions. In Section 3 we describe several trivial attacks. We develop a model for privacy of the full handshake in Section 4 and extend the model by adding resumption in Section 5. We conclude in Section 6.

2 Preliminaries

The results in this paper are proved for an abstract form of the full and resumed TLS 1.3 handshake. We stress that our model of the TLS 1.3 handshakes is incomplete as detailed in Section 1.4. In this section we first describe the way we model the TLS 1.3 full handshake and two resumption modes. We then describe in more detail the key scheduling and a protocol idealization in Section 2.1. We introduce cryptographic assumptions in Section 2.2. They are formally defined in Appendix B. We illustrate, in Figure 1, an idealization of the TLS 1.3 handshake in full mode. We designate this protocol as Π_{TLS}. Then, we also illustrate, in Figure 2, session resumption in both pre-shared key (PSK) and in pre-shared key with ephemeral Diffie Hellman (PSK-DHE) modes. We denote this by $\Pi_{\mathsf{TLS+res}}$. To clarify, in $\Pi_{\mathsf{TLS+res}}$ the parties can execute either full handshakes or session resumptions, while in Π_{TLS} they can only execute full handshakes. The notations we use when describing the protocols are summarized in Table 1. The key schedule is presented in Figure 3.

2.1 TLS 1.3 handshake and session resumption

In TLS 1.3, the client chooses the handshake mode by the way it constructs its first message protocol. From that point onward, the execution of the protocol follows the path illustrated in the figures. If a message is ill-formed, incomplete, invalid or out of order, the session is terminated with a relevant error message. Anticipating a bit, we model this by having the Send oracle returning $\perp$ (we do not model the multiple error messages defined in the RFC).

Overview. We can distinguish three main phases in the TLS protocol: the key exchange, the server authentication and the session authentication[8]. During the first phase, the client and the server exchange desired session parameters and

[8] We use terminology that is slightly different than the one used in the RFC, for readability purposes.

es	early secret	RO	random oracle
hs	handshake secret	k	server's ticket encrypt. key
ms	master secret	CHello...Fin_C	TLS messages
C.hs/ S.hs	client/server handshake secret	g	generator of a group $\mathbb{G}$
C.htk/ S.htk	client/server handshake traffic key	$\|\|$	concatenation
C.fk/ S.fk	client/server finished key	$\|$	or operator
C.ts/ S.ts	client/server traffic secret	$\{\}_{key}$	AE encryption with key
C.tk/ S.tk	client/server traffic key	$\{\}^{-1}_{key}$	AE decryption with key
rms	resumption master secret	""	empty string
psk	preshared key	H_τ	hash of the partial transcript:
STicket	session ticket		C.hs, S.hs: H(CHello....KE_S)
N_T,in.N_T	ticket nonces		C.ts, S.ts: H(CHello....Fin_S)
in. (out.)	prefixes (see Fig 2, Fig 11)		rms: H(CHello....Fin_C)
bk	binder key		Fin_S: H(CHello....CVf)
bnd	preshared key binder		Fin_C: H(CHello....Fin_S)
ℓ_0...ℓ_9	labels/strings		resumption specific:
PRF	pseudo-random fct.		bnd: H(CHello....STicket)
MAC	message auth. code		Fin_S: H(CHello....KE_S)

Table 1. List of notations used in Figure 1 and Figure 2.

key share components. This allows them to compute an intermediate secret and temporary encryption keys, which they use to encrypt the rest of the handshake. In the second phase, the server sends his certificate and a signature; the client uses them to authenticate the server. At the end of the protocol, the client and server exchange a Message Authentication Code (MAC) over the transcript. If they correctly verify these messages, they compute a new secret, from which they derive the traffic encryption keys.

At this stage, the server can send some secret data (i.e., a ticket) to the user enabling him to later on "jumpstart" a new session. This is so-called session resumption. In order to initiate a resumption, a client will send a ticket to a server, followed by an associated MAC. If the ticket is valid, this allows parties to simplify the negotiation/key exchange phase and completely eliminate the second phase, that of server authentication. Resumption comes in two flavours: pre-shared key (PSK) and pre-shared key with ephemeral Diffie Hellman (PSK-DHE) modes, depending on whether or not a Diffie-Hellman key exchange is executed. Incorporating the Diffie-Hellman into resumption offers stronger security guarantees, at the price of extra computation and protocol messages.

Key schedule. The key schedule of TLS 1.3 appears in Figure 3. We distinguish three main secrets: the early secret es, the handshake secret hs and the master secret ms. In a full mode handshake, es is simply a publicly-computable string. However, it is used to "inject" the preshared key psk into the key schedule when resuming a session. The handshake secret hs is used to derive client and server handshake secrets (C.hs and S.hs), from which the parties compute two handshake traffic keys (C.htk and S.htk) as well as message authentication keys for the Finished messages (C.fk and S.fk). We introduce the prefix "C." to designate the client secrets and keys used to send data to the server. Similarly,

Client C		**Server** S $(\mathsf{sk}_S,\ \mathsf{k}_S)$
	Key Exchange	
$\mathsf{es} := \mathsf{RO}(0;0)$	$\xrightarrow{1.\mathsf{CHello},\mathsf{KE}_C=g^x}$	$\mathsf{es} := \mathsf{RO}(0;0)$
	$\xleftarrow{2.\mathsf{SHello},\mathsf{KE}_S=g^y}$	
$\mathsf{hs} := \mathsf{RO}(g^{xy};\mathsf{es}\|\ell_0)$		$\mathsf{hs} := \mathsf{RO}(g^{xy};\mathsf{es}\|\ell_0)$
$\mathsf{C.hs} := \mathtt{PRF}(\mathsf{hs};\ell_1,H_\tau)$		$\mathsf{C.hs} := \mathtt{PRF}(\mathsf{hs};\ell_1,H_\tau)$
$\mathsf{S.hs} := \mathtt{PRF}(\mathsf{hs};\ell_2,H_\tau)$		$\mathsf{S.hs} := \mathtt{PRF}(\mathsf{hs};\ell_2,H_\tau)$
$\mathsf{C.htk} := \mathtt{PRF}(\mathsf{C.hs};\ell_3,\text{""})$		$\mathsf{C.htk} := \mathtt{PRF}(\mathsf{C.hs};\ell_3,\text{""})$
$\mathsf{S.htk} := \mathtt{PRF}(\mathsf{S.hs};\ell_3,\text{""})$		$\mathsf{S.htk} := \mathtt{PRF}(\mathsf{S.hs};\ell_3,\text{""})$
	Server Authentication	
Verify $\mathsf{Cert}_S,\mathsf{CVf}$	$\xleftarrow{3.\{\mathsf{Cert}_S,\mathsf{CVf}\}_{\mathsf{S.htk}}}$	$\mathsf{CVf} := \mathtt{Sign}.\mathsf{Sign}(\mathsf{sk}_S,H_\tau)$
$\mathsf{S.fk} := \mathtt{PRF}(\mathsf{S.hs};\ell_4,\text{""})$		$\mathsf{S.fk} := \mathtt{PRF}(\mathsf{S.hs};\ell_4,\text{""})$
$\mathsf{C.fk} := \mathtt{PRF}(\mathsf{C.hs};\ell_4,\text{""})$		$\mathsf{C.fk} := \mathtt{PRF}(\mathsf{C.hs};\ell_4,\text{""})$
	Session Authentication	
Verify Fin_S	$\xleftarrow{4.\{\mathsf{Fin}_S\}_{\mathsf{S.htk}}}$	$\mathsf{Fin}_S := \mathtt{MAC}(\mathsf{S.fk};H_\tau)$
$\mathsf{Fin}_C := \mathtt{MAC}(\mathsf{C.fk};H_\tau)$	$\xrightarrow{5.\{\mathsf{Fin}_C\}_{\mathsf{C.htk}}}$	Verify Fin_C
$\mathsf{ms} := \mathsf{RO}(0;\mathsf{hs}\|\ell_0)$		$\mathsf{ms} := \mathsf{RO}(0;\mathsf{hs}\|\ell_0)$
$\mathsf{C.ts} := \mathtt{PRF}(\mathsf{ms};\ell_5,H_\tau)$		$\mathsf{C.ts} := \mathtt{PRF}(\mathsf{ms};\ell_5,H_\tau)$
$\mathsf{S.ts} := \mathtt{PRF}(\mathsf{ms};\ell_6,H_\tau)$		$\mathsf{S.ts} := \mathtt{PRF}(\mathsf{ms};\ell_6,H_\tau)$
$\mathsf{C.tk} := \mathtt{PRF}(\mathsf{C.ts};\ell_3,\text{""})$		$\mathsf{C.tk} := \mathtt{PRF}(\mathsf{C.ts};\ell_3,\text{""})$
$\mathsf{S.tk} := \mathtt{PRF}(\mathsf{S.ts};\ell_3,\text{""})$		$\mathsf{S.tk} := \mathtt{PRF}(\mathsf{S.ts};\ell_3,\text{""})$
$\boxed{\mathsf{rms} := \mathtt{PRF}(\mathsf{ms};\ell_7,H_\tau)}$		$\boxed{\mathsf{rms} := \mathtt{PRF}(\mathsf{ms};\ell_7,H_\tau)}$
		$\boxed{\mathsf{STicket} := \{\mathsf{rms},\mathsf{N}_T\}_{\mathsf{k}}}$
	Secure record layer with tk(second model only)	
	$\xleftarrow{6.\boxed{\{\mathsf{STicket},\mathsf{N}_T\}_{\mathsf{S.tk}}}}$	

Fig. 1. Our modelling of the TLS 1.3 handshake - full handshake mode. We do not explicitly include the length of parameters. The sections in boxes concern only the extended protocol, $\Pi_{\mathsf{TLS+res}}$.

we use the prefix “S.” to designate the secrets and keys of the server used to send data to the client. The master secret ms is first used to compute the client and server secrets ($\mathsf{C.ts}$ and $\mathsf{S.ts}$), and ultimately the traffic (encryption) keys: $\mathsf{C.tk}$ and $\mathsf{S.tk}$. Optionally, the master secret can be used to derive a resumption secret rms and, from it, the preshared key psk. The preshared key and associated ticket are needed in order to resume a session.

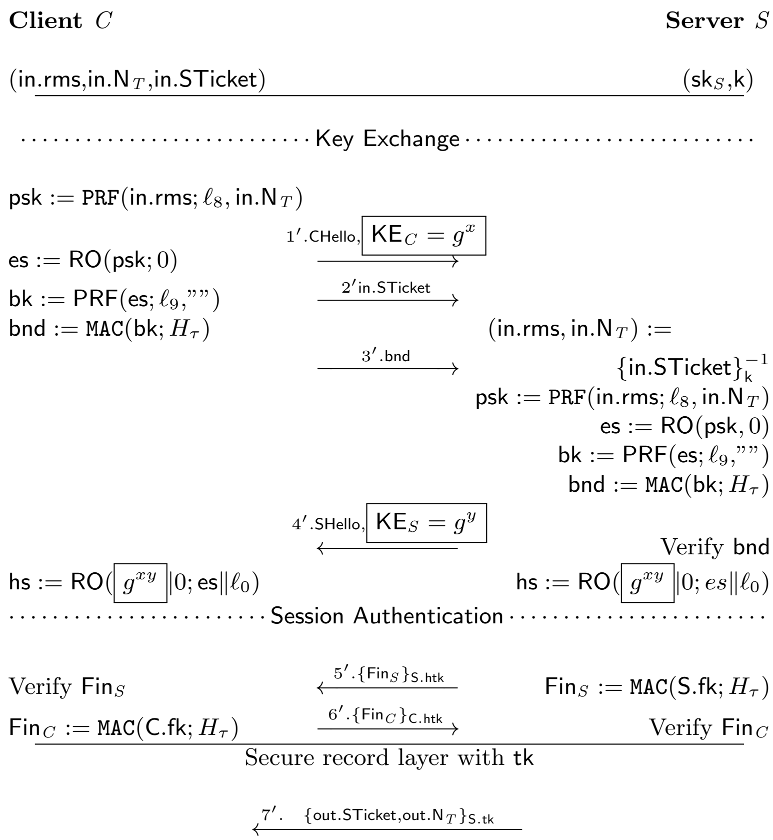

Fig. 2. Our modelling of the TLS 1.3 handshake- session resumption, both preshared key-only and preshared key with Diffie Hellman key exchange. In the boxes we have the protocol elements specific to the pre-shared key with Diffie-Hellman key exchange mode. We have prefixed the N_T, rms, STicket used at the beginning of the session with in, and those created at the end of a session by out. This emphasises they are different variables.

Modelling the key derivation. In order to perform the key derivation, TLS 1.3 uses the Hash Key-Derivation Function (HKDF) [24], which has two main operations: Extract and Expand. The Extract operation is applied to an *input key material*, with some (optional) *salt*. Its role is to transform a sufficiently random input into a secret that has high entropy and is compact. The Expand operation takes a *secret*, a *label* and an *input*. The *secret* is usually the output of a preceding Extract operation. The *label* is a publicly-known string and serves to have different outputs for the same inputs. For example, two keys might be computed in almost the same way, but using different labels will produce distinct and independent keys for different contexts. The *input* is usually a session hash

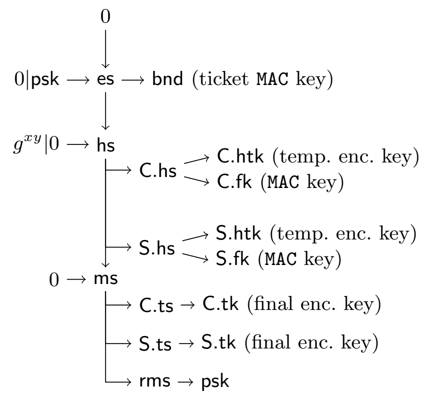

Fig. 3. The key schedule of TLS 1.3, with bidirectional keys. We use psk if it is a resumption and 0 otherwise. We use g^{xy} if the handshake mode requires a Diffie-Hellman key exchange, and 0 otherwise. Encrypting the handshake and record layer messages with distinct sets of keys is one of the improvements of TLS 1.3.

(the hash applied to the partial transcript), but it can sometimes be an empty string "".

Although the original protocol uses the HKDF function [24] to extract key material, then expand it into keys, we choose to model the extraction steps as a random oracle (RO), and the expansion steps as runs of a pseudorandom function (PRF). Using the random-oracle model is a strong idealization; however, we deem it acceptable for two reasons: (1) previous analyses of TLS 1.3 do show that the keys obtained through HKDF are indistinguishable from random [14], under stronger assumptions like PRF-ODH; (2) our focus here is privacy, and not the security of keys – in idealizing the key derivation, our proofs are cleaner and easier to follow.

Protocol messages. A TLS 1.3 protocol run consists of the following messages. The numbers in brackets indicate the position of these messages in Fig 1 and Fig 2.

CHello(1,1'): The Client Hello message consists of the protocol version(s), a nonce N_C generated by the client, as well as lists of supported cryptographic primitives and client extensions.

KE_C(1,1'): Both in the full handshake, and in the case of session resumption with PSK+DHE, the client provides a client key-share message KE_C consisting of (the description of) a series of groups and an element g^x in each group, respectively[9].

[9] This key-share is also present in the PSK resumption mode; however, if only PSK resumption is used, the DH element provided by the client is not used.

$\underline{\mathsf{SHello}(2,4')}$: The Server Hello message includes the server's nonce N_S, as well as the server's selection of the version, extensions, and supported cryptographic primitives (from amongst the alternatives stated in CHello).

$\underline{\mathsf{KE}_S(2,4')}$: The server's key-share element consists of a single element g^y chosen for one single group, chosen from amongst those sent in KE_C.

$\underline{\mathsf{Cert}_S(3)}$: The server's certificate Cert_S is modelled here as just a public key, which we assume is only attributed to one legitimate entity holding the corresponding private key.

$\underline{\mathsf{CVf}(3)}$: The server issues the Certificate Verify message to authenticate to the client as the owner of the key in Cert_S. The CVf is a signature on the hash of the handshake messages, up to, and including Cert_S.

$\underline{\mathsf{Fin}_S(4,5')}$: The server Finished message Fin_S is a MAC keyed with the Server Finished Key $\mathsf{S.fk}$, on input the current session hash, up to, and including CVf.

$\underline{\mathsf{Fin}_C(5,6')}$: The client Finished message Fin_C is a MAC keyed with the key $\mathsf{C.fk}$, on input the session hash up to, and including Fin_S.

$\underline{\mathsf{STicket}(6,2',7')}$: At the end of a handshake, the server may send a client a session resumption ticket $\mathsf{STicket}$ followed by a nonce N_T. The $\mathsf{STicket}$ encapsulates rms and a nonce N_T in an encrypted and authenticated manner. These values will be used to compute psk. The N_T needs to be forwarded to the client as well. When resuming a session, a client sends the $\mathsf{STicket}$ after CHello and KE_C.

$\underline{\mathsf{bnd}(3')}$: In the cases of PSK and PSK+DHE, the client sends a pre-shared key binder bnd, which is a MAC keyed with the key bk, on input CHello, KE_C (if present) and $\mathsf{STicket}$. The key bk is derived from the early secret es, which takes as input the pre-shared key psk.

2.2 Cryptographic assumptions

Let $\mathcal{A}$ designate an algorithm, commonly referred to as an adversary. We denote $a \xleftarrow{\$} A$ if the element a is uniformly randomly sampled from the set A.

The Computational Diffie-Hellman (CDH*).* Let $\mathbb{G}$ be a multiplicative cyclic group of order $|\mathbb{G}|$ and g a generator. Let us define $\mathsf{Exp}_{\mathbb{G}}^{\mathsf{CDH}}(\mathcal{A}) : x, y \xleftarrow{\$} |\mathbb{G}|$, $g, g^x, g^y \to \mathcal{A}$, $g^* \leftarrow \mathcal{A}$. We define the advantage of $\mathcal{A}$ as $\mathsf{Adv}_{\mathsf{CDH}}^{\mathbb{G}}(\mathcal{A}) = \mathbb{P}[g^* = g^{xy}]$. Any adversary $\mathcal{A}$ against CDH in the group $\mathbb{G}$ running in time t and making at most q queries has an advantage of at most ϵ_{CDH}: $\epsilon_{\mathsf{CDH}} \geq \mathsf{Adv}_{\mathsf{CDH}}^{\mathbb{G}}(\mathcal{A})$.

Pseudorandom functions (prf*).* Let $\mathcal{K}$ be a keyspace. Let $\mathtt{PRF} : \mathcal{K} \times \{0,1\}^m \to \{0,1\}^n$ be some function family. Let $\mathcal{R}^{m\to n}$ be the set of all functions from $\{0,1\}^m$ to $\{0,1\}^n$ and f a function from $\mathcal{R}^{m\to n}$. We define the following oracle $\mathsf{PRF}^{\mathsf{b}}(z)$: If $\mathsf{b} = 1$, output $\mathtt{PRF}(k; z)$; otherwise, output $f(z)$. Let us define $\mathsf{Exp}_{\mathtt{PRF}}^{\mathsf{prf}}(\mathcal{A})$: $\mathsf{b} \xleftarrow{\$} \{0,1\}$, $k \xleftarrow{\$} \mathcal{K}$, $f \xleftarrow{\$} \mathcal{R}^{m\to n}$, $\mathsf{d} \leftarrow \mathcal{A}^{\mathsf{PRF}^{\mathsf{b}}(\cdot)}$. We define the advantage of $\mathcal{A}$ as $\mathsf{Adv}_{\mathsf{prf}}^{\mathtt{PRF}}(\mathcal{A}) = \left|\mathbb{P}[\mathsf{b}{=}\mathsf{d}] - \frac{1}{2}\right|$. Any adversary $\mathcal{A}$ against the prf property of $\mathtt{PRF}$ running in time t and making at most q queries has an advantage of at most ϵ_{CDH}: $\epsilon_{\mathsf{CDH}} \geq \mathsf{Adv}_{\mathsf{CDH}}^{\mathbb{G}}(\mathcal{A})$.

Existential unforgeability ($\mathsf{EUF\text{-}CMA}$*).* A digital signature scheme $\mathtt{Sign}$ is a tuple of three algorithms: $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vfy})$. $\mathsf{Gen}()$ outputs a pair of a signing key sk and a verification key pk. The algorithm Sign takes as input the key sk and a message msg and outputs a signature σ. The algorithm Vfy takes as input the key pk, a message msg, and a signature σ, and outputs 1 if the signature σ is valid for the message msg and 0 otherwise.

We define an oracle $\mathsf{Sign}(\mathsf{msg})$ that returns $\mathtt{Sign}.\mathsf{Sign}(\mathsf{sk}, \mathsf{msg})$ and stores msg in a list $\mathcal{L}_{\mathsf{sig}}$. Let us define $\mathsf{Exp}^{\mathsf{EUF\text{-}CMA}}_{\mathtt{Sign}}(\mathcal{A})$: $\mathsf{b} \xleftarrow{\$} \{0,1\}$, $\mathcal{L}_{\mathsf{sig}} \leftarrow \emptyset$, $\mathsf{sk}, \mathsf{pk} \leftarrow \mathtt{Sign}.\mathsf{Gen}()$,$(\mathsf{msg}^*, \mathsf{sk}^*) \leftarrow \mathcal{A}^{\mathsf{Sign}(\cdot)}$. We define the advantage of $\mathcal{A}$ as $\mathsf{Adv}^{\mathtt{Sign}}_{\mathsf{EUF\text{-}CMA}}(\mathcal{A}) = \mathbb{P}[\mathtt{Sign}.\mathsf{Vfy}(\mathsf{pk}; \mathsf{sk}^*) = 1 \wedge \mathsf{msg}^* \notin \mathcal{L}_{\mathsf{sig}}]$. Any adversary $\mathcal{A}$ against the $\mathsf{EUF\text{-}CMA}$ property of $\mathtt{Sign}$ running in time t and making at most q queries has an advantage of at most $\epsilon_{\mathsf{EUF\text{-}CMA}}$: $\epsilon_{\mathsf{EUF\text{-}CMA}} \geq \mathsf{Adv}^{\mathtt{Sign}}_{\mathsf{EUF\text{-}CMA}}(\mathcal{A})$.

$\mathtt{AE}$ is *a stateful-length hiding authenticated encryption scheme* (or stLHAE). Such a scheme provides confidentiality of communication, integrity of ciphertexts and additional data, protection against message reordering and replay, as well as hiding the length of the messages to some degree. An adversary $\mathcal{A}$ against stLHAE is given access to encryption and decryption oracles, $\mathsf{AEnc}^{\mathsf{b}}$ and ADec. Informally, an adversary wins the security game if he can distinguish between two possible outputs of AEnc (he chooses two messages and receives a ciphertext encrypting one of them) or if he can desynchronize the ADec oracle by inputting a successful forgery.

We formally define the stLHAE security experiment in Appendix B.

3 Trivial attacks

In this section we detail trivial attacks applicable to TLS and related protocols, using an intuitive pseudo-protocol notation.

Parties (Alice, Bob, or a website W/W') exchange messages. The messages are written on top of arrows going from a sender to a receiver of that message. Rather than formally defining a full protocol and its set of messages, we informally describe message contents or their intended role in between commas. If a party knows a symmetric encryption key k or a session resumption ticket t, this is noted as a superscript. An encrypted text is written within accolades, with the encryption key specified as a subscript.

$\mathcal{A}$ denotes the adversary. If a text is *emphasised*, this is a precondition or an assumption (on the powers of the adversary) needed for the attack. Non-emphasised text is a comment or an explanation. If we list two parties, followed by a question mark, this means the adversary is unsure which of the two is the real sender or receiver of the message.

3.1 Full handshake attacks

Trivial privacy leaks (Figure 4). Messages used to authenticate a user are, by their nature, privacy-sensitive. The party sending the first authentication message in a protocol has no way of knowing, at the time the message is sent, if they are communicating with an honest or malicious party. In previous versions of TLS, the server used to send its certificate in the clear. This trivially leaks the identity of the server to any eavesdropper. Encrypting the message protects it to some degree. Sessions between honest parties no longer leak sensitive information, but active adversaries could mount a man-in-the-middle attack by initiating a protocol session. As discussed in [23], AKE protocols cannot hide the identity of both parties against active adversaries.

a)Alice $\xleftarrow{\text{“I am W''}}$ W

Eavesdroppers can read the message.

b)$\mathcal{A}^k \xleftarrow{\{\text{“I am W”}\}_k}$ W^k

Man-in-the-middle attack against privacy.

Fig. 4. Unencrypted versus encrypted authentication messages: encrypting provides stronger privacy guarantees, but it cannot defend against adversaries impersonating the unauthenticated party.

Impersonating a server leaks information about clients. Specifically, if they want to connect or not to that server. (Figure 5). Assume Alice wants to have a session with a website, either W or W'. However, assume there exists an adversary that can convince Alice that he is, for example, the website W. Alice will accept the session if and only if the adversary impersonated the correct website. Even if the adversary impersonates the wrong server, he still ends up learning something about Alice that he didn't know before mounting the attack: namely, Alice did not wish to initiate a session with that particular server.

$\mathcal{A}$ *is able to impersonate W.*

Alice $\xrightarrow{\text{“Start new session ”}}$ W or W'?

. . . .

Alice $\xleftarrow{\text{“I am W''}}$ $\mathcal{A}$(W)

a)If Alice accepts the session, W is the intended partner.

b)If Alice rejects, W'$\neq$W is the intended partner.

Fig. 5. If an adversary can convince Alice he is W, he can learn whether Alice intended to start a session with W or with another server.

3.2 Resumption attacks

Client with a ticket, distinguishable from a client without a ticket (Figure 6). Alice and Bob wish to connect to a website W. Alice has a resumption ticket, Bob does not. An adversary sees either Alice or Bob establishing a session with W. If it is a session resumption, he can, by process of elimination, conclude that it was Alice who initiated the resumption.

$$
\begin{aligned}
&\text{a)Alice}^t \xrightarrow{\text{“Start new session”}} \text{W} \\
&\text{b)Bob} \xrightarrow{\text{“Start new session”}} \text{W} \\
&\text{c)Alice}^t \xrightarrow{\text{“Resume session using } t\text{”}} \text{W}
\end{aligned}
$$

Fig. 6. Alice can either start a new session or resume an older one, while Bob can only start a new session. This makes them distinguishable.

We can apply the same argument for servers. Assume that Alice has a resumption ticket from a website W, but she doesn't have such a ticket from a website W'. If she resumes, she is clearly in a session with W, and not with W'.

Session Linking (Figure 7). An adversary can learn more about the identity of the participants in a session, if he is able to "link" it to another one he knows more about. This is possible due to session resumption tickets that appear identical in succeeding sessions. However, to mount this attack, the adversary must first retrieve the ticket, e.g., by decrypting the first message encoding the ticket. The attack works as follows. We assume the adversary obtains the transcripts of various protocol sessions, amongst which, a full handshake and its resumption. Let us assume he has a way of "accessing" the ticket the server sent in the first session. Because he sees the same ticket in both sessions, he can conclude that one session is the resumption of the other.

Assume the adversary is uncertain about the identity of the parties in one of the sessions. Due to this additional information (having linked the two sessions), he may now resolve this uncertainty. We illustrate one such example in Figure 7.

$$
\begin{gathered}
\text{Alice}^k \xleftarrow{\{\text{Ticket } t\}_k} \text{W}^k \\
\mathcal{A} \;\; \textit{retrieves Alice's ticket } t. \\
\text{Alice or Bob?}^t \xrightarrow{\text{“Resume session using } t\text{”}} \text{W} \\
\mathcal{A} \;\; \text{concludes Alice, and not Bob resumed the session.}
\end{gathered}
$$

Fig. 7. By seeing the same ticket in two sessions, the adversary concludes that one session is the resumption of the other one. Alice, who received the ticket in the initial session, is the party who resumed the session.

Ticket redirection (Figure 8). Assume Alice wishes to resume a session with a server unknown to the adversary. To find out to whom Alice wishes to connect, the adversary will intercepts her first message in the protocol and forward it to various servers. When he encounters a server W that accepts the ticket, he has identified Alice's intended website.

$$\text{Alice}^t \xrightarrow{\text{“Resume session with W using } t\text{”}} \mathcal{A}$$
$$\mathcal{A} \xrightarrow{\text{“Resume session with W using } t\text{”}} \text{W or W'?}$$

Fig. 8. The adversary reroutes a ticket meant for W to a server he is uncertain about. If the server accepts, it is W. If not, it is some other website.

Cascading pre-shared key compromise (Figure 9). It's not exactly a stand-alone attack, but a "feature" that exacerbates other types of attacks, such as session linking.

TLS's pre-shared key only resumption mode does not ensure perfect forward secrecy. Thus, once the adversary obtains the key material in a session, he can compute the keys and secrets of their session resumption.

Session linking appeared when an adversary identified the same ticket in a session and its resumption. If that session is instead resumed n number of times using pre-shared key only resumption and the first session is compromised, then an adversary could pair any of the n sessions in order to obtain information about one of the parties.

$$\text{Alice}^k \xleftarrow{\{\text{Ticket } t\}_k} \text{W}^k$$

$\mathcal{A}$ *obtains t and its associated preshared key.*
Using the preshared key, $\mathcal{A}$ computes the
key schedule of subsequent resumptions.

....

$$\text{Alice}^{k'} \xleftarrow{\{\text{Ticket } t'\}_{k'}} \text{W}^{k'}$$
$$\text{Alice or Bob?}^{t'} \xrightarrow{\text{“Resume session using } t'\text{”}} \text{W}$$

Fig. 9. Assume the adversary compromises a session. Then he can compromise any subsequent resumptions if they do not include a Diffie Hellman exchange. This allows him to mount the session linking attack even after an n-th resumption.

Ticket encryption key compromise (Figure 10). Assume an adversary that knows the key a website W is using to encrypt its tickets. Next, the adversary tries to decrypt any tickets he sees on the network. If the decryption succeeds, he knows the messages are meant for W.

$\mathcal{A}$ *has W's ticket encryption key.*

$$\text{Alice}^k \xleftarrow{\{\text{Ticket } t\}_k} \text{W or W'?}^k$$

$\mathcal{A}$ decrypts t, concludes Alice is talking to W
if and only if decryption succeeds.

Fig. 10. Assume that $\mathcal{A}$ has the long term ticket encryption key of a server W. He then decrypts any tickets on the network. If decryption succeeds, the ticket is meant for W.

3.3 Other attacks

There exist other implicit trivial attacks. First of all, TLS allows the parties to propose from and choose between various cryptographic primitives and extensions. As a consequence, one can certainly distinguish between parties that implement the protocol differently or do not support the same extensions. This will happen in any protocol that offers "freedom of choice". But for sets of parties that implement the protocol in the same manner, our results apply. Secondly, traffic in the record layer can also be a distinguishing factor, for example by number of messages sent and received during a session. The model by Hevia and Micciancio [21] better captures what happens at the record layer in terms of privacy, what guarantees we have and do not have. In our model, we only analyse the handshake.

4 TLS full handshake mode

In this section, we formally model the privacy of TLS' full handshake mode (unilateral authentication and no resumption). We begin by introducing the concept of a virtual identifier. Next, we formally define the parties and instances involved in the protocol, including their attributes. In the following section, we describe a set of auxiliary functions and lists that allow us to elaborate the oracles and the winning conditions. We provide a theorem and a proof regarding the privacy achieved by Π_{TLS} in this model.

4.1 Virtual identifiers

Central to our model is the concept of virtual identifier, or vid. The adversary has access to an oracle allowing him to bind together two parties, P_i and P_j. The output of the oracle is a string $\mathsf{vid}=P_i|P_j$, the concatenation of the two parties. The adversary will use the vid to interact with the party "behind" it, which is either P_i or P_j, based on a secret bit b. The adversary wins if he correctly identifies b, given certain winning conditions.

The adversary can choose $P_i = P_j$. Such sessions help him learn more about the party and the protocol, but they leak no information about the bit b.

The adversary can create multiple vids, as long as a party is not bound inside two or more "active" vids at a time. He can also "activate" and "deactivate" the same vid multiple times.

4.2 Parties, instances and attributes

Let $\mathcal{C} = \{C_i, C_j ...\}$ be a set of clients and $\mathcal{S} = \{S_i, S_j ...\}$ a set of servers. We denote by $\mathcal{P} = \{P_i, P_j ...\}$ the set of parties, namely the disjoint union $\mathcal{S} \uplus \mathcal{C}$. Each party has the following attributes:

- pk and sk: the public key in the certificate and the corresponding secret key. Clients have undefined certificates ($\perp$);
- corr: a corruption attribute which is initialized to 0 (uncorrupted) and becomes 1 if the adversary has corrupted that party using a query;

A party may run multiple instances. We denote by π_i^s the s^{th} instance of the party P_i. We often substitute i by vid, and π_{vid}^s instantiates the real party behind a vid. Each instance possesses a series of attributes:

- pk, sk and corr, inherited from the real party behind a vid;
- sid, a unique session identifier, used for matching instances. What constitutes the session identifier is protocol specific. In some protocols, one party generates a unique string as the session identifier. In Bellare-Rogaway models, two instances are involved in the same session if their transcript, up to the last message, is the same. In the case of $\mathit{\Pi}_{\mathsf{TLS}}$ and $\mathit{\Pi}_{\mathsf{TLS+res}}$, we will consider the sid to be the concatenation of the nonce of the client and the nonce of the server. Note that this constitues a subset of the transcript;
- pid, the partner of π_i^s, initialized to $\perp$;
- the accept bit, initialized to $\perp$. It takes the value 1 when the instance finishes in an accepting state and 0 if the instance aborts/rejects. A value of 1 also implies that partner authentication succeeded (if it was the case);
- the keys: `C.htk` - the *client handshake traffic key*, `S.htk` - the *server handshake traffic key*, `C.tk` - the *client traffic key* and `S.tk` - the *server traffic key*;
- for all session keys $\mathsf{key} \in \{\texttt{C.htk}, \texttt{S.htk}, \texttt{C.tk}, \texttt{S.tk}\}$, there exists a reveal bit ρ_{key}, set to 1 when the adversary obtains the value of the key.

At the beginning of the privacy game we run an algorithm called $\mathsf{Setup}(\cdot)$. $\mathsf{Setup}(1^\lambda)$ takes as input a security parameter λ in unary notation. From the security parameter we determine the nrsv (number of servers), nrcl (number of clients), and the keyspaces for all cryptographic primitives. We then initialize the parties and create the keys for the servers. We initialize all lists to the empty list and any required cryptographic primitives involved in the protocol. The set $\mathcal{P}$ containing all parties is then given to the adversary.

4.3 Auxiliary functions and lists

We also define a set of auxiliary functions and lists. These are simply tools we use in modelling.

$\mathsf{type}(P_i)$	If $P_i \in \mathcal{S}$, return S. Otherwise return C.
$\mathsf{type}(\mathsf{vid})$	Specifies whether the vid corresponds to a client or a server. Let $\mathsf{vid} = P_i \vert P_j$. If $P_i, P_j \in \mathcal{S}$, return S. Otherwise return C.

$\mathsf{real}(\mathsf{vid})$ Outputs the true party behind a vid. For all i, j, including $i = j$:
If $\mathsf{vid} = P_i|P_j \wedge \mathsf{b} = 0$, return P_i.
If $\mathsf{vid} = P_i|P_j \wedge \mathsf{b} = 1$, return P_j.
The bit b is uniformly randomly sampled by the challenger at the beginning of the privacy game. This function is not accessible by the adversary.

We further denote:

$\mathcal{L}_{\mathsf{vid}}$ The list of active vids. A vid is *active* if it is the output of a $\mathsf{DrawParty}$ and it was not deactivated by a corresponding Free query. We detail these queries in the next section. A party can appear in at most one active vid at a time.
$\mathcal{L}_{\mathsf{act}}$ The list of active parties. A party is *active* if it is part of an active vid.
$\mathcal{L}_{\mathsf{inst}}$ The list of all instances π^s_{vid} ever created.
$\mathcal{L}_{\mathsf{chg}}$ The *challenge list* contains the list of server instances such that their vid is binding distinct servers and they execute a full handshake. Otherwise said, it contains instances π^s_{vid} such that $\mathsf{vid} = S_k|S_l$, with $S_k \neq S_l$.

4.4 Adversarial oracles

The attack capabilities of a probabilistic polynomial time adversary $\mathcal{A}$ are modelled by providing him access to the following oracles (also see pseudocode form in Fig. 11):

- $\underline{\mathsf{DrawParty}^{\mathsf{b}}(P_i, P_j)}$ allows the adversary to obtain a vid binding two parties, activating it. It adds P_i, P_j to $\mathcal{L}_{\mathsf{act}}$, creates a $\mathsf{vid} = P_i|P_j$, adds vid to $\mathcal{L}_{\mathsf{vid}}$ and returns vid. This oracle aborts if $\mathsf{type}(P_i) \neq \mathsf{type}(P_j)$ (they are a server and a client) or one of the parties is already bound in an active vid at the time of the query: $P_i \in \mathcal{L}_{\mathsf{act}} \vee P_j \in \mathcal{L}_{\mathsf{act}}$. We also abort if the adversary queries two distinct clients: $\mathsf{type}(P_i) = \mathsf{type}(P_j) = C \wedge P_i \neq P_j$;
- $\underline{\mathsf{NewSession}(\mathsf{vid}, \mathsf{vid}')}$ creates a new instance of a given active vid that will communicate with a specified partner vid'. It returns a new instance π^s_{vid}, with $\pi^s_{\mathsf{vid}}.\mathsf{pid} = \mathsf{real}(\mathsf{vid}')$. Its other attributes are set to default values. If $\mathsf{vid} = P_i|S_j$ with either $P_i.\mathsf{corr} = 1$ or $P_j.\mathsf{corr} = 1$, then $\pi^s_{\mathsf{vid}}.\mathsf{corr} = 1$. If $\mathsf{vid} = S_k|S_l$ with $S_k \neq S_l$, we add π^s_{vid} to $\mathcal{L}_{\mathsf{chg}}$. We add π^s_{vid} to $\mathcal{L}_{\mathsf{inst}}$ and return π^s_{vid}. This oracle aborts if $\mathsf{vid} \notin \mathcal{L}_{\mathsf{vid}}$, if $\mathsf{vid}' \notin \mathcal{L}_{\mathsf{vid}}$, or if $\mathsf{type}(\mathsf{vid}) = \mathsf{type}(\mathsf{vid}')$. The vids should be active at the time of the oracle call, and they should partner with a party of the opposite type.
- $\underline{\mathsf{Send}(\pi^s_{\mathsf{vid}}, \mathsf{msg})}$ enables an adversary to send the message msg to π^s_{vid} and outputs msg', the next message in the protocol. If π^s_{vid} is a freshly-initialized client instance and msg is the string prompt, π^s_{vid} starts the protocol with $\pi^s_{\mathsf{vid}}.\mathsf{pid}$.
- $\underline{\mathsf{Reveal}(\pi^s_{\mathsf{vid}}, \mathsf{key})}$ returns to the adversary the key $\pi^s_{\mathsf{vid}}.\mathsf{key}$, where $\mathsf{key} \in \{\mathtt{C.htk}, \mathtt{S.htk}, \mathtt{C.tk}, \mathtt{S.tk}\}$, and sets $\pi^s_{\mathsf{vid}}.\rho_{\mathsf{key}}$ to 1.
- $\underline{\mathsf{Corrupt}(P_i)}$ is an oracle that returns $P_i.\mathsf{sk}$ and sets $P_i.\mathsf{corr}$ to 1. This automatically updates any existing and future instances of a vid containing P_i: $\forall \pi^s_{\mathsf{vid}}, \mathsf{vid} = P_i|P_j \vee P_j|P_i, \pi^s_{\mathsf{vid}}.\mathsf{corr} = 1$.

– $\underline{\mathsf{Free}(\mathsf{vid})}$ allows the adversary to release the parties from the binding of a vid and terminates any sessions involving that vid. It removes vid from $\mathcal{L}_{\mathsf{vid}}$ and the corresponding parties from $\mathcal{L}_{\mathsf{act}}$. For all instances of vid and all instances with $\mathsf{pid} = \mathsf{real}(\mathsf{vid})$, if $\pi_i^s.\mathsf{accept} = \perp$, it sets $\pi_i^s.\mathsf{accept} = 0$ (session rejected/aborted).

4.5 Privacy experiment

Roughly speaking, an adversary is capable of winning the privacy game if he is able to distinguish between two parties of his choice, either by identifying them or by studying their behaviour. We formally define the privacy game in Table 2.

$\underline{\mathsf{Exp}^{\mathsf{full.priv}}_{\Pi}(\mathcal{A})\text{:}}$
$\mathsf{Setup}(1^{\lambda});$
$\mathsf{b} \xleftarrow{\$} \{0,1\}$
$\mathsf{d} \leftarrow \mathcal{A}^{\mathsf{DrawParty}^{\mathsf{b}}(\cdot,\cdot),\mathsf{NewSession}(\cdot,\cdot),\mathsf{Send}(\cdot,\cdot),\mathsf{Reveal}(\cdot,\cdot),\mathsf{Corrupt}(\cdot),\mathsf{Free}(\cdot)}$
$\forall \mathsf{vid} \in \mathcal{L}_{\mathsf{vid}}, \mathsf{Free}(\mathsf{vid})$

$\mathcal{A}$ wins if $\mathsf{b} = \mathsf{d}$ and:

- $\forall \pi^s_{\mathsf{vid}} \in \mathcal{L}_{\mathsf{chg}}\ \exists \pi^t_{\mathsf{vid}'} \in \mathcal{L}_{\mathsf{inst}}$ s.t.
  - $\pi^s_{\mathsf{vid}}.\mathsf{sid} = \pi^t_{\mathsf{vid}'}.\mathsf{sid}$
  - $\pi^s_{\mathsf{vid}}.\mathsf{accept} = \pi^t_{\mathsf{vid}'}.\mathsf{accept} = 1$
  - $\forall \mathsf{key} \in \{\mathtt{S.htk}\}, \pi^s_{\mathsf{vid}}.\rho_{\mathsf{key}} = \pi^t_{\mathsf{vid}'}.\rho_{\mathsf{key}} = 0$
  - $\pi^s_{\mathsf{vid}}.\mathsf{corr} = \pi^t_{\mathsf{vid}'}.\mathsf{corr} = 0$

Table 2. Privacy experiment.

The privacy game proceeds in the following way. First, the challenger runs $\mathsf{Setup}(1^{\lambda})$. He then uniformly samples a bit b. The adversary interacts with the challenger, using the oracles that have been given to him. Then the adversary outputs a bit d. The challenger Frees all active vid, thus terminating any ongoing sessions if any still exist. The adversary wins the game if $\mathsf{d} = \mathsf{b}$ (he correctly determined b) and he fulfilled the winning conditions:

We require that for all server instances where the vid binds distinct servers ($\forall \pi^s_{\mathsf{vid}} \in \mathcal{L}_{\mathsf{chg}}$), there exists an honest client instance ($\exists \pi^t_{\mathsf{vid}'} \in \mathcal{L}_{\mathsf{inst}}$) such that the two instances have had a matching conversation ($\pi^s_{\mathsf{vid}}.\mathsf{sid} = \pi^t_{\mathsf{vid}'}.\mathsf{sid}$), they were not trivially Revealed/opened by the adversary ($\forall \mathsf{key} \in \{\mathtt{S.htk}\}, \pi^s_{\mathsf{vid}}.\rho_{\mathsf{key}} = \pi^t_{\mathsf{vid}'}.\rho_{\mathsf{key}} = 0$) and that they both accepted the session ($\pi^s_{\mathsf{vid}}.\mathsf{accept} = \pi^t_{\mathsf{vid}'}.\mathsf{accept} = 1$). This circumvents the first trivial attack, where the adversary creates a session with a challenge server and obtains its certificate. In addition, the adversary cannot corrupt the servers involved in the challenge ($\forall \pi^s_{\mathsf{vid}} \in \mathcal{L}_{\mathsf{chg}}, \pi^s_{\mathsf{vid}}.\mathsf{corr} = \pi^t_{\mathsf{vid}'}.\mathsf{corr} = 0$). This circumvents the second trivial attack, where the adversary impersonates one of the challenge servers to the users.

Definition 1. *The advantage $\epsilon_{\mathsf{full.priv}}$ of an adversary running in time t' to win the game $\mathsf{Exp}^{\mathsf{full.priv}}_{\Pi_{\mathsf{TLS}}}$ is :*

$$\epsilon_{\mathsf{full.priv}} = \left| \mathbb{P}[\mathcal{A} \textit{ wins } \mathsf{Exp}^{\mathsf{full.priv}}_{\Pi_{\mathsf{TLS}}}] - \frac{1}{2} \right|.$$

Theorem 1. *Let $\mathbb{G}$ be a group of order $|\mathbb{G}|$, let 2^t be the size of the nonce space, and let 2^r be the size of the codomain of the RO. The advantage $\epsilon_{\mathsf{full.priv}}$ of an adversary running in time t', interacting with at most nrsv servers, making at*

$\mathsf{Setup}(1^\lambda)$

Compute $\mathsf{nrcl}, \mathsf{nrsv}$ from 1^λ
$\mathcal{P} = \emptyset, \mathcal{L}_{\mathsf{vid}} = \emptyset, \mathcal{L}_{\mathsf{act}} = \emptyset, \mathcal{L}_{\mathsf{inst}} = \emptyset, \mathcal{L}_{\mathsf{chg}} = \emptyset$
for $i := 1$ to nrcl **do**
  $C_i.\mathsf{pk} = \perp$
  $C_i.\mathsf{sk} = \perp$
  $C_i.\mathsf{corr} = 0$
  $\mathcal{P} = \mathcal{P} \cup \{C_i\}$
for $k := 1$ to nrsv **do**
  $(S_k.\mathsf{pk}, S_k.\mathsf{sk}) = \mathtt{Sign}.\mathsf{Gen}()$
  $S_k.\mathsf{corr} = 0$
  $\mathcal{P} = \mathcal{P} \cup \{S_k\}$

$\mathsf{NewSession}(\mathsf{vid}, \mathsf{vid}')$

if $\mathsf{vid} \notin \mathcal{L}_{\mathsf{vid}} \vee \mathsf{vid}' \notin \mathcal{L}_{\mathsf{vid}}$
  return $\perp$
if $\mathsf{type}(\mathsf{vid}) = \mathsf{type}(\mathsf{vid})$
  return $\perp$
$\pi^s_{\mathsf{vid}}.\mathsf{pid} \leftarrow \mathsf{real}(\mathsf{vid}')$
$\pi^s_{\mathsf{vid}}.\mathsf{sid} \leftarrow \perp$
$\pi^s_{\mathsf{vid}}.\mathsf{pk} \leftarrow \mathsf{real}(\mathsf{vid}).\mathsf{pk}$
$\pi^s_{\mathsf{vid}}.\mathsf{sk} \leftarrow \mathsf{real}(\mathsf{vid}).\mathsf{sk}$
$\pi^s_{\mathsf{vid}}.\mathsf{corr} \leftarrow \mathsf{real}(\mathsf{vid}).\mathsf{corr}$
if $\mathsf{vid} = S_i|S_j \wedge (S_i.\mathsf{corr} = 1 \vee S_j.\mathsf{corr} = 1)$
  $\pi^s_{\mathsf{vid}}.\mathsf{corr} = 1$
$\pi^s_{\mathsf{vid}}.\mathsf{freed} \leftarrow 0$
$\pi^s_{\mathsf{vid}}.\mathsf{accept}, \pi^s_{\mathsf{vid}}.\mathsf{C.htk}, \pi^s_{\mathsf{vid}}.\mathsf{S.htk}, \pi^s_{\mathsf{vid}}.\mathsf{C.tk}, \pi^s_{\mathsf{vid}}.\mathsf{S.tk} \leftarrow \perp$
if $\mathsf{vid} = S_k|S_l \wedge S_k \neq S_l$
  $\mathcal{L}_{\mathsf{chg}} \leftarrow \mathcal{L}_{\mathsf{chg}} \cup \mathsf{vid}$
$\mathcal{L}_{\mathsf{inst}} \leftarrow \mathcal{L}_{\mathsf{inst}} \cup \pi^s_{\mathsf{vid}}$
return π^s_{vid}

$\mathsf{Reveal}(\pi^s_{\mathsf{vid}}, \mathsf{key})$

$\pi^s_{\mathsf{vid}}.\mathsf{key} = 1$
return $\pi^s_{\mathsf{vid}}.\mathsf{key}$

$\mathsf{Corrupt}(P_i)$

$P_i.\mathsf{corr} = 1$
$\forall \pi^s_{\mathsf{vid}}, \mathsf{vid} = P_i|P_j \vee \mathsf{vid} = P_j|P_i$
  $\pi^s_{\mathsf{vid}}.\mathsf{corr} = 1$
return $P_i.\mathsf{sk}$

$\mathsf{DrawParty}(P_i, P_j)$

if $P_i \in \mathcal{L}_{\mathsf{act}} \vee P_j \in \mathcal{L}_{\mathsf{act}}$
  return $\perp$
if $\mathsf{type}(P_i) \neq \mathsf{type}(P_j)$
  return $\perp$
if $\mathsf{type}(P_i) = \mathsf{type}(P_j) = C \wedge P_i \neq P_j$
  return $\perp$
$\mathcal{L}_{\mathsf{act}} \leftarrow \mathcal{L}_{\mathsf{act}} \cup \{P_i, P_j\}$
$\mathcal{L}_{\mathsf{vid}} \leftarrow \mathcal{L}_{\mathsf{vid}} \cup \mathsf{vid}$
$\mathsf{vid} \leftarrow P_i|P_j$
return vid

$\mathsf{Send}(\pi^s_{\mathsf{vid}}, \mathsf{msg})$

if $\pi^s_{\mathsf{vid}}.\mathsf{freed} = 1$
  return $\perp$
if $\mathsf{msg} = \mathsf{prompt} \wedge \mathsf{type}(\mathsf{vid}) = C \wedge \pi^s_{\mathsf{vid}}.\mathsf{sid} = \perp$
  Update $\pi^s_{\mathsf{vid}}.\mathsf{sid}$
  return msg'(Start protocol with $\pi^s_{\mathsf{vid}}.\mathsf{pid}$)
if msg is valid
  Update $\pi^s_{\mathsf{vid}}.\mathsf{sid}$
  return msg'
else return $\perp$

$\mathsf{Free}(\mathsf{vid})$

$\mathcal{L}_{\mathsf{vid}} = \mathcal{L}_{\mathsf{vid}} - \mathsf{vid};$
if $\mathsf{vid} = P_i|P_j$
  $\mathcal{L}_{\mathsf{act}} = \mathcal{L}_{\mathsf{act}} - \{P_i, P_j\}$
for $\pi^s_{\mathsf{vid}} \in \mathcal{L}_{\mathsf{inst}}$
  if $\pi^s_{\mathsf{vid}}.\mathsf{accept} = \perp$
    $\pi^s_{\mathsf{vid}}.\mathsf{accept} = 0$
  $\pi^s_{\mathsf{vid}}.\mathsf{freed} = 1$
  for $\pi^t_{\mathsf{vid}'} \in \mathcal{L}_{\mathsf{inst}}$
    if $\pi^s_{\mathsf{vid}}.\mathsf{sid} = \pi^t_{\mathsf{vid}'}.\mathsf{sid} \wedge \pi^t_{\mathsf{vid}'}.\mathsf{accept} = \perp$
      $\pi^t_{\mathsf{vid}'}.\mathsf{accept} = 0$
      $\pi^t_{\mathsf{vid}'}.\mathsf{freed} = 1$

Fig. 11. Adversarial oracles in the full handshake mode model.

most q_i queries to NewSession *and at most q_{ro} queries to* RO *and q' queries to all its oracles is:*

$$\epsilon_{\mathsf{full.priv}} \leq \frac{q_i^2}{2^t} + \frac{q_i^2}{|\mathbb{G}|} + \frac{q_{ro}^2}{2^r} + q_{ro}\epsilon_{\mathsf{CDH}} + 4q_i\epsilon_{\mathsf{prf}} + 2q_i\epsilon_{\mathsf{stLHAE}} + \frac{1}{\mathsf{nrsv}}\epsilon_{\mathsf{EUF\text{-}CMA}},$$

where ϵ_{CDH}, ϵ_{prf}, $\epsilon_{\mathsf{stLHAE}}$, $\epsilon_{\mathsf{EUF\text{-}CMA}}$ represent the maximum advantage of an adversary against the CDH, prf, stLHAE, *and* EUF-CMA *respectively.*

4.6 Proof of Theorem 1

An *honest* instance is an instance that was generated as a result of a NewSession query and is not under adversarial control (the adversary has not substituted the messages it has generated or compromised the instance in any way).

$\mathbb{G}_0$: The original privacy game $\mathsf{Exp}^{\mathsf{full.priv}}_{\Pi_{\mathsf{TLS}}}$.

$\mathbb{G}_1$: We abort $\mathbb{G}_0$ if two honest instances generate the same nonce. This ensures that no two sessions have the same sid. Once the server receives the first message in the protocol, we can identify which two instances are supposed to form a session together. There are at most $q_i(q_i-1)/2$ pairs of instances, and the probability of a collision occurring is $1/2^t$, where 2^t is the size of the nonce space:

$$\mathbb{P}[\mathcal{A} \text{ wins } \mathbb{G}_0] \leq \mathbb{P}[\mathcal{A} \text{ wins } \mathbb{G}_1] + q_i^2/2^t.$$

$\mathbb{G}_2$: We abort the game if two sessions between honest instances obtain the same Diffie-Hellman value g^{xy}. There are at most $q_i(q_i-1)/2$ pairs of sessions (identifying a session by its client instance) and with a $1/|\mathbb{G}|$ chance of a collision. To obtain the chance of a collision, consider $g^x, g^y, g^{x'}$ fixed. Then $g^{y'}$ must be equal to $g^{xy-x'}$ in order to have a collision, and there is a $1/|\mathbb{G}|$ chance of this occurring:

$$\mathbb{P}[\mathcal{A} \text{ wins } \mathbb{G}_1] \leq \mathbb{P}[\mathcal{A} \text{ wins } \mathbb{G}_2] + q_i^2/|\mathbb{G}|.$$

$\mathbb{G}_3$: We abort the game if two hs collide on different input to the RO. Coupled with $\mathbb{G}_1$, this ensures that two instances obtain the same hs if and only if they provide the same input to the RO:

$$\mathbb{P}[\mathcal{A} \text{ wins } \mathbb{G}_2] \leq \mathbb{P}[\mathcal{A} \text{ wins } \mathbb{G}_3] + q_{ro}^2/2^r.$$

$\mathbb{G}_4$: The same as $\mathbb{G}_3$, but we abort if the adversary queries $\mathsf{RO}(g^{xy}, H_{\mathsf{sid}})$, where g^{xy} corresponds to values g^x and g^y generated by honest instances. If $\mathcal{A}$ would make this query, we show how to construct an adversary $\mathcal{R}_{\mathsf{CDH}}$ against CDH.

$\mathcal{R}_{\mathsf{CDH}}$ initializes the privacy game and guesses a client instance π_C^s. Let π_S^t be its partnering server instance. We essentially guess the session the adversary is trying to compromise by computing the hs by himself, in an attempt to win the privacy game. Next, $\mathcal{R}_{\mathsf{CDH}}$ answers most oracle queries for $\mathcal{A}$ as defined in the privacy game. However, when answering $\mathsf{Send}(\pi_C^s, \mathsf{prompt})$ and $\mathsf{Send}(\pi_S^t, \mathsf{Send}(\pi_C^s, \mathsf{prompt}))$, $\mathcal{R}_{\mathsf{CDH}}$ inserts the values g^x and g^y given by his own challenger. Since $\mathcal{R}_{\mathsf{CDH}}$ cannot compute hs, he instead uniformly randomly samples a value to use as the handshake secret, to which the restriction of $\mathbb{G}_3$ also applies.

As per our assumption, $\mathcal{A}$ will query $\mathsf{RO}(g^{xy}, H_{\mathsf{sid}})$, but we cannot identify which one is correct, so we have to guess which query contains the g^{xy}. We forward our guess to the challenger and win with a $1/q_{ro}$ chance of success.

$$\mathbb{P}[\mathcal{A} \text{ wins } \mathbb{G}_3] \leq \mathbb{P}[\mathcal{A} \text{ wins } \mathbb{G}_4] + q_{ro}\epsilon_{\mathsf{CDH}}.$$

$\mathbb{G}_5$: We replace the $\mathsf{C.htk}$ and $\mathsf{S.htk}$ computations with calls to a truly random function f with the same domain and codomain as $\mathtt{PRF}$. This means that keys can be seen as independent from the context that generated them.

We go session by session and key by key using a hybrid argument. We present an intermediate hop. The adversary $\mathcal{R}_{\mathsf{prf}}$ against PRF simulates the privacy game for the game distinguisher. For the first i sessions, the function f is used. For the $i+2^{th}$ session onwards, $\mathtt{PRF}$ is used. To obtain the $\mathsf{C.htk}$ or $\mathsf{S.htk}$ for the $i+1^{th}$ session, $\mathcal{R}_{\mathsf{prf}}$ queries his own challenger and uses that value as the $\mathsf{C.htk}$ or $\mathsf{S.htk}$. $\mathcal{R}_{\mathsf{prf}}$ forwards the guess bit of the distinguisher to his own challenger:

$$\mathbb{P}[\mathcal{A} \text{ wins } \mathbb{G}_4] \leq \mathbb{P}[\mathcal{A} \text{ wins } \mathbb{G}_5] + 4q_i\epsilon_{\mathsf{prf}}.$$

$\mathbb{G}_6$: We abort if the adversary injects, in any of the challenge sessions, a ciphertext that decrypts incorrectly. Since doing so would make him lose the game, the probability of winning the game remains unchanged:

$$\mathbb{P}[\mathcal{A} \text{ wins } \mathbb{G}_5] = \mathbb{P}[\mathcal{A} \text{ wins } \mathbb{G}_6].$$

$\mathbb{G}_7$: We abort the game if the adversary submits a new, valid ciphertext in a challenge session. We use this to build an adversary $\mathcal{R}_{\mathsf{stLHAE}}$ against stLHAE. The adversary $\mathcal{R}_{\mathsf{stLHAE}}$ guesses the session in which $\mathcal{A}$ will inject the encrypted message. $\mathcal{R}_{\mathsf{stLHAE}}$ simulates $\mathbb{G}_6$ but, for that particular session, he asks his challenger to encrypt any messages $(m_0 = m_1)$. When $\mathcal{A}$ injects the new encryption, $\mathcal{R}_{\mathsf{stLHAE}}$ forwards this message to ADec, having made sure the oracle is in a consistent state. $\mathcal{R}_{\mathsf{stLHAE}}$ answers his challenge depending if the output is ADec is $\perp$ or a message.

$$\mathbb{P}[\mathcal{A} \text{ wins } \mathbb{G}_6] \leq \mathbb{P}[\mathcal{A} \text{ wins } \mathbb{G}_7] + q_i\epsilon_{\mathsf{stLHAE}}.$$

$\mathbb{G}_8$: In a challenge session, when a server instance has to send the encrypted message $(\mathsf{Cert}_S, \mathsf{CVf})$, we replace it with an encryption of the bit 0 of matching length. This change also propagates throughout the transcript of the session. We use a hybrid argument, going session by session. If a hop distinguisher exists, we can build an adversary $\mathcal{R}_{\mathsf{stLHAE}}$ against stLHAE. We describe an intermediate hop.

Consider that we have made this transition in the first i sessions. For the $i+1^{th}$ session, $\mathcal{R}_{\mathsf{stLHAE}}$ simulates $\mathbb{G}_{8.i}$ but, sends to his encryption oracle $m_0 = (\mathsf{Cert}_S, \mathsf{CVf})$ and $m_1 = \{0\}$, receives a ciphertext and returns that as the Send answer. He proceeds analogously for Fin_S, which has to be encrypted with the same unknown key. $\mathcal{R}_{\mathsf{stLHAE}}$ has to simulate the challenge sessions without decrypting and doing consistency checks.

Due to $\mathbb{G}_6$ and $\mathbb{G}_7$, we reject all new ciphertexts we see on the network.

At this point, the adversary has no other choice but to faithfully relay messages, if he hopes to win. In particular, it implies $\mathcal{R}_{\mathsf{stLHAE}}$ can successfully simulate the privacy game even in the absence of a decryption oracle, to which we have limited access (due to the way the stLHAE game works).

$\mathcal{R}_{\mathsf{stLHAE}}$ forwards the guess bit of the $\mathbb{G}_7/\mathbb{G}_8$ distinguisher to win the game.

$$\mathbb{P}[\mathcal{A} \text{ wins } \mathbb{G}_7] \leq \mathbb{P}[\mathcal{A} \text{ wins } \mathbb{G}_8] + q_i \epsilon_{\mathsf{stLHAE}}.$$

$\mathbb{G}_9$: We abort if the adversary successfully forges the certificate of a challenge server, in an attempt to impersonate it to a client. We guess the server the adversary will try to impersonate. For this server, instead of generating the pk and the sk, the adversary against EUF-CMA will use the pk and the Sign oracle in order to simulate the privacy game. When the adversary sends the forged certificate to an honest client, $\mathcal{R}_{\mathsf{EUF\text{-}CMA}}$ forwards it to the challenger in the EUF-CMA game.

$$\mathbb{P}[\mathcal{A} \text{ wins } \mathbb{G}_8] \leq \mathbb{P}[\mathcal{A} \text{ wins } \mathbb{G}_9] + \frac{1}{\mathsf{nrsv}} \epsilon_{\mathsf{EUF\text{-}CMA}}.$$

$\mathbb{G}_{10}$: We abort if the adversary succeeds in forging a Fin_S or Fin_C message to an honest instance. However, in order to for the instance to accept the message, the instance must also verify the integrity of the encryption. Or, this is already captured by the previous game hops. Fin_S is protected by S.htk, and the adversary would need to decrypt the previous messages in the protocol in order to create the forged C.htk(and he has no access to S.htk, as per the winning conditions). The probability of winning the game remains unchanged.

$$\mathbb{P}[\mathcal{A} \text{ wins } \mathbb{G}_9] \leq \mathbb{P}[\mathcal{A} \text{ wins } \mathbb{G}_{10}].$$

At this point, the adversary can only guess the challenge bit, therefore $\mathbb{P}[\mathcal{A} \text{ wins } \mathbb{G}_{10}] = \frac{1}{2}$. We obtain the theorem by combining the intermediate results.

5 TLS with session resumption

Session resumption is an integral feature of the TLS protocol. At the end of a handshake, a server can choose to send one or multiple tickets to a client. The client may then use these tickest to "jumpstart" their next sessions. In our extended model, we limit the adversary to a single pair of clients or servers that he can challenge in the game. Once he makes a valid query of the form $\mathsf{DrawParty}(P_i, P_j)$ with $P_i \neq P_j$ and $\mathsf{type}(P_j) = \mathsf{type}(P_j)$, we register the P_i, P_j parties as "challenged". The adversary is not allowed to compromise these parties, their instances or their partnering parties/instances. He can fully compromise all other parties and sessions instead.

Remember the trivial attack where Alice has a ticket, Bob does not, and an adversary can distinguish them based on their (in)ability to resume a session. For this reason, we make sure that parties have the same number of tickets given to/received from all parties when creating a $\mathsf{DrawParty}$ query and when releasing the challenge vid using Free.

5.1 Global Lists

In our model, we introduce two new lists, $\mathcal{L}_{\mathsf{chg.pty}}$ and $\mathcal{L}_{\mathsf{chg.vid}}$ and modify the definition of $\mathcal{L}_{\mathsf{chg}}$.

$\mathcal{L}_{\mathsf{chg.pty}}$ The list of parties the adversary queried as $\mathsf{DrawParty}(P_i, P_j)$ by the adversary, with $P_i \neq P_j$. sessions.

$\mathcal{L}_{\mathsf{chg.vid}}$ The list of vids to which particular winning conditions apply. When the adversary queries for the first time $\mathsf{DrawParty}(P_i, P_j)$ with $P_i \neq P_j$, we register in this list $P_i|P_j$, $P_j|P_i$, $P_i|P_i$, $P_j|P_j$.

$\mathcal{L}_{\mathsf{chg}}$ A subset of $\mathcal{L}_{\mathsf{inst}}$, namely instances π_s^{vid} where $\mathsf{vid} \in \mathcal{L}_{\mathsf{chg.vid}}$.

5.2 Local ticket management

Each party stores internally information about valid unused tickets. We refer to this data as $\mathcal{L}_{\mathsf{tickets}}$ and call $P_i.\mathcal{L}_{\mathsf{tickets}}$ the list associated to P_i. If any change occurs to the locally-stored list $P_i.\mathcal{L}_{\mathsf{tickets}}$, then all instances of P_i will instantly have access to the new list. The elements of $P_i.\mathcal{L}_{\mathsf{tickets}}$ contain entries of the type $(\mathsf{STicket}, \mathsf{rms}, \mathsf{N}_T, S)$, detailed below. Server parties will store $\perp$ for any attribute except the $\mathsf{STicket}$.

$\mathsf{STicket}$: the value of the ticket sent by the server to the client. The client will resend this string when resuming. The server will decrypt and authenticate this string and, if this succeeds, he will use the information stored inside to compute the pre-shared key.

rms: the value of the resumption master secret.

N_T: the nonce used to compute the pre-shared key.

S: The identity of the server who created the ticket (this is used by the client to select the right ticket to resume).

Additionally, every instance stores two ticket tuples: the so called input ticket $(\mathsf{in.STicket}, \mathsf{in.rms}, \mathsf{in.N}_T, \mathsf{psk})$ referring to the ticket used in the resumption and output ticket $(\mathsf{out.STicket}, \mathsf{out.rms}, \mathsf{out.N}_T)$, the ticket created/received at the end of the handshake. The prefixes in and out distinguish between the two tuples. The psk is part of the key scheduling, but we don't need to compute it at the end of session, so there is no need for an $\mathsf{out.psk}$ attribute. Also see Figure 12.

$$\pi^s_{\mathsf{vid}} \xrightarrow{\mathsf{in.STicket}} \pi^t_{\mathsf{vid}'}$$

$$\xleftarrow{\{\ldots, \mathsf{out.STicket}\}_{\mathsf{S.tk}}}$$

Fig. 12. An intuitive figure to illustrate the in.STicket and out.STicket. π^s_{vid} is a client instance and $\pi^t_{\mathsf{vid}'}$ is a server instance.

5.3 Additional attributes and lists

In addition to the previous lists, we also introduce the following attributes for both parties and instances.

- The handshake mode mode is an instance-specific attribute. There are three modes: $\mathsf{mode} \in \{\mathsf{dhe}, \mathsf{psk}, \mathsf{psk} + \mathsf{dhe}\}$, corresponding to a full handshake mode, pre-shared key only mode, or a pre-shared key with Diffie-Hellman key exchange. This attributes indicates which protocol the instance is or will be following.
- k, a server party attribute, is the symmetric encryption key that the server uses to encrypt and decrypt the tickets. This is $\perp$ for clients.
- a (server) party attribute $\mathsf{corr_k}$. Initialized to 0, it becomes 1 when the adversary obtains the long term encryption key of the server using a TCorrupt query (defined below). For clients, this is $\perp$.

left(vid)	If $\mathsf{vid} = P_i \vert P_j$, returns P_i.
right(vid)	If $\mathsf{vid} = P_i \vert P_j$, returns P_j.
$\mathsf{count}(P_i, P_j)$	Returns the number of tuples (tickets) in $P_i.\mathcal{L}_{\mathsf{tickets}}$ received from the P_j (if P_i is a client and P_j is a server) or given to P_j (if P_i is a server and P_j is a client).

We add one extra line to the $\mathsf{Setup}(\cdot, \cdot)$ algorithm: $\mathsf{chg} = 0$. This is a flag we will set once the adversary makes a DrawParty query using two distinct parties (of the same type).

5.4 Adversarial oracles

We define privacy in terms of a game similar to that one defined in the full mode. Indeed, the privacy game for resumption can be viewed as an extension of the much-simpler notion that we used on the full mode (cf. Section 4). The adversary interacts with the system via oracles, as before. We briefly recall the purpose of each oracle and mention below only the *changes* and *additions* we make to those oracles.

- $\mathsf{DrawParty}^{\mathsf{b}}(P_i, P_j)$ outputs a vid binding two parties and activates it. In this model, the adversary is only allowed to query only one vid of the form $P_i|P_j$ with $P_i \neq P_j$. We call this the challenge vid. We also allow creating the reverse of the challenge vid, namely $P_j|P_i$. When creating (or reactivating) a challenge vid, the query will return $\perp$ if the parties do not have the same number of tickets. Succesive reactivations of the challenge vid or it reverse are permitted, as long as the condition regarding the number of tickets holds true. If $\mathsf{chg} = 0$ and $P_i \neq P_j$, we set chg to 1 and register P_i, P_j in $\mathcal{L}_{\mathsf{chg}}$. If $\mathsf{chg} = 1$ and $P_i \neq P_j$, we abort if $P_i \vee P_j \notin \mathcal{L}_{\mathsf{chg}}$. If type $(\mathsf{vid}) = S$, if there exists a client C_k such that $\mathsf{count}(\mathsf{left}(\mathsf{vid}), C_k) \neq \mathsf{count}(\mathsf{right}(\mathsf{vid}), C_k)$, we abort. If type $(\mathsf{vid}) = C$, if there exists a server S_k such that $\mathsf{count}(\mathsf{left}(\mathsf{vid}), S_k) \neq \mathsf{count}(\mathsf{right}(\mathsf{vid}), S_k)$, we abort.

- $\underline{\mathsf{NewSession}(\mathsf{vid}, \mathsf{vid}', \mathsf{mode})}$ serves to create a new instance of a given active vid that will communicate with a partner vid'. If the vid is a client, type $(\mathsf{vid}) = C$, we set $\pi^s_{\mathsf{vid}}.\mathsf{mode}{=}\mathsf{mode}$. If resuming, we also select a ticket from real $(\mathsf{vid}).\mathcal{L}_{\mathsf{tickets}}$ such that real $(\mathsf{vid}'){=}S$. We set $\pi^s_{\mathsf{vid}}.\mathsf{in.rms}$, $\pi^s_{\mathsf{vid}}.\mathsf{in.N}_T$ to the values from the tuple and compute $\pi^s_{\mathsf{vid}}.\mathsf{in.psk}$ using the two values. We abort if resumption is demanded but no ticket is available in $(\mathsf{vid}).\mathcal{L}_{\mathsf{tickets}}$.
- $\underline{\mathsf{Send}(\pi^s_{\mathsf{vid}}, \mathsf{msg})}$ enables an adversary to send the message msg to π^s_{vid} and outputs msg', the next message in the protocol. Also allows ticket creation. If π^s_{vid} is a server instance receiving its first message, we set $\pi^s_{\mathsf{vid}}.\mathsf{mode}$ accordingly. If π^s_{vid} is a server instance that has accepted the session key ($\pi^s_{\mathsf{vid}}.\mathsf{accept} = 1$) and msg is the string prompt, then the server creates and sends a new session ticket to the client, as defined by the protocol.
- $\underline{\mathsf{TCorrupt}(P_i)}$ is a new, resumption-specific oracle. It allows the adversary to obtain the long-term ticket encryption key of a server. Therefore, it returns $P_i.\mathsf{k}$ and sets $P_i.\mathsf{corr_k}$ to 1.
- $\underline{\mathsf{Free}(\mathsf{vid})}$ Inactivates a vid. In case of a challenge vid or its reverse, we delete the minimum of tickets to make the parties indistinguishable by number of tickets given to or received from another party.

5.5 Privacy experiment

Informally speaking, it should be impossible for an adversary to distinguish between two parties, even when adding the possibility of session resumption. We formally define the privacy experiment $\mathsf{Exp}^{\mathsf{res.priv}}_{\Pi_{\mathsf{TLS+res}}}(\mathcal{A})$ in Figure 3. The privacy game consists in an interaction between a challenger and an adversary as described before.

$\mathsf{Exp}^{\mathsf{res.priv}}_{\Pi_{\mathsf{TLS+res}}}(\mathcal{A})$:

$\mathsf{Setup}(1^\lambda)$;

$\mathsf{b} \xleftarrow{\$} \{0,1\}$

$\mathsf{d} \leftarrow \mathcal{A}^{\mathsf{DrawParty}^{\mathsf{b}}(\cdot,\cdot),\mathsf{NewSession}(\cdot,\cdot),\mathsf{Send}(\cdot,\cdot),\mathsf{Reveal}(\cdot,\cdot),\mathsf{Corrupt}(\cdot),\mathsf{TCorrupt}(\cdot),\mathsf{Free}(\cdot)}$

$\forall \mathsf{vid} \in \mathcal{L}_{\mathsf{act}}, \mathsf{Free}(\mathsf{vid})$

$\mathcal{A}$ wins if $\mathsf{b} = \mathsf{d}$ and:

- If $\mathcal{L}_{\mathsf{chg.pty}}$ contains two clients:
  - $\forall \pi^s_{\mathsf{vid}} \in \mathcal{L}_{\mathsf{chg}}$
    - $\pi^s_{\mathsf{vid}}.\rho_{\mathsf{S.tk}} = 0$
  - $\forall \pi^t_{\mathsf{vid}'} \in \mathcal{L}_{\mathsf{inst}}$ s.t.$\exists \pi^s_{\mathsf{vid}} \in \mathcal{L}_{\mathsf{chg}}, \pi^s_{\mathsf{vid}}.\mathsf{sid} = \pi^t_{\mathsf{vid}'}.\mathsf{sid}$
    - $\pi^t_{\mathsf{vid}'}.\rho_{\mathsf{S.tk}} = \pi^t_{\mathsf{vid}'}.\mathsf{corr} = \pi^t_{\mathsf{vid}'}.\mathsf{corr_k} = 0$
- If $\mathcal{L}_{\mathsf{chg.pty}}$ contains two servers:
  - $\forall \pi^s_{\mathsf{vid}} \in \mathcal{L}_{\mathsf{chg}}, \pi^s_{\mathsf{vid}}.\rho_{\mathsf{S.tk}} = 0$
    - $\pi^t_{\mathsf{vid}'}.\rho_{\mathsf{S.tk}} = \pi^t_{\mathsf{vid}'}.\mathsf{corr} = \pi^t_{\mathsf{vid}'}.\mathsf{corr_k} = 0$
    - If $\pi^s_{\mathsf{vid}}.\mathsf{mode} \neq \mathsf{dhe}, \pi^s_{\mathsf{vid}}.\mathsf{accept} = 1$
  - $\forall \pi^t_{\mathsf{vid}'} \in \mathcal{L}_{\mathsf{inst}}$ s.t. $\exists \pi^s_{\mathsf{vid}} \in \mathcal{L}_{\mathsf{chg}}, \pi^s_{\mathsf{vid}}.\mathsf{sid} = \pi^t_{\mathsf{vid}'}.\mathsf{sid}$
    - $\pi^t_{\mathsf{vid}'}.\rho_{\mathsf{S.tk}} = 0$
  - $\forall \pi^s_{\mathsf{vid}} \in \mathcal{L}_{\mathsf{chg}}$ s.t. $\pi^s_{\mathsf{vid}}.\mathsf{mode} = \mathsf{dhe} \exists \pi^t_{\mathsf{vid}'} \in \mathcal{L}_{\mathsf{inst}}$ s.t.
    - $\pi^s_{\mathsf{vid}}.\mathsf{sid} = \pi^t_{\mathsf{vid}'}.\mathsf{sid}$
    - $\pi^s_{\mathsf{vid}}.\mathsf{accept} = \pi^t_{\mathsf{vid}'}.\mathsf{accept} = 1$
    - $\forall \mathsf{key} \in \{\mathtt{S.htk}\}, \pi^s_{\mathsf{vid}}.\rho_{\mathsf{key}} = \pi^t_{\mathsf{vid}'}.\rho_{\mathsf{key}} = 0$

Table 3. Resumption privacy experiment.

The winning conditions can be described informally as follows. We use $\mathcal{L}_{\mathsf{chg}}$ to identify the sessions that are "challenged" and to which certain restrictions apply. For example, for all sessions of challenge servers ($\forall \pi^s_{\mathsf{vid}} \in \mathcal{L}_{\mathsf{chg}}$), we look for their matching client instances ($\forall \pi^t_{\mathsf{vid}'} \in \mathcal{L}_{\mathsf{inst}}$ s.t.$\exists \pi^s_{\mathsf{vid}} \in \mathcal{L}_{\mathsf{chg}}, \pi^s_{\mathsf{vid}}.\mathsf{sid} = \pi^t_{\mathsf{vid}'}.\mathsf{sid}$). For client instances, the $\mathsf{S.tk}$ key must be fresh($\pi^s_{\mathsf{vid}}.\rho_{\mathsf{S.tk}} = 0$). For server instances, their $\mathsf{S.tk}$ key must be fresh and their long term keys must be uncorrupted ($\pi^s_{\mathsf{vid}}.\mathsf{corr} = \pi^s_{\mathsf{vid}}.\mathsf{corr_k} = 0$). If the adversary is trying to attack to distinguish between two servers, two additional constraints apply. First of all, the winning conditions from the first model apply (last white bullet point) to all challenge full handshake sessions($\pi^s_{\mathsf{vid}}.\mathsf{mode} = \mathsf{dhe}$). Additionally, all resuming challenge server instances must accept the session(If $\pi^s_{\mathsf{vid}}.\mathsf{mode} \neq \mathsf{dhe}, \pi^s_{\mathsf{vid}}.\mathsf{accept} = 1$). This prevents ticket redirection attacks.

Definition 2. *The advantage $\epsilon_{\mathsf{full.priv}}$ of an adversary running in time t' to win the game $\mathsf{Exp}^{\mathsf{full.priv}}_{\Pi_{\mathsf{TLS+res}}}$ is :*

$$\epsilon_{\mathsf{full.priv}} = \left| \mathbb{P}[\mathcal{A} \ wins \ \ \mathsf{Exp}^{\mathsf{full.priv}}_{\Pi_{\mathsf{TLS+res}}}] - \frac{1}{2} \right|.$$

$\mathsf{Setup}(1^\lambda)$

Compute $\mathsf{nrcl}, \mathsf{nrsv}$ from 1^λ
$\mathcal{P} = \emptyset, \mathcal{L}_{\mathsf{vid}} = \emptyset, \mathcal{L}_{\mathsf{act}} = \emptyset, \mathcal{L}_{\mathsf{inst}} = \emptyset, \mathcal{L}_{\mathsf{chg}} = \emptyset$
for $i := 1$ to nrcl **do**
  $C_i.\mathsf{pk} = \perp, C_i.\mathsf{sk} = \perp, C_i.\mathsf{corr} = 0$
  $\mathcal{P} = \mathcal{P} \cup \{C_i\}$
for $k := 1$ to nrsv **do**
  $(S_k.\mathsf{pk}, S_k.\mathsf{sk}) = \mathsf{Sig.Gen}()$
  $S_k.\mathsf{corr} = 0$
  $\mathcal{P} = \mathcal{P} \cup \{S_k\}$
$\mathsf{chg} = 0$

$\mathsf{NewSession}(\mathsf{vid}, \mathsf{vid}', \mathsf{mode})$

if $\mathsf{vid} \notin \mathcal{L}_{\mathsf{vid}} \vee \mathsf{vid}' \notin \mathcal{L}_{\mathsf{vid}}$
  return $\perp$
if $\mathsf{type}(\mathsf{vid}) = \mathsf{type}(\mathsf{vid})$
  return $\perp$
$\pi^s_{\mathsf{vid}}.\mathsf{pid} \leftarrow \mathsf{real}(\mathsf{vid}')$
$\pi^s_{\mathsf{vid}}.\mathsf{sid} \leftarrow \perp$
$\pi^s_{\mathsf{vid}}.\mathsf{pk} \leftarrow \mathsf{real}(\mathsf{vid}).\mathsf{pk}$
$\pi^s_{\mathsf{vid}}.\mathsf{sk} \leftarrow \mathsf{real}(\mathsf{vid}).\mathsf{sk}$
$\pi^s_{\mathsf{vid}}.\mathsf{corr} \leftarrow \mathsf{real}(\mathsf{vid}).\mathsf{corr}$
if $\mathsf{vid} = S_i|S_j \wedge (S_i.\mathsf{corr} = 1 \vee S_j.\mathsf{corr} = 1)$
  $\pi^s_{\mathsf{vid}}.\mathsf{corr} = 1$
$\pi^s_{\mathsf{vid}}.\mathsf{freed} \leftarrow 0$
$\pi^s_{\mathsf{vid}}.\mathsf{accept}, \pi^s_{\mathsf{vid}}.\mathsf{C.htk}, \pi^s_{\mathsf{vid}}.\mathsf{S.htk}, \pi^s_{\mathsf{vid}}.\mathsf{C.tk}, \pi^s_{\mathsf{vid}}.\mathsf{S.tk} \leftarrow \perp$
if $\mathsf{vid} = S_k|S_l \wedge S_k \neq S_l$
  $\mathcal{L}_{\mathsf{chg}} \leftarrow \mathcal{L}_{\mathsf{chg}} \cup \mathsf{vid}$
$\mathcal{L}_{\mathsf{inst}} \leftarrow \mathcal{L}_{\mathsf{inst}} \cup \pi^s_{\mathsf{vid}}$
if $\mathsf{type}(\mathsf{vid}) = C$
  $\pi^s_{\mathsf{vid}}.\mathsf{mode} = \mathsf{mode}$
if $\pi^s_{\mathsf{vid}}.\mathsf{mode} \neq \mathsf{dhe} \wedge \mathsf{count}(\mathsf{real}(\mathsf{vid}), \mathsf{real}(\mathsf{vid}')) = 0$
  return $\perp$
if $\pi^s_{\mathsf{vid}}.\mathsf{mode} \neq \mathsf{dhe}$
  $\mathsf{real}(\mathsf{vid}).\mathcal{L}_{\mathsf{tickets}} = \mathsf{real}(\mathsf{vid}).\mathcal{L}_{\mathsf{tickets}} - (\mathsf{STicket}, \mathsf{rms}, \mathsf{t}, \mathsf{real}(\mathsf{vid}'))$
  $\pi^s_{\mathsf{vid}}.\mathsf{in.STicket} = \mathsf{STicket}, \pi^s_{\mathsf{vid}}.\mathsf{in.rms} = \mathsf{rms}, \pi^s_{\mathsf{vid}}.\mathsf{in.N}_T = \mathsf{N}_T$
return π^s_{vid}

$\mathsf{TCorrupt}(P_i)$

$P_i.\mathsf{corr_k} = 1$
$\forall \mathsf{vid}, \mathsf{real}(\mathsf{vid}) = P_i, \pi^s_{\mathsf{vid}}.\mathsf{corr_k} = 1$
return $P_i.\mathsf{k}$

$\mathsf{DrawParty}(P_i, P_j)$

if $P_i \in \mathcal{L}_{\mathsf{act}} \vee P_j \in \mathcal{L}_{\mathsf{act}}$
  return $\perp$
if $\mathsf{type}(P_i) \neq \mathsf{type}(P_j)$
  return $\perp$
if $P_i \neq P_j \wedge \mathsf{chg} = 0$
  $\mathsf{chg} = 1, \mathcal{L}_{\mathsf{chg.pty}} = \{P_i, P_j\}$
if $P_i \neq P_j \wedge \mathsf{chg} = 1$
  if $P_i \notin \mathcal{L}_{\mathsf{chg.pty}} \vee P_j \notin \mathcal{L}_{\mathsf{chg.pty}}$
if $\mathsf{type}(P_i) = \mathsf{type}(P_j) = C$
  if $\exists S_k, \mathsf{count}(\mathsf{left}(\mathsf{vid}), S_k) \neq \mathsf{count}(\mathsf{right}(\mathsf{vid}), S_k)$
    return $\perp$
if $\mathsf{type}(P_i) = \mathsf{type}(P_j) = S$
  if $\exists C_k, \mathsf{count}(\mathsf{left}(\mathsf{vid}), C_k) \neq \mathsf{count}(\mathsf{right}(\mathsf{vid}), C_k)$
    return $\perp$
$\mathcal{L}_{\mathsf{act}} \leftarrow \mathcal{L}_{\mathsf{act}} \cup \{P_i, P_j\}, \mathcal{L}_{\mathsf{vid}} \leftarrow \mathcal{L}_{\mathsf{vid}} \cup \mathsf{vid}$
return $\mathsf{vid} \leftarrow P_i|P_j$

$\mathsf{Send}(\pi^s_{\mathsf{vid}}, \mathsf{msg})$

if $\pi^s_{\mathsf{vid}}.\mathsf{freed} = 1$
  return $\perp$
if $\mathsf{msg} = \mathsf{prompt} \wedge \mathsf{type}(\mathsf{vid}) = C \wedge \pi^s_{\mathsf{vid}}.\mathsf{sid} = \perp$
  Update $\pi^s_{\mathsf{vid}}.\mathsf{sid}$
  return msg'(Start protocol with $\pi^s_{\mathsf{vid}}.\mathsf{pid}$)
if msg is valid
  Update $\pi^s_{\mathsf{vid}}.\mathcal{L}_{\mathsf{tickets}}$ if/as needed
  Update $\pi^s_{\mathsf{vid}}.\mathsf{sid}$
  return msg'
else return $\perp$

$\mathsf{Free}(\mathsf{vid})$

$\mathcal{L}_{\mathsf{vid}} = \mathcal{L}_{\mathsf{vid}} - \mathsf{vid}$;
if $\mathsf{vid} = P_i|P_j$
  $\mathcal{L}_{\mathsf{act}} = \mathcal{L}_{\mathsf{act}} - \{P_i, P_j\}$
for $\pi^s_{\mathsf{vid}} \in \mathcal{L}_{\mathsf{inst}}$
  if $\pi^s_{\mathsf{vid}}.\mathsf{accept} = \perp$
    $\pi^s_{\mathsf{vid}}.\mathsf{accept} = 0$
  $\pi^s_{\mathsf{vid}}.\mathsf{freed} = 1$
  for $\pi^t_{\mathsf{vid}'} \in \mathcal{L}_{\mathsf{inst}}$
    if $\pi^s_{\mathsf{vid}}.\mathsf{sid} = \pi^t_{\mathsf{vid}'}.\mathsf{sid} \wedge \pi^t_{\mathsf{vid}'}.\mathsf{accept} = \perp$
      $\pi^t_{\mathsf{vid}'}.\mathsf{accept} = 0, \pi^t_{\mathsf{vid}'}.\mathsf{freed} = 1$
if $\mathsf{vid} \in \mathcal{L}_{\mathsf{chg.vid}}$ update $\mathcal{L}_{\mathsf{tickets}}$ such that
  $\forall P_k, \mathsf{count}(\mathsf{left}(\mathsf{vid}), P_k) \neq \mathsf{count}(\mathsf{right}(\mathsf{vid}), P_k)$

Fig. 13. Adversarial oracles in the extended model. See Figure 11 for the Reveal and Corrupt oracles.

Theorem 2. *Let $\mathbb{G}$ be a group of order $|\mathbb{G}|$, let 2^t be the size of the nonce space, and let 2^r be the size of the codomain of the* RO*. The advantage $\epsilon_{\mathsf{full.priv}}$ of an adversary running in time t', interacting with at most* nrsv *servers, making at most q_i queries to* $\mathsf{NewSession}$*,q_{Send} queries to* Send*, q_{ro} queries to* RO *and q' queries to all its oracles is:*

$$\epsilon_{\mathsf{res.priv}} \leq \frac{q_i^2}{2^t} + \frac{q_i^2}{|\mathbb{G}|} + \frac{q_{ro}^2}{2^r} + q_{ro}\epsilon_{\mathsf{CDH}} + 8q_i\epsilon_{\mathsf{prf}} + (3q_i + q_{\mathsf{Send}})\epsilon_{\mathsf{stLHAE}} + \frac{1}{\mathsf{nrsv}}\epsilon_{\mathsf{EUF\text{-}CMA}},$$

where ϵ_{CDH}, ϵ_{prf}, $\epsilon_{\mathsf{stLHAE}}$, $\epsilon_{\mathsf{EUF\text{-}CMA}}$represent the maximum advantage of an adversary against the CDH*,* prf*,* stLHAE*, and* $\mathsf{EUF\text{-}CMA}$*respectively.*

5.6 Proof of Theorem 2

A challenge session is any instance in $\mathcal{L}_{\mathsf{chg}}$or any partnering instance (that have matching sid).

$\mathbb{G}_{0'}$: The original privacy game $\mathsf{Exp}_{\Pi}^{\mathsf{res.priv}}$.

$\mathbb{G}_{1'}$: We execute the game steps $\mathbb{G}_0$ - $\mathbb{G}_8$ for sessions executing a full handshake. We apply steps $\mathbb{G}_6$ - $\mathbb{G}_8$ only for challenge sessions.

$$\mathbb{P}[\mathcal{A} \text{ wins } \mathbb{G}_{0'}] \leq \mathbb{P}[\mathcal{A} \text{ wins } \mathbb{G}_{1'}] + \epsilon_{\mathsf{full.priv}}.$$

$\mathbb{G}_{2'}$: We abort if two ms values coincide:

$$\mathbb{P}[\mathcal{A} \text{ wins } \mathbb{G}_{1'}] \leq \mathbb{P}[\mathcal{A} \text{ wins } \mathbb{G}_{2'}] + q_{ro}^2/2^r.$$

$\mathbb{G}_{3'}$: We replace $\mathtt{PRF}$ by a random function when computing rms.

$$\mathbb{P}[\mathcal{A} \text{ wins } \mathbb{G}_{2'}] \leq \mathbb{P}[\mathcal{A} \text{ wins } \mathbb{G}_{3'}] + 2q_i\epsilon_{\mathsf{prf}}.$$

$\mathbb{G}_{4'}$: In challenge sessions, we replace the content of messages encrypting tickets with encryptions of 0, while forwarding (in an out of band manner) the real tickets to the honest parties. This proceeds almost as in $\mathbb{G}_6$ to $\mathbb{G}_8$, with a slight difference. On one hand, we no longer care about parsing invalid message. On the other hand, if the adversary somehow submits a valid ticket (however improbable), he will detect being in a simulation if he fails to resume that ticket.

$\mathbb{G}_{4.1'}$:We abort if the adversary injects a valid ciphertext. We will have to guess the query (and the session) where he will do his first such forgery. We will forward this to the Dec and answer $\mathsf{b} = 0$ if the oracle returns $\perp$ or $\mathsf{b} = 1$ if the oracle succesfully decrypts the forgery.

$\mathbb{G}_{4.2'}$ We substitute ticket contents with 0, while forwarding the real content to the parties in an out of band manner. Then, were he to distinguish between two succesive hops (where the first contains the real ticket, and the second contains the encryption of 0), we would be able to use this adversary in order to win against stLHAE. As before, we simulate the privacy game to the adversary by querying our AEnc oracle. As the $\mathsf{S.tk}$ key must be fresh (as per the winning conditions), we have no problems simulating the game.

$$\mathbb{P}[\mathcal{A} \text{ wins } \mathbb{G}_{3'}] \leq \mathbb{P}[\mathcal{A} \text{ wins } \mathbb{G}_{4'}] + (q_{\mathsf{Send}} + q_i)\epsilon_{\mathsf{stLHAE}}.$$

The adversary cannot impersonate any servers involved in challenge sessions, as their long term keys must be fresh. He can also not inject any bad messages in the record layer of challenge full handshakes, as that would break stLHAE. At this point, all challenge client instances have only legitimate tickets. Additionally, he can no longer link sessions in order to compromise the privacy of either clients or servers. When we replaced the tickets given by the server in the full handshake by encryptions of 0, we have broken the connection between a session and its resumption.

$\mathbb{G}_{5'}$: We abort the game if the adversary redirects a ticket to a server such that the vid of the server who issued the ticket is different from the vid of the server receiving the ticket (ticket redirection attack). However, as per our winning conditions, the adversary will lose the game if does this and he guesses incorectly (so the instance rejects the sessions). Therefore, the probability of the adversary winning the game remains the same.

$$\mathbb{P}[\mathcal{A} \text{ wins } \mathbb{G}_{4'}] = \mathbb{P}[\mathcal{A} \text{ wins } \mathbb{G}_{5'}].$$

$\mathbb{G}_{6'}$: We abort the game if the adversary injects a fresh ticket to a server involved in the challenge. However, as the server keeps a list of all tickets he has created, $\mathcal{L}_{\mathsf{tickets}}$, it will reject any ticket it hasn't created. Bad forgeries will make the adversary lose the game.

Note that, in the real protocol, our maintaining of $\mathcal{L}_{\mathsf{tickets}}$ corresponds to employing anti-replay techniques. The RFC strongly recommends storing some information about past sessions, even in an condensed manner (e.g. a hash of CHello).

$$\mathbb{P}[\mathcal{A} \text{ wins } \mathbb{G}_{5'}] = \mathbb{P}[\mathcal{A} \text{ wins } \mathbb{G}_{6'}].$$

$\mathbb{G}_{7'}$: In any sessions involving a $\mathsf{vid} = P_i|P_j$ with $P_i \neq P_j$ (or succeeding such a session in the resumption chain), we substitute psk with a truly random function.

$$\mathbb{P}[\mathcal{A} \text{ wins } \mathbb{G}_{6'}] \leq \mathbb{P}[\mathcal{A} \text{ wins } \mathbb{G}_{7'}] + 2q_i\epsilon_{\mathsf{prf}}.$$

As psk is an integral part of the key schedule, we must take care not to substitute the psk with a random value in any place where the adversary is allowed to compromise keys, hence the slightly different requirement.

A resumption chain is a succession of sessions, 'linked' to each other by tickets (the ticket created in one is used to start another session). In the case of multiple tickets being created, this can even form a tree of sessions. Once we see a session involving a $\mathsf{vid}= P_i|P_j$ with $P_i \neq P_j$ in a chain, we can start applying the game hop. After this moment, we know these sessions are part of the challenge and can apply the game hop, knowing the adversary cannot open the tickets and detect our substitution. Before this moment, only 'simple' vids are part of the game, and the adversary cannot hope to do a linking attack to win the privacy game, so it's ok if we don't apply this game hop for these sessions (even if they are technically part of the challenge).

After this game hop, the adversary cannot hope to use any ticket he sees on the network to open any challenge session, as the content they encrypt is no longer linked to the key schedule.

Up until this point, we have proven that the transcript or tickets do not allow the adversary to win the privacy game. However, our proof only covers full handshakes and the early steps of their resumption.

$\mathbb{G}_{8'}$: We repeat the earlier games for the resumptions of full handshakes. We then repeat this game for resumptions of resumptions of full handshakes and so on, until all sessions are exhausted. The upper bounds in the earlier steps are large enough to include this game. We make the observations that not all games apply to all sessions, *e.g.* $\mathbb{G}_2$ and $\mathbb{G}_4$ don't apply for sessions that don't execute a Diffie-Hellman key exchange, and games $\mathbb{G}_6$ to $\mathbb{G}_8$ don't apply to resumptions (the servers don't authenticate with a certificate).

6 TLS 1.3 privacy in perspective

Our results show that TLS 1.3, when considered in isolation, does provide some measure of privacy. For full handshakes, the protocol provides a notion of server unlinkability, which must be relaxed in order to account for the server-only authentication of the protocol.

By contrast, session resumption introduces a means of linking sessions between the same two parties. The simple fact of possessing and using a resumption ticket already leaks out some information about a party (the existence of at least one session in the past). We showed in this paper that TLS 1.3 privacy does indeed suffer when resumption is considered; however, this lack of privacy seems inherent to the use of session tickets. In that sense, TLS 1.3 offers an optimal degree of privacy.

The results we prove in this paper depend heavily on how session tickets are implemented. In this paper we included only one such implementation, which is also featured in WolfSSL: namely, the server encrypts the session resumption-state with a long-term symmetric key known only to itself. Alternative approaches are also possible. In our proof, we replace the session ticket in question with a random string of the same length; essentially any other implementation of session tickets for which this can still hold would provide the same degree of privacy.

Interestingly, this rules out session tickets that include public information, such as the session identifier of the session in which we generated the ticket. This would allow the adversary to immediately link that session with the resumed session, thus winning the game. Similarly, just using a counter that is incremented at every session also leads to privacy breaches.

Future work could explore either TLS in conjunction of protocols in the network layer, or features of the TLS that we did not model (more significantly, the SNI extension).TLS 1.3 is run as part of a stack of protocols, not all of which are privacy-preserving. We discuss the limitations of our results in that sense both in Section 1.4 and in Section 1.5. One of the main problems of the encapsulation of TLS messages appears at the network layer. The best bet to achieve better privacy in this context is to use protocols such as Tor; however, to our knowledge, no current result for Tor would allow for a composition with the type of property we are proving here.

Acknowledgements

The authors thank the anonymous reviewers of PETS for their helpful comments. This work was supported in part by the French Agence Nationale de Recherche(ANR) through grant 16 CE39 0012 (SafeTLS).

References

1. David Adrian, Kartihkeyan Bhargavan, Zakir Durumeric, Pierrick Gaudry, Matthew Green, J. Alex Halderman, Nadia Heninger, Drew Springall, Emmanuel Thomé, Benjamin VanderSloot, Eric Wustrow, Santiago Zanella Béguelin, and Paul Zimmermann. Imperfect forward secrecy: How Diffie-Hellman fails in practice. In *Proceedings of ACM CCS 2015*, pages 5–17. IEEE, 2015.
2. Nadhem J. AlFardan, Daniel J. Bernstein, Kenneth G. Paterson, Bertram Poettering, and Jacob C. N. Schuldt. On the security of RC4 in TLS and WPA. In *USENIX Security Symposium*, 2013.
3. Nadhem J. AlFardan and Kenneth G. Paterson. Lucky thirteen: Breaking the TLS and DTLS record protocols. In *IEEE Symposium on Security and Privacy (SP'13)*, 2013.
4. Antoine Delignat-Lavaud and Kartihkeyan Bhargavan. Network-based origin confusion attacks against HTTPS virtual hosting. In *Proceedings of WWW'15*, pages 227–237. Springer, 2015.
5. Nimrod Aviram, Sebastian Schinzel, Juraj Somorovsky, Nadia Heninger, Maik Dankel, Jens Steube, Luke Valenta, David Adrian, J. Alex Halderman, Viktor Dukhovni, Emilia Käsper, Shaanan Cohney, Susanne Engels, Christof Paar, and Yuval Shavitt. Drown: Breaking TLS using SSLv2. `https://drownattack.com`, 2016.
6. Michael Backes, Aniket Kate, Praveen Manoharan, Sebastian Meiser, and Esfandiar Mohammadi. Anoa: A framework for analyzing anonymous communication protocols. In *Proceedings of CSF*. IEEE, 2013.
7. Mihir Bellare and Phillip Rogaway. Entity authentication and key distribution. In *CRYPTO*, pages 232–249, 1993.
8. Benjamin Berdouche, Kartikeyan Bhargavan, Antoine Delignat-Lavaud, Cédric Fournet, Markulf Kohlweiss, Alfredo Pironti, Pierre Yves Strub, and Jean Karim Zinzindohoue. A messy state of the union: Taming the composite state machines of TLS. In *Proceedings of IEEE S&P 2015*, pages 535–552. IEEE, 2015.
9. Benjamin Beurdouche, Karthikeyan Bhargavan, Antoine Delignat-Lavaud, Cédric Fournet, Markulf Kohlweiss, Alfredo Pironti, Pierre-Yves Strub, and Jean Karim Zinzindohoue. A messy state of the union: Taming the composite state machines of TLS. In *Proceedings of IEEE S&P 2015*, pages 535–552. IEEE, 2015.

10. Karthikeyan Bhargavan, Antoine Delignat-Lavaud, Cédric Fournet, Alfredo Pironti, and Pierre-Yves Strub. Triple handshakes and cookie cutters: Breaking and fixing authentication over TLS. In *Proceedings of IEEE S&P 2014*, pages 98–113. IEEE, 2014.
11. Karthikeyan Bhargavan and Gaetan Leurent. Transcript collision attacks: Breaking authentication in TLS, IKE, and SSH. In *Accepted at NDSS 2016, to appear*, 2016.
12. Daniel Bleichenbacher. Chosen ciphertext attacks against protocols based on the RSA encryption standard pkcs #1. In *Proceedings of (CRYPTO'98)*, volume 1462 of *LNCS*, pages 1–12, 1998.
13. Tim Dierks and Eric Rescorla. The transport layer security (TLS) protocol version 1.2. RFC 5246, August 2008.
14. Benjamin Dowling, Marc Fischlin, Felix Günther, and Douglas Stebila. A cryptographic analysis of the TLS 1.3 handshake protocol candidates. In *ACM CCS*, pages 1197–1210, 2015.
15. Nir Drucker and Shay Gueron. Selfie: reflections on tls 1.3 with psk. Cryptology ePrint Archive, Report 2019/347, 2019. `https://eprint.iacr.org/2019/347`.
16. EU. General Data Protection Regulation - GDPR.
17. EU. Regulation on Privacy and Electronic Communications.
18. Marc Fischlin and Felix Günther. Multi-stage key exchange and the case of google's QUIC protocol. In *ACM CCS*, pages 1193–1204, 2014.
19. Pierre-Alain Fouque, Cristina Onete, and Benjamin Richard. Achieving better privacy for the 3gpp aka protocol. In *Proceedings of PETS (PoPETS)*, volume 4, 2016.
20. Christina Garman, Kenneth G. Paterson, and Thyla Van der Merwe. Attacks only get better: Password recovery attacks against RC4 in TLS. In *Proceedings of USENIX 2015*, pages 113–128. USENIX Association, 2015.
21. Alejandro Hevia and Daniele Micciancio. An indistinguishability-based characterization of anonymous channels. In *Proceedings of PETS*, volume 5134 of *LNCS*, pages 24–43. Springer, 2008.
22. Jens Hermans and Andreas Pashalidis and Frederik Vercauteren and Bart Preneel. A New RFID Privacy Model. In *Computer Security - ESORICS 2011 - 16th European Symposium on Research in Computer Security, Leuven, Belgium, September 12-14, 2011. Proceedings*, 2011.
23. Hugo Krawczyk. SIGMA: the 'sign-and-mac' approach to authenticated diffie-hellman and its use in the ike-protocols. In *Advances in Cryptology - CRYPTO 2003, 23rd Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 2003, Proceedings*, pages 400–425, 2003.
24. Hugo Krawczyk. Cryptographic extraction and key derivation: The HKDF scheme. In *Advances in Cryptology — CRYPTO 2010*, volume 6223 of *LNCS*. Springer, 2010.
25. Kenneth G. Paterson, Thomas Ristenpart, and Thomas Shrimpton. Tag size does matter: Attacks and proofs for the TLS record protocol. In *Advances in Cryptology — ASIACRYPT 2011*, volume 7073 of *LNCS*, pages 372–389. Springer-Verlag, 2011.
26. Angelo Prado, Neal Harris, and Yoel Gluck. SSL, gone in 30 seconds: A BREACH beyond CRIME. Black Hat 2013, 2013.
27. Eric Rescorla. The Transport Layer Security (TLS) Protocol Version 1.3. RFC 8446, August 2018.
28. Juliano Rizzo and Thai Duong. The CRIME attack. Ekoparty 2012, 2012.

29. J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, and E. Schooler. SIP: Session Initiation Protocol. RFC 3261, June 2002.
30. Serge Vaudenay. Security flaws induced by CBC padding – applications to SSL, IPSEC, WTLS. In *Proceedings of EUROCRYPT 2002*, volume 2332 of *LNCS*, pages 534–545, 2002.
31. Serge Vaudenay. On privacy models for RFID. In *Advances in cryptology – ASIACRYPT*, volume 4833 of *LNCS*, pages 68–87. Springer, 2007.
32. Wikipedia. Global surveillance disclosures (2013–present).

A TLS and privacy-preservation

Please note that we only focus on aspects that concern privacy, rather than giving a complete analysis of the entire TLS protocol. For more details about the latter, we refer the interested reader to the TLS specifications.

The design of the TLS protocol up to, and including TLS 1.2 was not privacy-centric. Its goal was to simply allow two parties, a client and a server, to securely establish session keys. In the following paragraphs we describe the elements of TLS 1.2 which are relevant to privacy, then underline the differences between how TLS 1.2 and TLS 1.3 handled those elements.

The full handshake. In the full TLS 1.2 handshake, the client and server compute keys from a secret value called a pre-master secret, which they can both compute. The standard gives a choice of using several types of key-exchange methods: RSA, static Diffie-Hellman, ephemeral Diffie-Hellman or Anonymous Diffie-Hellman. In all but the Anonymous Diffie-Hellman key-exchange cipher suite, the server has to provide a certificate for a public key – either used for signatures, or used for public-key encryption. The certificate is sent in clear, allowing sessions featuring the same server to be linked. Moreover, although TLS 1.2 is mostly used in practice with server-only authentication, RFC 5246 does allow the handshake to use mutual authentication as well, in which case the client had to also provide a certificate to be sent in clear.

It is worth noting that although TLS 1.3 does revolutionize the design of modern key-exchange protocols, its core key-exchange algorithm is signed ephemeral Diffie-Hellman, which was also used by TLS 1.2. One key difference is that in TLS 1.3 the certificate is no longer sent in clear, but rather, it is AEAD-encrypted with keys derived from the so-called handshake secret. This has two immediate consequences:

- The certificate, hence server identity, remains confidential as long as the adversary cannot break the AEAD security of the encrypted message. This allows us to formulate some privacy properties for the use of ephemeral Diffie-Hellman in TLS 1.3, however, no such properties can be formulated for TLS 1.2.
- The handshake secret is computed *before* the authentication step takes place. This, ironically, detracts from the privacy of the full handshake, as it opens the door to trivial Man-in-the-Middle attacks which do not harm the security

of the keys, but do affect privacy. This is reflected in our winning conditions for the full-handshake privacy game.

We also note that in giving less choice regarding the key-exchange algorithm, TLS 1.3 actually gains one protocol move, reducing the full handshake to three moves (or 1.5 rounds).

Session resumption. Session resumption was introduced as an orthogonal mechanism that allows TLS 1.2 connections to bypass length authentication and derive fresh session keys based on previously-authenticated keys. In the case of TLS 1.2 the new keys are derived from the resumed session nonces and the previous-session's master secret. In many ways, TLS 1.2 session resumption resembles TLS 1.3 PSK-only session resumption, with a few key differences:

- In TLS 1.3 all session keys (including the pre-shared secret psk value) are computed from the same secrets (early keys are computed from the handshake secret, post-authentication keys from the master secret), but this is done via independent calls to the key-derivation function. This has a dual effect: first nothing is revealed about the master secret in the execution of both handshakes and session resumption; and secondly, learning one computed key does not immediately imply the insecurity of the other keys computed in that session.
- In TLS 1.3 all session keys (including the psk) are computed using the entire protocol transcript, and not just the session nonces. This bypasses attacks such as the Triple Handshake attack [10] or version-downgrade problems like FREAK or LogJam [9,1], which rely on a Man-in-the-Middle changing protocol parameters such that the key remains unaffected.
- A ticket-nonce is added in the computation of a new ticket, to prevent replays.

In terms of privacy, the changes made to the computation of the preshared key psk will allow us to prove stronger privacy statements, by allowing the adversary better corruption and revelation capabilities. However, as we show in this paper, session resumption inherently brings some session linkability. This is how resumption was designed, since the only way to authenticate the resumed key is by linking it to a key established in a fully-authenticated handshake. In this sense, both TLS 1.2 and TLS 1.3 session resumption present serious privacy flaws *despite* not using concrete authentication elements, such as certificates.

In addition to PSK-only resumption, TLS 1.3 also allows session to resume by using PSK-DHE handshakes. In this case, additional freshness is injected, by using two Diffie-Hellman elements, which are in their own turn not authenticated. While this provides a measure of backward security, it does nothing to improve privacy.

The Server Name Indication extension. The SNI extension is indeed a very interesting feature of TLS 1.3, which somehow expands the scope of the privacy game. We did initially want to include this extension in our analysis; however, it soon became clear that the task of defining and quantifying privacy in that context is far from being a trivial extension of our current result. First, note that although non-trivial, it would not be overly hard to extended the model mechanics (syntax, oracles) to capture multiple domains. When defining privacy in that context, however, we would no longer be speaking of server- and client-, but rather server-, client-, and domain-privacy. This raises serious complications in terms of the restrictions on the adversary's actions in the winning conditions, since (a) not all domains exist on all servers(implicitly allowing an adversary to distinguish between potential servers); (b) the fact that the domain name appears in cleartext in the full handshake may implicitly make that domain name traceable if the parties subsequently resume. In our opinion, this topic is relevant and deserves its own paper.

B Cryptographic Experiments

B.1 Stateful length hiding authenticated encryption

A *stateful length-hiding authenticated encryption scheme* (stLHAE) consists of three algorithms: $\mathtt{AE} = (\mathsf{Init}, \mathsf{Enc}, \mathsf{Dec})$. We use the following notations:

k	A symmetric key, sampled uniformly random from a keyspace $\mathcal{K}$;
h	A message header;
msg	A message;
c	A ciphertext;
l	The desired ciphertext length;
stD	State corresponding to decryption;
stE	State corresponding to encryption;
$\mathcal{L}_{\mathsf{cip}}$	An ordered list of ciphertexts. $\mathcal{L}_{\mathsf{cip}}.i$ accesses or modifies the i^{th} ciphertext.

The algorithms are:

- $\underline{\mathsf{Init}()}$: Returns the initial values of stE and stD (deterministic algorithm).
- $\underline{\mathsf{Enc}(k;\ l,\ h,\ \mathsf{msg},\ \mathsf{stE})}$: Returns $\perp$ in case of failure or a ciphertext c of length l. It also returns an updated state stE.
- $\underline{\mathsf{Dec}(k,\ h,\ c,\ \mathsf{stD})}$: Returns $\perp$ in case of failure or a message msg. It also returns an updated state stD.

A $\mathtt{AE}$ scheme is *correct* if the decrypting a sequence of ciphertexts c_i recovers the initial messages msg_i, in the right order, if several initial conditions are met: if a) k was generated by $\mathsf{AE.Gen}$ b) the initial states of stE and stD were generated by Init and c)there exists no $c_i = \perp$ in the sequence.

In the security game, we give the adversary access to two oracles, defined below. We will use two auxiliary global values, i and j, to keep track of the number of encryptions, respectively decryptions, executed. We also use a global variable $\mathsf{in\text{-}sync}$ that takes the value 0 when an inconsistency appears in the decryption oracle.

– $\underline{\mathsf{AEnc}^{\mathsf{b}}(l,\, h,\, \mathsf{msg}_0, \mathsf{msg}_1)}$: Increment i. We encrypt both messages and, if at least one encryption fails, we return $\perp$. Let $(c_{\mathsf{b}}, \mathsf{stE}^{\mathsf{b}})$ be the output of Enc when encrypting the message $\mathsf{msg}_{\mathsf{b}}$ ($\mathsf{msg}_{\mathsf{b}} = \mathsf{msg}_0$ if $\mathsf{b} = 0$, and $\mathsf{msg}_{\mathsf{b}} = \mathsf{msg}_1$ otherwise). We store the ciphertext $\mathcal{L}_{\mathsf{cip}}.i \leftarrow c_{\mathsf{b}}$, update stE=$\mathsf{stE}_{\mathsf{b}}$, and return c_{b} to the adversary.
– $\underline{\mathsf{ADec}^{\mathsf{b}}(h,\, c)}$: If $\mathsf{b} = 0$, return $\perp$. Otherwise, increment j and decrypt c using the header h. If $j > i$ or $\mathcal{L}_{\mathsf{cip}}.j \neq c$ then $\mathsf{in\text{-}sync}$ is set to 0. If $\mathsf{in\text{-}sync}$ equals 0 then output the decrypted message msg, else output $\perp$.

The same oracles, in pseudocode format:

$\mathsf{AEnc}^{\mathsf{b}}(l,\, h,\, \mathsf{msg}_0, \mathsf{msg}_1)$	$\mathsf{ADec}^{\mathsf{b}}(c,\, h)$
$i = i + 1$	$j = j + 1$
$(c_0, \mathsf{stE}_0) \leftarrow \mathtt{AE}.\mathsf{Enc}(k; l, h, \mathsf{msg}_0, \mathsf{stE})$	**if** $\mathsf{b}_0 = \perp$
$(c_1, \mathsf{stE}_1) \leftarrow \mathtt{AE}.\mathsf{Enc}(k; l, h, \mathsf{msg}_1, \mathsf{stE})$	**return** $\perp$
if $c_0 = \perp$	$(\mathsf{msg}, \mathsf{stD}) \leftarrow \mathtt{AE}.\mathsf{Dec}(k; h, c, \mathsf{stD})$
return $\perp$	**if** $j > i$
if $c_1 = \perp$	$\mathsf{in\text{-}sync} = 0$
return $\perp$	**if** $c \neq \mathcal{L}_{\mathsf{cip}}.j$
$\mathcal{L}_{\mathsf{cip}}.i = c_{\mathsf{b}}$	$\mathsf{in\text{-}sync} = 0$
$\mathsf{stE} = \mathsf{stE}_{\mathsf{b}}$	**if** $\mathsf{in\text{-}sync} = 0$
return c_{b}	**return** msg
	return $\perp$

Note the asymmetric nature of the ADec oracle. Its output is different from $\perp$ if and only if $\mathsf{b} = 1$ and the adversary queries a ciphertext c that somehow passes the integrity checks inherent in Dec.

The security game is as follows:

$\underline{\mathsf{Exp}^{\mathsf{stLHAE}}_{\mathsf{AE}}(\mathcal{A})}$:
$k \xleftarrow{\$} \mathcal{K}$
$(\mathsf{stE}, \mathsf{stD}) \leftarrow \mathsf{Init}()$
$\mathcal{L}_{\mathsf{cip}} \leftarrow \emptyset$
$i, j \leftarrow 0$
$\mathsf{in\text{-}sync} \leftarrow 1$
$\mathsf{b} \xleftarrow{\$} \{0, 1\}$
$\mathsf{d} \leftarrow \mathcal{A}^{\mathsf{AEnc}(\cdot,\cdot,\cdot,\cdot), \mathsf{ADec}(\cdot,\cdot)}$

$\mathcal{A}$ wins if $\mathsf{b} = \mathsf{d}$.

Definition 3. *An authenticated encryption scheme* $\mathtt{AE}$ *is a stateful length hiding* $(t,\, q,\, \epsilon_{\mathsf{stLHAE}})$-secure *authenticated encryption scheme if, for all adversaries* $\mathcal{A}$ *running in time* t *and making at most* q *queries:*

$$|\mathbb{P}[\mathcal{A} \text{ wins } \mathsf{Exp}_{\mathtt{AE}}^{\mathsf{stLHAE}}(\mathcal{A})] - \frac{1}{2}| \leq \epsilon_{\mathsf{stLHAE}}.$$