A Unilateral-to-Mutual Authentication Compiler for Key Exchange (with Applications to Client Authentication in TLS 1.3)

Hugo Krawczyk*

July 18, 2016

Abstract

We study the question of how to build "compilers" that transform a unilaterally authenticated (UA) key-exchange protocol into a mutually-authenticated (MA) one. We present a simple and efficient compiler and characterize the UA protocols that the compiler upgrades to the MA model, showing this to include a large and important class of UA protocols. The question, while natural, has not been studied widely. Our work is motivated in part by the ongoing work on the design of TLS 1.3, specifically the design of the client authentication mechanisms including the challenging case of post-handshake authentication. Our approach supports the analysis of these mechanisms in a general and modular way, in particular aided by the notion of "functional security" that we introduce as a generalization of key exchange models and which may be of independent interest.

1 Introduction

A natural question in the area of key-exchange (KE) protocols is how to transform a given KE protocol that is secure in the unilateral-authentication (UA) setting (where only one of the parties authenticates and the other remains anonymous) into a secure mutual-authentication (MA) protocol. Somewhat surprisingly this question has not received much attention in the complexity-based KE literature, in part since much of the work in this area has focused on the mutual authentication case. In practice, however, the most widely used KE protocol, TLS, is centered around unilateral (server-only) authentication, with optional upgrade to mutual authentication. Namely, one starts from a core protocol where only the server authenticates and then extends it with an optional mechanism for client authentication.

In this work we investigate the above question in the context of complexity-theoretic models of KE, striving for results that are general as well as practically relevant to real-world protocols, in particular to the ongoing work on the design of TLS 1.3 [33] which provides particular motivation for this work. We think of this problem as developing *compilers* that extend secure unilateral key-exchange protocols into secure mutual key-exchange protocols. We focus on signature-based compilers, namely the case in which the second party to authenticate - which we refer to as the

*IBM Research. hugo@ee.technion.ac.il

client - uses a digital signature as the means for public-key for authentication (even though our approach can potentially be extended to other forms of authentication).

The SIGMAC Compiler. Our compiler is simple: To upgrade a unilateral protocol Π_1 into a mutually authenticated Π_2, upon completion of Π_1 the client sends a single message comprised of: (i) the client's signature on a portion of the Π_1's transcript; and (ii) a MAC value computed on the client and server's identities with a MAC key computed by Π_1. The intuition of the design is simple too: First note that anyone can sign the transcript, hence just a signature by the client on the transcript is not sufficient. Just MACing the identities is clearly not sufficient either since any party that participated in the unilateral protocol can compute the MAC on any identities. Yet the combination of signature and MAC ensures a binding between an identity and the knowledge of a key (computed in the unilateral protocol). We call the compiler SIGMAC for SIGnature and MAc Compiler, and also since it is reminiscent of the SIGMA protocol [24].

While intuitively appealing, proving the compiler, namely, showing that it can upgrade a secure unilateral protocol into a secure mutual one turns out to be non-trivial, in particular regarding what information needs to be covered by the signature. One can show examples, even in natural and practical settings, where even signing the whole transcript is not enough to ensure client authentication. Fortunately, we show that the compiler works for important classes of protocols, including any protocol that derives its session key from a Diffie-Hellman exchange. A core issue is to characterize what needs to be included under the signature. For this, we find a general sufficient condition (through a notion we call *transport replication security,* abbreviated as "treplication security") that if satisfied by a unilateral KE protocol Π, then applying the SIGMAC compiler to Π results in a provably secure mutually authenticated KE protocol. We then show extensive classes of unilateral protocols to possess this property, including KE mechanisms used in TLS 1.3.

By characterizing the *minimal* amount of information to be included under the client's signature, we achieve more general results that not only apply to a wider class of protocols but also allow to remove from the signature any information that the client may not want to sign with a non-repudiable signature. In particular, signing the server's identity leaves a transferable proof that the client communicated with the specific server. While full deniability of communication is hard to achieve against a malicious server [32], an explicit signature on the server's identity fails deniability even with a passive (or "honest-but-curious") server. We show that signing the server's identity can be safely omitted. On the other hand, we stress that if the protocol contains specific information that needs explicit authentication, e.g., the negotiation of security properties, these can and must be included under the signature even if these elements are not essential for the correct functioning of SIGMAC.

Finally, we comment on the use of a MAC function in the SIGMAC compiler. The essential observation (see Section 3.1) is that even if the client's signature covers both server's and client's identities, omitting the MAC - in particular, the MACed client identity - voids the ability of the compiler to produce a mutually authenticated protocol. As we note below, the use of a MAC function fits particularly well with TLS and its Finished message.

TLS 1.3: Application and Motivation. While the main results in this paper are of general applicability and intended to generically lift unilateral authentication protocols into mutual authentication ones, one timely motivation and application of this work is the analysis of TLS 1.3 [33]. The TLS key exchange (or *handshake*) protocol in the current and prior versions is built as a server-only authentication protocol with optional extension to client authentication. This approach calls exactly for a unilateral-to-mutual compiler as studied here. In particular, the specific SIG-MAC compiler captures the type of mechanism used in TLS which is based on the combination of a signature and a MAC (the latter referred to as the client's Finished message in TLS).

A challenging aspect of the new TLS 1.3 design is that the protocol allows for *post-handshake* client authentication, namely, a setting where a session key, authenticated by the server only, is used to protect TLS data (record layer) and only *later* is the client authentication extension applied. This raises the question of what's the value of this late authentication; can the exchanged data be considered as mutually authenticated upon this late client authentication? Fortunately, our compiler approach turns out to be particularly useful for analyzing this setting. It allows us to show that upon client authentication and successful verification by the server, the data exchanged and protected up to this point (as well as subsequent data) enjoys mutual authentication security in the sense of "secure channels". That is, it is guaranteed that only the parties that passed authentication (the named server and client) are those generating the information and being able to decrypt it. Of course, the server obtains this assurance for past data only *a-posteriori*, i.e., only after verifying client authentication. Other questions our methodology helps addressing regard the effect of late authentication on keys derived prior to the authentication (e.g., a "resumption key") or the validity of using a session key to encrypt the very messages that carry client authentication (see Section 6).[1]

In all, our results are relevant to the analysis of client authentication for the PFS modes of TLS 1.3, namely, 1-RTT and PSK, and for PSK without PFS (hence, also for resumption mode). This includes the cases of post-handshake authentication and the use of the session key (or application key) for encrypting the client authentication message.

An important tool for analyzing the above complex questions is our "functional" generalization of the key exchange model from [9] which may be of independent interest. It provides an abstraction that frees the analysis from specific model and functionality details while at the same time adding generality. For example, one can use this abstraction to reason about whether a session key used to protect key exchange messages may still be secure for another purpose (say, to derive other keys) or whether a late client authentication implies authentication of keys derived prior to the authentication (e.g., a resumption key in TLS 1.3).

Final note: Familiarity with TLS 1.3 is not essential for understanding the results in this paper that, while applicable and motivated by the TLS 1.3 design, are more general and emphasize the conceptual aspects rather than the details of a specific protocol. Even in the context of the ongoing specification of TLS 1.3, generality is a benefit for informing basic aspects of the design (such as the complex issues related to late client authentication). At the same time, we stress that a full analysis of TLS 1.3 is well beyond the scope of this work.

Related Work. The literature on key-exchange protocols is extensive with many different models and many analyzed protocols. Yet, the number of papers dealing with unilateral authentication is relatively small. In particular, the basic earlier models, such as those of Bellare-Rogaway [1] and Canetti-Krawczyk (CK) [9], do not treat it. One exception is the work by Shoup [34] that deals with

[1] An immediate conclusion from these results is that it is not necessary to introduce a dedicated key for encrypting post-handshake messages, a design choice being contemplated by the TLS working group. Avoiding such dedicated key has the major advantage of dispensing with cumbersome mechanisms involving trial decryption and the like.

"anonymous protocols" explicitly. Halevi and Krawczyk [17] treat unilateral security in the context of password protocols. Our treatment is based on the CK model which we specialize to the unilateral case with minimal changes following the approach of [26, 25]. More recently, and motivated by the increased interest in analyzing TLS, several papers study unilateral authentication. Particularly, the works of Maurer et. al [29] and Goldberg et al. [16] center on this model but do not deal with the question of building generic transformations from unilateral to mutual authentication.

Several works have presented authentication compilers for *unauthenticated* key-exchange protocols including [2, 9] and [19]. Authenticators from [2, 9] can be applied to upgrade unilateral authentication (UA) to mutual authentication (MA) but since authenticators are applied separately to each UA protocol message they result in added (and possibly interleaved) interaction, hence necessitating of changes in the UA part of the protocol (rather than acting as an extension). The compilers from [19] are intended to lift unauthenticated protocols to mutually authenticated ones, but a closer look reveals that they do not achieve this goal except if identities are somehow included under their signatures and MACs. Even if fixed, applying such compilers to a UA protocol is "overkill" (in terms of communication and complexity, e.g., the compilers from [19] add 3-6 flows to the protocol) and applying "half" compiler (in one direction only) does not necessarily work. For example, the SIGMA protocol from [24] shows that if *each party* signs the transcript (but not the identities) and also applies a MAC on its own identity then the protocol is secure. However, if we used the same technique for one party only to upgrade a UA protocol to MA we would end with an insecure protocol (see Section 3.1). The recent work of Kohlweiss et al. [23] analyzes a UA-to-MA transformation based on client passwords.

Analyses of TLS that address unilateral authentication include [30, 26, 21, 5, 22, 6, 3] and work directly relevant to TLS 1.3 [28, 14, 12, 23, 25, 13, 11]. Of particular relevance to the present paper is the work of Cremers et al. [11] who present a detailed formal analysis of TLS 1.3 (draft 10) using the Tamarin prover. It includes a formulation of delayed client authentication whose analysis uncovered a surprising attack on an early TLS 1.3 version of this mechanism which we use as a motivating example in Section 3.1. Additional work related to post-authentication in different protocol settings is the design of "channel binding" protocols whose track record of repeated breaks illustrates the misleading intuition behind many of these techniques - see Bhargavan et al. [7, 4]. It will be interesting to apply the treplication-security notion as a formal basis for studying channel binding and related notions. Finally, we point out that our "functional" approach from Section 6.1 bears resemblance to Shoup's model [34] and to the "suitable for" notion from [8].

2 Unilateral and mutually authenticated key-exchange models

This section presents an abridged description of the Canetti-Krawczyk security model for key-exchange protocols [9, 10] which serves as the basis for the formal security treatment in this paper. Please consult [9] for complete details.

2.1 Mutual Authentication (MA) Model

We start by presenting the original CK model for mutual authenticated KE, and then specialize this model to the case of unilateral authentication. For the purpose of this paper we will refer to this model as MA (for mutual authentication) and differentiate it from the unilateral authentication model denoted UA.

A $\mathsf{key\text{-}exchange\ (KE)\ protocol}$ is run in a network of interconnected parties (all of which, including adversaries, are modeled as probabilistic polynomial-time machines) where each party can be activated to run an instance of the protocol called a $\mathsf{session}$. Within a session a party can be activated to initiate the session or to respond to an incoming message. As a result of these activations, and according to the specification of the protocol, the party creates and maintains a $\mathsf{session\ state}$, generates outgoing messages, and eventually $\mathsf{completes}$ the session by outputting a $\mathsf{session\ key\ sk}$ and erasing the session state. A session may not complete (e.g., if authentication fails) in which case it is $\mathsf{aborted}$ without generating a session key (output $\mathsf{sk} = \perp$). A KE session is associated with its owner (the party at which the session exists), a peer (the party with which the session key is intended to be established), and a $\mathsf{session\ identifier}$, denoted sid. A KE protocol defines the contents of sid, typically including nonces and other transcript information. We name sessions by their owner and sid, and if the session peer is identified then also by the peer identity. For example, (S, sid) for a session at party S or (S, C, sid) if the peer to the session is identified as C. It is assumed (and enforced by the protocol) that any two sessions at the same (honest) party have different identifiers.

Credentials. We consider the public key setting where parties possess public keys of their choice and parties can verify the authentic binding between identities and public keys, be it either a certification authority (CA) or other means such as out-of-band distribution. We also consider the “pre-shared key setting” where pairs of parties share a secret symmetric key that they use for authentication.

Attacker model. The attacker, denoted $\mathcal{A}$, is an active “man-in-the-middle” adversary with full control of the communication links between parties. $\mathcal{A}$ can intercept and modify messages sent over these links, it can delay or prevent their delivery, inject its own messages, interleave messages from different sessions, etc. (Formally, it is $\mathcal{A}$ to whom parties hand their outgoing messages for delivery.) $\mathcal{A}$ also schedules all session activations and session-message delivery. In addition, the attacker is allowed access to secret information via the following attacks:

- A Reveal query can be performed against an individual session after completion and the result is that the attacker learns the corresponding session key sk.
- A $\mathsf{Corrupt}$ against a party means that the attacker learns the long-term secret of that party; in addition, from the moment a party is corrupted all its actions may be controlled by the attacker. Non-corrupted parties are referred to as uncorrupted or honest.

For simplicity, in the current presentation we do not consider the CK $\mathsf{StateReveal}$ query.

Basic security (mutual authentication). To define security, we establish a notion of $\mathsf{matching}$ defined via session identifiers. If session (C, S, sid) completes at party C, then session (S, C, sid), if it exists at S and completes, is said to be matching to (C, S, sid) (note the correspondence of peers and equality of sid and that only complete session have matching sessions).

Sessions against which any one of the attacks Reveal or $\mathsf{Corrupt}$ is performed (including sessions compromised via party corruption) are called $\mathsf{exposed}$. A session is called fresh if it is complete and not exposed, and its matching session (if it exists) is also not exposed.

The security of session keys generated in fresh sessions is captured via the inability of the attacker $\mathcal{A}$ to distinguish the session key of a test session, chosen by $\mathcal{A}$ among all *fresh* sessions in the protocol, from a random value. This is captured via a Test query. This query may only be asked once during the security game. It sets $K_0 := \mathsf{sk}$ (or random if $\mathsf{sk} = \perp$) as the real session key, and sets $K_1 \leftarrow_{\mathrm{R}} \{0,1\}^\lambda$. Then, it picks $b \leftarrow_{\mathrm{R}} \{0,1\}$ and returns K_b. The attacker can continue with the regular actions against the protocol also after the Test query; at the end of its run $\mathcal{A}$ outputs a bit b', which is meant as a guess for the value of b.

A key-exchange protocol π is secure if for polynomial-time attackers $\mathcal{A}$ running against π it holds:

1. If two uncorrupted parties complete matching sessions in a run of protocol π under attacker $\mathcal{A}$ then they output the same key (except for a negligible probability).
2. The probability that $\mathcal{A}$ wins the Test experiment, namely, it outputs a correct guess $b = b'$ is at most $1/2$ plus a negligible fraction.[2]

We note that this model is general enough to capture protocols with implicit or explicit authentication and with and without key confirmation (or liveness). This is important since it makes our results more general, showing that our compiler applies to a UA protocol with any of the above characteristics.

A key-exchange protocol is said to achieveperfect forward secrecy (PFS) if it satisfies the above definition when relaxing the notion of fresh sessions to allow a Corrupt query against the owner of the session but only upon completion of the session.

2.2 Unilateral Authentication (UA) KE Model

The model as described above is intended to capture protocols where both peers authenticate to each other. Here we specialize this model to the case of unilateral authentication (UA), namely, when only one party authenticates to its peer (but the second party remains "anonymous"). The treatment is similar to [25].

The unilateral setting is best described by client-server terminology. We denote by C, S the client-server parties to the protocol as well as their identities, and refer to the authenticating party as the server. In the public key setting servers have public keys and in the case of a server and client sharing a symmetric key (the pre-shared key setting) we assume that the client associates the key to a server identity but not (necessarily) the other way. Sessions at a client are denoted as triples (C, S, sid) as in the regular MA model but sessions at servers do not have a named peer hence they are denoted (S, sid).

The essential characteristic of the UA model is that *only (fresh) client sessions are allowed as test sessions.*[3] Thus, matching is defined only for client sessions, namely, if session (C, S, sid) exists and completes, and session (S, sid) exists and completes then (S, sid) is called a matching session to (C, S, sid).

[2]We use asymptotic, polynomial-time, terminology throughout the paper as it simplifies presentation. Moving to a concrete security setting is mostly straightforward.

[3]An equivalent definition would allow also to test server sessions but only if they have a matching fresh client session.

SIGMAC applied to UA protocol Π_1 uses the following components:

- A key derivation function KDF applied to session keys output by Π_1 for producing two keys K_a and K_s (this can be as simple as computing $K_a = f_K(0)$ and $K_s = f_K(1)$ where f is a PRF, or any other derivation of mutually pseudorandom values K_a, K_s).
- A signature algorithm and a MAC algorithm (with the usual chosen message security requirements for both).
- A value sid^* defined for each session in Π_1 and consisting of the concatenation of the session id sid and a subset of transcript information, called a transcript core, specifically defined for each protocol Π_1 (see Section 4).
- Identities and keys for clients: Clients are anonymous in Π_1, but in Π_2 they have identities and signature keys whose public keys other parties can obtain and validate. Servers have same identities and keys as in Π_1.

Protocol Π_2 is defined identically to Π_1 except for the following extension:

- When a client C completes a session (C, S, sid) in Π_1 with peer S and outputs a session key K, it performs the following operations in Π_2. It derives keys K_a and K_s from K using KDF; sends a message, denoted CSM, consisting of a signature on the value sid^* and a MAC value, $\mathsf{MAC}_{K_a}(C, S)$, on the identities C and S; it erases K_a and completes session (C, S, sid) in Π_2 with session key K_s.
- When a server S would establish a session (S, sid) with session key K according to Π_1, in Π_2 it does not complete the session but computes K and keeps it in the session's state. When a CSM message (allegedly) from client C is delivered to a session (S, sid), S checks that a key K was previously computed for this session and if so it derives keys K_a and K_s form K using KDF. It then checks that the incoming signature is valid (under the public key of C) and that it covers the appropriate value sid^* (including the session id sid). Finally, it verifies that the MAC computed with key K_a covers the client identity C and its own identity S. If all checks succeed, S completes session (S, C, sid) with session key K_s.

Figure 1: The SIGMAC Compiler.

3 The SIGMAC Compiler

We present our signature-based compiler that we call SIGMAC and which augments a UA-secure protocol Π_1 into a MA-secure protocol Π_2. Roughly, protocol Π_2 is obtained via the SIGMAC compiler by adding to Π_1 a single message sent from client C to server S upon completion of a session in Π_1, where the message, denoted CSM (for "client sign-and-mac"), comprises a signature of C and a MAC (with a key derived from Π_1's session key). The signature is applied to a value sid^* defined for Π_1 while the MAC is applied on the identities of C and S. The compiler is presented in Figure 1.

We stress that the value sid^* plays an essential role in the security of SIGMAC and its requirements are presented in detail in Section 4 via the notions of a transcript core and treplication security. We simplify some of our treatment by assuming that the session identifiers sid in protocol Π_1 include unique random nonces contributed by C and S.

Another simplification in our treatment is that we assume that the signature keys used by clients for the SIGMAC signature are not used for other purposes or as part of protocol Π_1. This limitation is not essential as long as different uses are properly insulated via domain separation or similar methods.

More generally, we require that no two sessions at a party will have the same sid (except for negligible probability); also, we assume that sid^* includes a fresh session-specific value from S (needed, in particular, against replay attacks).

Note on preserving the session key from Π_1*.* The above specification of SIGMAC defines the keys K_a, K_s used by Π_2 as derived from the session key K output by Π_1. Alternatively, one can specify that Π_1 outputs K_a, K_s, where K_s serves as the session key in both Π_1 and Π_2 while K_a is only used in Π_2 as a MAC key. We chose the former approach as it allows us to keep Π_1 unmodified and is somewhat more convenient in the formal treatment. On the other hand, if one requires that Π_1 and Π_2 use the exact same session key, one needs to accommodate the generation of key K_a already as part of Π_1 (this would be the case in TLS 1.3).

Note on (not) signing the server's identity. If one wants to provide some deniability against a passive adversary as discussed in the introduction, one should avoid entering the identity of S into the signature (i.e., into sid^*).

On (hashing) the identities included under SIGMAC's MAC. The identities C and S included under the MAC computation in SIGMAC can have different forms, e.g., a subject name in a public key certificate, a whole certificate, an email address, etc. However, we note that an identity of a party X can be replaced, for the sake of including it under the SIGMAC MAC, by $H(X)$ if H is a collision resistant function (this is similar to the use of hash functions for generating inputs to digital signatures). Moreover, hashing X with additional information is also acceptable as long as the parsing of the input to the hash function is defined as to uniquely determine the identity included in the hash. Such hashing of identities is used in many protocols including TLS. As a subtle example, resumption mode of TLS 1.3, replaces the server's identity included in the client's Finished message (that implements the SIGMAC's MAC functionality) with a unique value, `resumption_context`, computed as a hash of the server's identity and carried from a prior exchange between this client and server. For this to be secure one has to assume that the function used to compute `resumption_context` is collision resistant. In the case of TLS 1.3 this function is based on HKDF-Expand which can be shown to be collision resistant if the underlying hash is collision resistant. We stress that it is imperative in all uses of hashed identities that the output from the hash function not be truncated in a way that weakens the collision resistance property.

3.1 Rationale and examples

As explained in the introduction, the intuition behind SIGMAC is that the matching of the same client identity as the signer as well as under the MAC creates the needed binding between the client and the session key. Adding the server's identity to the MAC is needed to prevent UKS attacks (see below), namely, ensuring the server that the client's view of the peer is correct. Alternatively, one can include the server's identity under the signature which raises some privacy issues as discussed earlier.

The actual considerations in analyzing the compiler are more subtle particularly with respect to what needs to be covered by the signature. In some cases, even covering the whole transcript is insufficient to ensure mutual authentication via SIGMAC while in other cases covering part of the transcript is enough. We strive to identify the essential parts of a transcript that need to be signed (this adds to the generality of the compiler and to our understanding of this mechanism, and can help achieving some level of deniability).

We illustrate some of the subtleties through some important examples serving as motivation for the notion of *transcript replication attacks* introduced in the next section and central to our analysis (and with direct implications to the security of client authentication in TLS 1.3).

MACing the server identity S. We show a UA protocol that when upgraded to MA via SIGMAC but without including the server's identity under the client's MAC, results in a MA protocol that is open to a UKS attack (the necessity of including C under the MAC is more obvious since anyone can sign the transcript hence such a signature has no binding to the UA run). The UA protocol is a simple server-authenticated DH exchange where the client sends a DH value g^x to which the server responds with g^y and a signature on g^x and g^y. The session key K is derived from g^{xy}. If we extend this to a MA protocol via SIGMAC but removing S from the MAC, namely, C sends $MAC_{K_a}(C)$ and its signature on both g^x and g^y (whose concatenation defines sid), the resultant protocol is open to a UKS attack as follows. A malicious server S' relays g^y to C but signs it with its own signature. When C responds with $MAC_{K_a}(C)$, S' sends this message to S. In this case K_s is computed by both C and S. But while S has C as the peer to the exchange, C has S' as the peer, hence resulting in a UKS attack.

Pre-shared key protocols and why sometimes even signing the whole transcript is not enough. We show an example of a UA protocol where applying the SIGMAC compiler with the *entire transcript* under the client's signature does not result in a MA-secure protocol. For this we consider a UA-secure protocol Π_1 where the client uses as the server's credentials the server's identity and a symmetric pre-shared key (PSK). We assume that this key, that might have been previously shared by C with S through another UA key exchange protocol, has been authenticated only by S (as in the resumption mode of TLS 1.3). We define Π_1 as an implicitly-authenticated protocol where the parties just exchange nonces (which form the whole transcript) and the session key is derived by applying a PRF keyed with PSK to the nonces. It is not hard to show that Π_1 with sid set to the concatenation of the nonces is UA-secure. Now, consider the protocol Π_2 resulting from applying SIGMAC to Π_1 where we set the transcript core included in sid^* to the entire transcript. Let S' be an attacker that has a shared key with C and one with S (the two keys may be random and independent) and interacts as a MitM between C and S, acting as the server with C and as a client with S. S' simply relays without modification the nonces chosen by C and S. In addition, S' forwards C's signature on sid^* to S but replaces the MAC generated by C with a MAC computed by S' on identities (C, S) using a MAC key derived from the PSK that S' shares with S. As a result S ends its session with peer C but with a session key that is known to S'! Note that if we add explicit authentication to Π_1 by sending a MAC from the server to client (using a key derived from the PSK) but this MAC is not included in sid^* then the above attack works as well. Moreover, as we will see in Section 4.2, in some cases even including the server's MAC under the signature is still not enough to provide MA security (in particular, this indicates that the source of problem was not in the implicit authentication nature of Π_1).

Interestingly, this attack scenario was discovered by the Tamarin team [11] in their formal analysis of an early draft of TLS 1.3, showing that this is not just an hypothetical concern but something that can arise in practice.

SIGMAC security for DH-based protocols (and some more subtleties). An important and extensive class of protocols where SIGMAC can be applied to lift a protocol from UA to MA are protocols that provide forward secrecy by deriving the session key from a DH value g^{xy}, where g^x, g^y are exchanged between C and S and g^y is authenticated by S. We show in Section 4.1 that defining sid^* to include both values g^x, g^y is sufficient for SIGMAC to ensure MA security.

As yet another interesting subtlety regarding SIGMAC, we can show that without including g^y under the client's signature, MA-security is not guaranteed (this is somewhat counter-intuitive as one could expect that signing the client's message, with some freshness information from the server, would suffice). For this we consider a simple UA protocol, namely, a DH exchange between C and S where the server signs the whole transcript. If g^y is not included under the client's signature, an attacker S' can change g^y to g^{2y} leading C to compute its MAC using a key derived from g^{2xy} while S derives the verification key from g^{xy}. Using ideas similar to the example used in [26] to show the necessity of the PRF-ODH assumption in the analysis of TLS 1.2, one can build a KDF and MAC so that the MAC computed on (chosen) values C and S' with a key derived from g^{2xy} results in the same MAC tag as when computed on values C and S with key derived from g^{xy}. Also, the session key computed through the KDF is the same when derived from g^{xy} and g^{2xy}. This results in a UKS attack where both C and S compute the same session key but while S binds it to peer C, C binds it to peer S' (which signed g^{2y}). While clearly unnatural, this attack shows that if one does not include g^y in sid^*, the SIGMAC compiler cannot be proved to work in this case (at least without assuming additional properties of the KDF and MAC functions).

Implicit authentication and 0-RTT. We stress that SIGMAC applies to UA protocols in the CK model which include both implicit and explicit authentication (and with or without key confirmation)[4] and we have seen such examples above. An interesting application case is the 0-RTT mode of TLS 1.3, a one-pass protocol with implicit server authentication. In Section 4.3, we discuss the applicability SIGMAC to this case.

4 Transcript-Replication (Treplication) Security

As shown in Section 3.1, the information that needs to be signed by the client in the SIGMAC compiler, namely sid^*, plays an essential role in the security of the compiler. The value sid^* is comprised of the session id sid and a subset of transcript elements as defined next via the notion of treplication security.

Definition 1 (Treplication security). *We say that a UA protocol* Π *is secure against* transcript replication (treplication) attacks *with respect to a transcript subset* τ *(called a* transcript core*)*[5] *if any efficient UA-attacker* S' *against* Π *has only negligible probability to win the* Test *experiment on a fresh session* (S, sid) *for which there is a session* (C, S', sid) *with same session identifier* sid *and same transcript core* τ *as* (S, sid)*, and where* C *is a honest client.*

[4] With SIGMAC one gets explicit authentication and key confirmation on the client side.

[5] That is, we view the protocol transcript as an ordered set of (possibly optional) elements and define the transcript core as a specified subset or projection.

Note that while S' is given the same capabilities of a UA attacker and the same indistinguishability Test, the conditions for the choice of the test session are very different than in the regular UA model (in particular, this is a server's session, not a client's one). Essentially, the requirement is that if S' acts as a man-in-the-middle between honest parties C and S but cannot change or choose the session id sid and transcript core τ sent between C and S, then S' cannot learn information on the session key computed by S. This requirement is incomparable to UA security, namely, it is not implied by nor it implies UA security.

The importance of the treplication security notion is demonstrated by Theorem 6 where we show that joint UA and treplication security suffices for the successful use of the SIGMAC compiler. Here we prove some important classes of UA protocols to be treplication-secure (with respect to specific defined transcript cores), hence upgradeable to mutual authentication via SIGMAC. These protocols include those discussed in Section 3.1 and those with direct application to TLS 1.3 client authentication (particularly to the PFS modes of the protocol as well as the non-PFS PSK mode which also functions as session resumption).

4.1 Treplication security of Diffie-Hellman Protocols

In Section 3.1 we showed some of the subtleties arising in proving treplication security of DH protocols. Here we show how such security can be achieved. We consider three cases: (i) plain DH where g^{xy} is only used to derive the session key; (ii) in addition to deriving the session key from g^{xy}, additional keys used during the key exchange protocol itself are derived from g^{xy} - however, we assume that the session key is secure (indistinguishable from uniform) even if these additional keys are provided to the attacker; (iii) the exponent x (and/or y) is used for additional computation (not just for computing g^{xy}). We note that case (i), while being the simplest and more natural, is generally insufficient for addressing real-world protocols. Case (ii) is directly relevant to TLS 1.3 in its 1-RTT mode and in the pre-shared key mode with PFS (the additional g^{xy}-derived keys are used for handshake traffic protection). Case (iii) would show up in a protocol like QUIC [27], or a similar extension to TLS 1.3, where two DH keys involving g^x are derived: an ephemeral g^{xy} as well as a key g^{xs} that combines g^x with a public key g^s of the server.

Lemma 2. *Let Π be a UA-secure Diffie-Hellman protocol of the above types over a group G. Then, under the DDH assumption on G, Π is treplication-secure if one defines the transcript core τ to include the session identifier sid of Π as well as the two DH values g^x and g^y.*

Proof (sketch). Case (i): We sketch the proof of treplication security by reduction from DDH. Let $T = (g^x, g^y, g^z)$ be a triple where g is a generator of G of order q and either $z = xy$ or $z \leftarrow_{\mathrm{R}} Z_q$. Let S' be a treplication attacker against protocol Π with sid^* defined to include the DH values exchanged by the parties. We build a DDH distinguisher SIM that simulates a run of S' against Π. SIM chooses a honest server's session at random as its guess for the test session (S, sid) to be chosen by S'. By the rules of the replication game, the test session will have an incoming message g^x sent by a honest client C and which S' is not allowed to change. SIM also guesses which client and session will send such a message and it uses g^x from the triple T as this message. Similarly, S' cannot change the response g^y coming out of S, so SIM uses g^y from T for this purpose. If SIM has all these guesses correct then the simulation of S' is same as in the real protocol. SIM then uses g^z from the triple T to answer the test query of S'. Clearly, if $z = xy$ the response corresponds to the real session key while if z is random so is SIM's response. In all, we get that the advantage of S' in the treplication game is (up to a polynomial factor introduced by the guessing of test session) the same as the distinguishing advantage in the DDH game, hence negligible.

Case (ii): Here, in addition to using g^{xy} to derive the session key, g^{xy} is used to derive other keys used during the execution of protocol Π. We assume that the output from the key derivation function KDF on either g^{xy} or a random group element is pseudorandom. The proof is similar to the above but we define a hybrid game where keys that are used during Π's execution and which include g^{xy} in their derivation are replaced with random values. The hybrid game argument is standard and omitted.

Case (iii): We consider a specific setting (e.g., QUIC) where parties compute two values from which keys are derived: g^{xy} and g^{xs}, where g^s is the server's static public key. Other cases can be treated similarly. For this case we use a somewhat different but equivalent formulation of the DDH assumption where a tuple $(g^x, g^y, g^s, g^{xs}, g^z)$ with random x, y, s is given and the task is to distinguish between the case where $z = xy$ or z is random. Clearly, the addition of g^s, g^{xs} does not change the hardness of DDH as these two values can be "simulated" by choosing s and adding to a regular DDH triple the values g^s and $(g^x)^s$. The only change to the proof of case (i) is that we now include $g^{xs'}$ in the input DDH tuple where $g^{s'}$ is the public key used by S' in its attack. Given $g^{xs'}$, the simulation can proceed as in the above lemma (one needs this value as input so that SIM can follow the actions of C which depend on $g^{xs'}$). $\square$

4.2 Treplication security of Pre-Shared Key Protocols

Here we consider *pre-shared key (PSK) key-exchange protocols*, namely, protocols where the parties use as their credentials a symmetric key they previously shared by some method (e.g., out-of-band or a previous key exchange execution). We are interested in protocols that are UA-secure, hence *we assume the shared key to have been authenticated only by the server* (TLS 1.3 resumption presents such a case).

PSK protocols that offer forward secrecy (i.e., derive their session key through a DH exchange) are treplication-secure under the conditions demonstrated earlier. Thus, here we focus on UA PSK protocols without forward secrecy and investigate under which conditions they are treplication-secure. We've seen in Section 3.1 that the basic implicitly-authenticated PSK protocol where the parties exchange nonces and derive a session key from their PSK and nonces (without further authentication) *cannot* be made treplication-secure even if the full transcript (the nonces in this case) is included in sid^*. Thus we consider an extension of this protocol that is UA-secure with explicit authentication where S sends a MAC computed on the nonces with a MAC key derived from the PSK (this MAC would correspond to the server's Finished message in TLS). For the moment we restrict the transcript to only include the random nonces of C and S, denoted n_C, n_S, and the server's MAC. We refer to this protocol as Basic-UA-PSK and define the whole transcript as the transcript core τ.

Interestingly, this protocol *is not treplication-secure for generic MAC functions.* However, we can prove its security under the following assumptions on the MAC function.

Definition 3. *We say that a MAC function is* collision safe *if any efficient algorithm for choosing keys K_1, K_2 has only negligible probability to find $K_1 \neq K_2$ such that for random nonces n_C, n_S*

(chosen independently of the keys), it holds that $\mathsf{MAC}_{K_1}(n_C, n_S) = \mathsf{MAC}_{K_2}(n_C, n_S)$.

Note that this property does not follow from the definition of a MAC function. For example, there can be a secure MAC function where keys that differ only by the least significant bit produce the same results, hence making it trivial to choose colliding keys K_1, K_2. Yet for natural MAC functions the above property is expected to hold. We note that in practical settings the finding algorithm (in the treplication setting this algorithm corresponds to the attacker S') may be further limited in the choice of K_1, K_2, e.g., these keys may need to be set through the run of a key-exchange protocol with honest parties (this is the case of the resumption protocols in TLS 1.3). This weakens the requirement from the MAC function, yet, for simplicity and generality, we will use the above more stringent definition.

Lemma 4. *Protocol Basic-UA-PSK where the transcript core τ includes the whole transcript (i.e., the parties' nonces and the server's MAC) is treplication-secure provided the MAC function in use is collision safe.*[6]

Proof: According to the definition of treplication security we have a setting where an attacker S' runs concurrent executions of protocol Basic-UA-PSK with honest parties C and S, where S' acts as server with C and as a client with S. In the present case, S' has a PSK K_1 shared with C and a PSK K_2 shared with S where $K_1 \neq K_2$. Since the whole transcript is included under the signed sid^*, S' can only relay the messages between C and S. In particular, S' must send as its MAC value to C the same MAC value sent by S. However, while S computed the MAC with (a key derived from) K_1, C will check the MAC with (a key derived from) K_2. The only way S' can succeed is by finding keys K_1, K_2 that result in the same MAC value when applied on the nonces chosen by the honest parties C and S. By the collision safeness assumption of the MAC, there is only negligible probability of the two MAC values be equal hence only negligible probability for S' to win the treplication game. $\square$

We now consider more realistic extensions of Basic-UA-PSK where in addition to the nonces, the parties exchange other information (e.g., algorithm negotiation or security parameters) which may or may not be included under the server's MAC. We denote by m_C (resp. m_S) the transcript values (other than the nonces) sent by C (resp. S) that are included under the server's MAC.

Note that if the values m_C, m_S are not included under sid^* they may be chosen by S' in a way that can lead to a MAC collision (particularly if these values can be chosen after seeing the parties' nonces). Namely, S' can choose values m_C, m_S that lead to a collision $\mathsf{MAC}_{K_1}(n_C, n_S, m_C, m_S) = \mathsf{MAC}_{K_2}(n_C, n_S, m_C, m_S)$. In particular, this is the case if the MAC is short, say 128 bits or even 96 bits as in some protocols and m_C, m_S have enough entropy.[7] The solution to this attack avenue is to define that all data included under the server's MAC is also part of the transcript core, hence included under sid^* (so that S' cannot choose this data; we note that values that are part of m_C, m_S but that are of low entropy may be omitted from τ).

With the above precautions, Basic-UA-PSK remains treplication-secure even with additional information exchanged between the parties and included under the server's MAC.

[6]We exclude the case where the attacker has the same PSK key shared with both C and S in which case a transcript replication attack is unavoidable.

[7]We note that while short MACs have been the subject of attack for their *wrong use* in protocols - see [31, 7], there is nothing inherently insecure in short MACs (of length not less than the security parameter) for the purpose of message authentication.

Lemma 5. *Consider a UA-secure PSK-based protocol* Π *where parties exchange random nonces as well as other information and the server computes a MAC (with a key derived from PSK) on the nonces and possibly other information denoted* m_C, m_S*. Assume the MAC is collision safe when applied to random nonces and to these values* m_C, m_S*. Then protocol* Π *is treplication-secure if the transcript core includes all information under the MAC.*

4.3 One-pass (0-RTT) Protocols

TLS 1.3 supports a so called 0-RTT handshake consisting of a one-pass authenticated key-exchange protocol where authentication is carried through the use of a pre-shared key that client and server exchange in a previous regular handshake, typically with server-only authentication. In particular, this mode is used to instantiate session resumption in TLS 1.3. The 0-RTT handshake essentially consists of the client sending a key identifier (that allows the server to identify or compute the pre-shared key K) and a nonce. The session key (used to immediately encrypt client's data) is derived from a combination of the key K and the client's nonce. Note that there is no server's nonce assumed in the protocol which opens this mode to replay attacks. Here we discuss how client authentication could be added to such a mode, although TLS 1.3 does not currently support client authentication for 0-RTT.

We first note that one-pass protocols can be framed in the CK model through an adaptation as presented in [18], and the above nonce-based protocol can be proven to be UA-secure in this model. Thus, we can apply SIGMAC to this protocol MAC-ing the identities of C and S (in the case of TLS 1.3, the MAC would be implemented via the client's Finished message that is part of the 0-RTT exchange) and signing the client's nonce (which acts as sid). Note, however, that SIGMAC assumes a server's nonce which is not present here but this requirement can be relaxed if one accepts replay attacks in the model (as in [18]). More fundamentally, the above one-pass protocol is *not* treplication secure for reasons similar to (and more straightforward than) the interactive PSK case discussed in Section 3.1. Specifically, the attacker S' would establish separate pre-shared keys with both C and S (with S' acting as server and client, respectively) and later run a 0-RTT handshake with both parties, relaying the client-chosen nonce to S. Clearly, S' learns the key computed by S as it knows the pre-shared key used by S, hence breaking treplication security.[8] Unfortunately, and in contrast to the interactive PSK case treated earlier, this treplication insecurity cannot be fixed without including the server's identity (or a value derived from it, such as `resumption_context` discussed before) in the transcript core. Thus, client authentication for 0-RTT would need to include the server's identity (and nonce) under the client's signature. Fortunately, in the next section we show that when the server's identity is signed by C, SIGMAC ensures MA security even if the base UA protocol is not treplication secure.

Finally, we comment on a variant of 0-RTT, not currently supported by TLS 1.3, that replaces the pre-shared key with a server's public key g^s (stored by the client) and an ephemeral value g^x provided by the client (see [25]). This mode is not treplication secure if the attacker S' is allowed to choose a public key related to that of a honest S (by reasons similar to the DH case treated in Section 3.1, e.g., by S' setting its public key to g^{2s}). Yet, as in the above pre-shared key 0-RTT case, one can leverage SIGMAC to provide MA security but only if one includes the server's identity

[8]The attack is possible even if a nonce by S is involved and may even work if the key identifier is part of the transcript core as long as S' can choose the same key identifier as S did.

or a derived value such as `resumption_context` under the client's signature.

5 Proof of the SIGMAC Compiler

We prove the security of the SIGMAC compiler defined in Section 3 for treplication-secure protocols. We assume the MAC and signature functions used in SIGMAC to be secure in the standard chosen-message unforgeability sense.

Theorem 6. *Let Π_1 be a secure unilaterally authenticated KE protocol that is also treplication-secure with transcript core τ and set sid^* to the concatenation of the session id sid and τ. Then protocol Π_2 resulting from the application of the SIGMAC compiler with this sid^* is a secure mutually authenticated KE protocol.*

Before proving the theorem we recall some of the adversarial actions in each model and how they differ in the case of Π_1 and Π_2.

- Party activation for session initiation and message receiving/sending: This is same in Π_1 and Π_2 except for the CSM message sent by C upon receiving the last message from S (in case C sends the last message in Π_1 then CSM is sent immediately after this last message).
- Party corruption: In both cases this includes revealing to the attacker all secret material of the corrupted party (and subsequent full control of that party by the attacker). Note that in the case of Π_2, client corruption includes revealing the signing key of C.
- Reveal queries: This can be applied to any completed session (other than the test session) at a honest party and returns the session key (note that the timing of completion for S may be different in Π_1 and Π_2).
- Session tests: In Π_1 session tests are only allowed against (C, S, sid) while in Π_2 they are also allowed against (S, C, sid).

5.1 The simulator

For the proof of Theorem 6 we specify a simulator that given an MA-attacker $\mathcal{A}_2$ against protocol Π_2 it defines a UA-attacker $\mathcal{A}_1$ against Π_1 such that if Π_2 wins a test session in Π_2 so does $\mathcal{A}_1$ in Π_1. The simulator SIM acts as the challenger (i.e., the orchestrator of the protocol run) for a given MA-attacker $\mathcal{A}_2$ against protocol Π_2 and uses $\mathcal{A}_2$ actions to implement the UA-attacker $\mathcal{A}_1$ against Π_1. We think of the parties running Π_1 as real parties, with their own secret keys, and those running Π_2 as simulated by SIM with the same keys as in Π_1 except for clients' signature keys (that exist in Π_2 but not in Π_1). The latter are chosen by SIM for honest clients in Π_2 while public keys for corrupted parties are chosen by the attacker.

SIM implements the CSM message in Π_2 as follows. Upon completion of a session (C, S, sid) by a client C in Π_1, SIM invokes $\mathcal{A}_1$ to run a reveal query on that session to obtain the corresponding session key K. From it, SIM derives keys K_a and K_s, and uses K_a and the signature key of C to compute the CSM message that it hands to $\mathcal{A}_2$. Completion of sessions by honest parties in Π_2

is decided by SIM. In the case of a client, a session is completed if and only if the corresponding session completes in Π_1. Completion of a session (S, C, sid) at a honest server in Π_2 depends on receiving a valid CSM message, namely with a valid signature on sid^* and the MAC under K_a. If this message is not originated with a honest client (e.g., it is sent by a corrupted client) then SIM verifies it by invoking $\mathcal{A}_1$ to reveal the corresponding completed server's session (S, sid) in Π_1 and deriving K_a from it. The validity of this action follows from Lemma 7 (c).

The actions of $\mathcal{A}_1$ as decided by SIM mostly mimic those of $\mathcal{A}_2$ with some important differences in the handling of the corrupt, reveal and test queries that we specify next. After each action we argue informally the validity of the action (showing that the attackers' actions and views are consistent with the protocol and the attack model). The formal proof of validity follows from Lemma 7 below. Throughout, we say that keys K_a, K_s in a session in Π_2 are consistent with a session key K in Π_1 if K_a, K_s are derived from K using the KDF function.

1. When $\mathcal{A}_2$ corrupts a party, SIM invokes $\mathcal{A}_1$ to do the same. $\mathcal{A}_2$ receives the secret information from this party as received by $\mathcal{A}_1$ and, in the case of clients, it receives from SIM the party's signature key.

2. When $\mathcal{A}_2$ reveals a session (C, S, sid) at honest client C in Π_2, SIM invokes $\mathcal{A}_1$ to reveal (C, S, sid) in Π_1; $\mathcal{A}_1$ is handed the session key K from Π_1 while $\mathcal{A}_2$ is handed the derived session key K_s from Π_2.

   - Validity (Lemma 7 (b)): We use the fact that these two sessions exist and complete together and have consistent keys.

3. When $\mathcal{A}_2$ reveals a session (S, C, sid) at honest server S in Π_2, SIM invokes $\mathcal{A}_1$ to reveal session (S, sid) in Π_1; $\mathcal{A}_1$ is handed session key K while $\mathcal{A}_2$ is handed the derived key K_s.

   - Validity (Lemma 7 (c)): We use the fact that if S is honest and session (S, C, sid) completes in Π_2 then session (S, sid) exists and completed in Π_1, and the two sessions have consistent keys.

4. The processing of the Test query and test sessions is handled by SIM as follows. At the onset of the protocol run, SIM selects a random session, called the *guess session*, from all sessions to be created during the run of protocol Π_2 under $\mathcal{A}_2$ (this assumes an a-priori bound on the number of sessions to be created by $\mathcal{A}_2$). If at any point, $\mathcal{A}_2$ issues an action that disqualifies the guess session as a test session, SIM instructs $\mathcal{A}_1$ to output a random bit b and aborts. The guess session may be at a client or server:

   (a) If the session is at a client, call it (C, S, sid), then as soon as $\mathcal{A}_2$ delivers the last incoming message to this session (from server S), SIM invokes $\mathcal{A}_1$ to deliver this last message (which, if the guess session is correct, results in C completing) and to issue a Test query on (C, S, sid) in Π_1.

   (b) If the session is at a server, call it (S, C, sid), then as soon as $\mathcal{A}_2$ delivers the last message from S to session (C, S, sid), SIM invokes $\mathcal{A}_1$ to deliver this last message and to issue a Test query on (C, S, sid) in Π_1.

In both cases SIM learns from $\mathcal{A}_1$ the value of the real-or-random key K returned by the Test query in Π_1. SIM derives from K keys K_a and K_s and uses K_a to compute the outgoing CSM message from C in Π_2 and for checking the validity of a CSM message delivered by $\mathcal{A}_2$ to session (S, C, sid). When $\mathcal{A}_2$ issues a Test query against the guess session (at C or S), SIM provides K_s as the answer to that query.

- Validity: We will show that the test sessions invoked by $\mathcal{A}_1$ are valid test sessions under $\mathcal{A}_1$'s run and that K_s is real if K is real and (pseudo) random otherwise. See Lemma 7 (f).

5. When $\mathcal{A}_2$ stops its run with an output bit b, SIM instructs $\mathcal{A}_1$ to stop with the same bit.

   - We need to show that $\mathcal{A}_1$ wins whenever $\mathcal{A}_2$ wins, hence proving that the advantage of $\mathcal{A}_2$ against Π_2 is bounded (up to a polynomial factor induced by the probability of a correct test session guess) by the advantage of $\mathcal{A}_1$ against Π_1.

5.2 Validity of SIM's actions

The proof of Theorem 6 uses the following properties of the simulator's actions and those induced on attacker $\mathcal{A}_1$, in particular showing the validity of these actions in the corresponding models.

Lemma 7. *The following properties hold based on the simulation actions.*

(a) Π_1 *and* Π_2 *as run under* $\mathcal{A}_1$ *and* $\mathcal{A}_2$*, respectively, have the same set of parties, including the same honest and same corrupted parties.*

(b) If C *is honest, then when a session* (C, S, sid) *completes in* Π_2*, a session* (C, S, sid) *exists and has completed in* Π_1*; in particular, the two sessions have consistent session keys.*

(c) If S *is honest, if a session* (S, C, sid) *completes in* Π_2 *then session* (S, sid) *exists and has completed in* Π_1*; in particular, the two sessions have consistent session keys.*

(d) If (C, S, sid) *is a valid test session chosen by* $\mathcal{A}_2$ *in* Π_2 *then, except for negligible probability,* (C, S, sid) *is fresh in* Π_1 *and (by part (b)) the two sessions have consistent session keys.*

(e) Assuming Π_1 *is treplication-secure, if session* (S, C, sid) *is a valid test session chosen by* $\mathcal{A}_2$ *in* Π_2*, then, except for negligible probability, session* (C, S, sid) *exists, completes and is fresh in* Π_1 *and both sessions have consistent session keys.*

(f) If the test session chosen by $\mathcal{A}_2$ *in* Π_2 *is a valid test session and SIM guessed this session correctly then, except for negligible probability, the test session chosen by SIM is a valid test session under* $\mathcal{A}_1$*'s run, and if the key* K *returned to* $\mathcal{A}_1$ *is real (resp. random) then* K_s *is real (resp. pseudo-random).*

Proof: Claims (a)-(c) can be verified by simple inspection of the simulator actions.

Proof of part (d). By part (b), if (C, S, sid) completes in Π_2 then it also completes in Π_1. Thus, we need to show that the validity of (C, S, sid) as a test session in Π_2 implies freshness of (C, S, sid) in Π_1. Since (C, S, sid) is fresh in Π_2 then C and S are uncorrupted in Π_2 and by part (a) they are also uncorrupted in Π_1. In addition, since $\mathcal{A}_2$ does not reveal (C, S, sid) in Π_2, this session is not revealed in Π_1 as part of item 2 of the simulation. Nor is this session revealed in Π_1 for obtaining the key K_a needed for simulating the CSM message (in this case, SIM derives K_a from the real-or-random key K). We conclude that (C, S, sid) is not exposed in Π_1. It remains to show that the matching session to (C, S, sid) in Π_1, namely, (S, sid), was not revealed. In Lemma 8 we will prove that if both (S, sid) and (C, S, sid) completed in Π_1 and (C, S, sid) is unrevealed in Π_1, then in Π_2 either S did not complete session sid or, if it did, it completed it as (S, C, sid). Thus, since (C, S, sid) is complete and unrevealed in Π_1 (follows from (C, S, sid) being fresh in Π_2), we can apply Lemma 8 to show that if (S, sid) completed in Π_1 and in Π_2, then in Π_2 it completed with peer C (i.e., establishing session (S, C, sid)). Thus, the only way (S, sid) would be revealed in Π_1 is if (S, C, sid) was revealed in Π_2; but this is not possible since (S, C, sid) is matching to the fresh session (C, S, sid).

Proof of part (e). In Lemma 9 we show that, except for negligible probability, if (S, C, sid) is fresh in Π_2 then (C, S, sid) exists and completes in Π_2. In this case the sessions are matching hence they have the same key. By part (b) above, we then have that (C, S, sid) completes in Π_1 and has consistent keys with (C, S, sid) in Π_2, hence also consistent with the keys from (S, C, sid) in Π_2 (note that part (b) assumes C to be honest which follows here from the fact that (S, C, sid) is fresh in Π_2). It remains to show that (C, S, sid) is fresh in Π_1. First note that C and S are honest in Π_2 (follows from (S, C, sid) being fresh in Π_2), hence, by part (a) they are also honest in Π_1. Next, note that SIM does not reveal (C, S, sid) in Π_1 in order to obtain K_a for simulating the CSM message (for this session, SIM derives K_a from the real-or-random K). Thus, the only case in which (C, S, sid) would be revealed in Π_1 is if $\mathcal{A}_2$ revealed it in Π_2 but this would contradict the freshness of (S, C, sid) in Π_2. Finally, the matching session of (C, S, sid) in Π_1, namely (S, sid), is also unrevealed since such session can only be revealed if session sid of S in Π_2 is revealed. But this session is (S, C, sid) which is fresh, hence unrevealed.

Proof of part (f). We assume that SIM does not abort, namely, it guesses correctly the test session. In item 4 of the simulation, if the test session chosen by $\mathcal{A}_2$ in Π_2 is at a client, namely (C, S, sid), then SIM chooses (C, S, sid) in Π_1 as the test session. We show that if $\mathcal{A}_2$'s choice of a test session is valid, then, except for negligible probability, (C, S, sid) is a valid test session for $\mathcal{A}_1$. Indeed, by Lemma 7 (b), if (C, S, sid) completed in Π_2, it also completed in Π_1 and the two sessions have consistent keys. By Lemma 7 (d), if (C, S, sid) is a valid test session in Π_2 then (C, S, sid) is fresh in Π_1, hence a valid test session. If the test session chosen by $\mathcal{A}_2$ in Π_2 is a valid session (S, C, sid), then we need to show that (C, S, sid) in Π_1 is a valid test session for $\mathcal{A}_1$. Indeed, by Lemma 7 (e), if (S, C, sid) is a valid test session in Π_2 then, except for negligible probability, (C, S, sid) is fresh in Π_1, hence a valid test session, and the two sessions have consistent keys.

$\square$

The next two lemmas, needed to complete the proof of Lemma 7 (parts (d) and (e)) and Theorem 6, show, respectively, the need to include the identities C and S under the CSM MAC.

Lemma 8. *If C and S are honest, session (C, S, sid) is complete and unrevealed in Π_1, and session (S, sid) is complete in Π_1, then in Π_2 either S does not complete session sid or, if it does, it completes it as (S, C, sid) (except for a negligible probability).*

Proof: We are interested in the case where (S, sid) and (C, S, sid) complete in Π_1 (hence by Lemma 7 (part b) (C, S, sid) also completes in Π_2) and S completes session sid in Π_2 with some peer C'. We need to show that $C' = C$. Note that S will complete its session only if it received a CSM message signed by C' and with a MAC that included S and C' (actually, for this lemma, including S under the MAC is unnecessary so we will ignore it here). We assume $C' \neq C$ and consider two cases: C' is honest and C' is corrupted.

Note that C completed session (C, S, sid) hence sid includes a random nonce chosen by C (following our convention that sid includes random nonces by the peers to the session). This means that (up to a negligible nonce collision probability or signature forgery) no other honest party will create such sid and, in particular, no honest client will sign such sid as part of sid^*. Since, by assumption, party C' did sign sid^* then C' cannot be honest.

In the case where C' is corrupted, we have that S got a CSM message with a signature of a corrupted C' on sid^* (any C' can do that) and a MAC on C' under the key K_a computed in session (S, sid). We show that such ability of C' (i.e., of $\mathcal{A}_2$) to compute a MAC under K_a (with non-negligible probability) would allow SIM to make $\mathcal{A}_1$ win in Π_1 as soon as (S, C', sid) completes in Π_2 by testing session (C, S, sid) in Π_1. Indeed, if the Test query on (C, S, sid) in Π_1 returns the real session key then, by assumption, $\mathcal{A}_2$ has non-negligible probability to compute the above MAC. However, if the response is random then also K_a in (C, S, sid) in Π_2 is random and by MAC security $\mathcal{A}_2$ cannot compute $\mathsf{MAC}_{K_a}(C')$. So the ability to compute this MAC distinguishes the real session key in Π_1 from random with non-negligible probability, contradicting the UA-security of Π_1. It remains to show that (C, S, sid) can be chosen as the test session in Π_1, namely, that it is fresh at the time when (S, C', sid) completes in Π_2. Since (C, S, sid) is complete and unrevealed in Π_1, we only need to show that its matching in Π_1, namely (S, sid), is unrevealed. However, since (S, sid) can only be revealed in Π_1 after (S, C', sid) is revealed in Π_2, which can only happen after (S, C', sid) completed in Π_2, then at the time that (S, C', sid) completes, (S, C', sid) is unrevealed and so is (S, sid) in Π_1. $\square$

Lemma 9. *If Π_1 is treplication-secure and (S, C, sid) is fresh in Π_2 then, except for negligible probability, (C, S, sid) exists and completes in Π_2.*

Proof: Since (S, C, sid) completes in Π_2, then S received a signature of C on sid^* and a MAC on identities C and S under the key K_a of session (S, C, sid). Since C is honest, its signature on sid^*, if not forged, must have been generated by C after completing a session (C, S', sid) for some S'. If $S' = S$ then we are done showing that (C, S, sid) exists in Π_2 and is complete. If $S' \neq S$ then C computed $\mathsf{MAC}_{K_a}(C, S')$ but not $\mathsf{MAC}_{K_a}(C, S)$ as received and verified by S. Since the value $\mathsf{MAC}_{K_a}(C, S)$ is delivered by $\mathcal{A}_2$, this constitutes a MAC forgery by $\mathcal{A}_2$ under key K_a. We show that such a forgery has negligible probability of success by showing a treplication attacker that succeeds against protocol Π_1 with this same probability (and by treplication security this probability is negligible).

We consider the UA attacker $\mathcal{A}_1$ defined in the simulation of Section 5.1 as a treplication attacker. By the above analysis, $\mathcal{A}_1$ induced the creation of sessions (C, S', sid) and (S, sid) with same session id sid and same sid^*. To see that the two sessions have the same sid^*, note that in the CSM extension of the Π_1 run into a Π_2 run, S accepts the signature of C on sid^* (hence showing that except for a signature forgery event both parties ended the Π_1 run with the same view of sid^*). It remains to show that $\mathcal{A}_1$ can win a Test experiment on (S, sid). Let K be the response to the test query received by $\mathcal{A}_1$, namely, either the real session key computed by (S, sid) or a random value. To decide on "real" or "random", $\mathcal{A}_1$ completes a simulated run of $\mathcal{A}_2$ on Π_2. When $\mathcal{A}_2$ delivers a

MAC value M as part of the CSM message into session (S, sid), $\mathcal{A}_1$ derives a key K_a from K and checks that M passes $\mathsf{MAC}_{K_a}(C, S)$ verification. If so, $\mathcal{A}_1$ guesses "real", else it guesses "random". Thus, when K is random, the MAC is correct with negligible probability (by MAC security), while if K is real, a MAC is correct with the same probability of a successful MAC forgery by $\mathcal{A}_1$. Thus, this probability is the same (up to negligible difference) as the treplication advantage of $\mathcal{A}_1$, hence negligible. $\square$

5.3 Proof of Theorem 6

Consider simulation executions where SIM does not abort (i.e., SIM guesses correctly the test session) and in which the negligible probability events from the proofs of Lemmas 8 and 9 do not happen. In this case, Lemma 7 shows the validity of the actions of both adversaries, $\mathcal{A}_1$ and $\mathcal{A}_2$, as set by the simulator actions, and their consistency with the UA and MA models, respectively. In particular, the views of $\mathcal{A}_1$ and $\mathcal{A}_2$ under the simulated games are exactly as those of a real run of protocols Π_1 and Π_2, respectively. Thus, the winning probability of $\mathcal{A}_2$ is the same as in a real run of protocol Π_2 and by Lemma 7 (part f) $\mathcal{A}_1$ wins its test session whenever $\mathcal{A}_2$ does. We conclude that the winning advantage of $\mathcal{A}_2$ is upper bounded by that of $\mathcal{A}_1$. In all, we have that if m is an upper bound on the number of sessions initiated by $\mathcal{A}_2$, then the winning advantage of $\mathcal{A}_2$ against Π_2 is at most m times the winning advantage of $\mathcal{A}_1$ against Π_1 plus a negligible fraction. Hence, if any UA-attacker $\mathcal{A}_1$ against Π_1 has only negligible advantage, then so is the case for any MA-attacker $\mathcal{A}_2$ against Π_2. This completes the proof of MA-security of the compiled protocol Π_2.

Note on security quantification. The proofs of Lemmas 8 and 9 show events where the selection of a test session for $\mathcal{A}_1$ fails due to some adversarial action. The lemmas show these events to have negligible probability. An examination of the proof shows this probability to be at most $2 \cdot (n \cdot \epsilon_{sig} + \epsilon_{mac} + \epsilon_{kdf}) + \epsilon_{UA} + \epsilon_{tr} + \epsilon_{nc}$ where the ϵ values denote the security of various elements in the protocol: the unforgeability of the signature and MAC schemes used in SIGMAC, the pseudorandomness of the KDF, the UA-security of protocol Π_1, and the treplication security of Π_1 (these bounds represent the attackers' advantage for a given time bound); ϵ_{nc} bounds the probability of nonce collision among honest clients and n bounds the number of clients (the latter is needed in the client's signature-forging reduction for which one needs to guess the client in the test session in order to embed the attacked signature key). The above expression is further multiplied by factor m (denoting an upper bound on the number of sessions established by adversary $\mathcal{A}_2$ running against Π_2) to account for the probability $1/m$ that SIM does not abort its run, namely, it guesses correctly the session that $\mathcal{A}_2$ will choose as its test session.[9]

Note on signing S and C. Inspection of the proof of Lemma 9 shows that if the server's identity S is included under sid^*, i.e., signed by the client, then there is no need to include that identity under the MAC or to resort to treplication security. Yet, our results show that if signing this identity is omitted (for privacy reasons or simply because the identity is not part of the signed transcript - as in the resumption mode of TLS 1.3) its inclusion under the MAC is sufficient. As for the client's identity, we note that including this identity under the client's signature but not under a MAC would fail to provide MA security.

[9]This may result in a $m \cdot n$ factor multiplying ϵ_{sig} but in this case one can set m to be the maximal number of sessions established by a single client rather than representing the total number of sessions.

6 Post-Handshake and Encrypted Authentication

In this section we adapt the analysis of the SIGMAC compiler to some specific settings arising in the ongoing specification of TLS 1.3 [33]. We also introduce the functional CK model which may be of independent interest.

The SIGMAC compiler as analyzed in previous sections can be readily applied in the context of TLS 1.3 in the case that the client sends its authentication message (the signature and Finished message, or CSM in our terminology[10]) in the third handshake flight, i.e., immediately upon verification of server authentication. However, TLS 1.3 also includes some cases where the SIGMAC analysis from Section 5 does not apply directly. We identify three such cases. The first considers the fact that TLS 1.3 always encrypts the CSM message for reasons of identity protection. In the regular handshake case, the encryption key, or "handshake transport key (HTK)", is derived from the same intermediate key from which K_a and K_s are derived, and by the properties of the key derivation function HTK is (computationally) independent from K_a and K_s. Extending the SIGMAC proof for this case is immediate: the simulator derives the HTK key from the UA session key K, as it does with K_a and K_s, and uses it to encrypt/decrypt the CSM message. Due to the independence of HTK from K_a and K_s, the proof holds with minor adjustments.

In the second case the CSM message is also encrypted but in this case the encryption uses the *same key* K_s that the protocol outputs as the session key. This immediately violates key indistinguishability since by the time the client establishes the session key (which is when the key can be tested for indistinguishability), this key has been used already. We show how to adapt the analysis of SIGMAC to this case, at the cost of reducing the security guarantee. Rather than ensuring generic security of K_s (as induced by the regular indistinguishability property of key-exchange protocols), we show that K_s can be used as a key to an authenticated encryption scheme for implementing *secure channels*. Our analysis is general and can be applied to different modelings of secure channels (e.g. [20, 15]) as well as to other applications of the session key[11] as long as the security of such application is not voided by the use of K_s for encrypting CSM. Our formalization introduces the notion of *functional tests* (Section 6.1) that abstracts out details of applications and implementations.

The third case is the so called *"post-handshake client authentication"*. Here, the client authenticates with a CSM message but only after the server and client have already exchanged application data (i.e., record layer communication in the TLS context) protected with the session key K_s. The encrypted data exchange starts after the parties compute the session key but before the client authenticates. This raises questions about the notion of security that can be claimed and the level of protection provided to the exchanged data (to which we will refer as *pre-authentication data*). One can see that since the handshake protocol without client authentication provides unilateral authentication, then the data gets the assurance of unilateral authentication (server authentication in the case of TLS). However, the SIGMAC analysis from previous sections showing that the CSM message upgrades the protocol to MA security does not hold anymore once the session key K_s is used to protect data before CSM is delivered and verified. Yet, we are able to show that the

[10]We take the liberty of referring to the signature and MAC combination used in TLS 1.3 as "the CSM message".

[11]In the case of TLS 1.3, such applications may include the derivation/update of keys or the issuing of resumption keys via a New Session Ticket Message. Here the session key can be the application key or other keys computed in the protocol such as the exporter master secret.

CSM message does provide mutual authentication of the exchanged data but only in the sense of secure channels, and only after the CSM message is verified by the server. Namely, the parties get the guarantee (delayed for the server) that only the named party that passed authentication can authenticate and decrypt the exchanged information (including data sent before client authentication). Here too, our treatment via functional tests adds generality and allows us to abstract the details of the secure channels modeling and implementation.

Finally, combining the last two cases one obtains secure channels security for the case (also in TLS 1.3) where in addition to the exchange of pre-authentication data, the post-handshake CSM message itself is encrypted with the session key.

Fortunately, we can address all these scenarios with simple adaptations of the analysis of the SIGMAC compiler. We do so in the following subsections.

6.1 Functional Queries and Functional Tests

Since our goal is to show that the above authentication variants are sufficient to ensure "secure channels" functionality, it is worth recalling how such functionality is modeled. Take as example the ACCE model of Jager et al. [20] that has been used to prove security of several variants of TLS 1.2. Such model extends the usual key exchange security formalism with the ability of the attacker to run Encrypt and Decrypt queries on messages of its choice where the session key (or keys derived from it) serves as the authenticated encryption key. In addition, to capture the inability of the attacker to subvert the desired secure channels functionality, the attacker is tested through a dedicated game that replaces the standard indistinguishability test of key exchange models. In the ACCE model this test is adapted from the treatment of stateful authenticated encryption by Paterson et al. [31]. Other secure channels models differ in their details and scope but they follow a similar approach.

To increase generality and simplicity, our treatment abstracts out the details of implementation of secure channels, framing the above model ingredients in an abstract way. That is, we extend the capabilities of a key exchange attacker with the ability to query abstract "functional queries" on session keys and we replace the usual indistinguishability test with an abstract "functional test" that is run on a session chosen by the attacker (the "test session") and where the input to the test is the test session key. The only condition on such a test is that if one replaces the input session key with a random independent key, then the advantage of the attacker in winning the test is negligible. The ACCE model is an instantiation of this abstract framework where the functional queries represent the Encrypt/Decrypt queries and the session key experiment is modeled as a stateful authenticated encryption test (note that this test has the required property from a functional test, namely, if the authenticated encryption scheme is keyed with a random value then the attacker has negligible advantage in winning the ACCE game). This abstraction frees our analysis from dealing with specific details and models while at the same time adding generality. For example, one can use this abstraction to reason about whether a session key used to protect key exchange messages may still be secure for another purpose (say, to derive other keys) or whether a late client authentication implies authentication of keys derived prior to the authentication (e.g., a resumption key in TLS 1.3).

Functional family F. Let F be a parametrized family of (possibly randomized) functions that accept a single input to which we refer as a *session key.* Given a parameter p and a session key k, f_p denotes a member of F and $f_p(k)$ denotes an output distribution (or, more commonly, it denotes a specific value sampled from that distribution).

Functional queries and tests. We present a modification of the CK model of key exchange security [9] (which we recalled in Section 2). The only changes to the model are as follows (we refer by $\mathcal{A}_F$ to the attacker in this setting):

- We add a new adversarial query type, called a *functional query* FQuery, associated to a functional family F. $\mathcal{A}_F$ can issue FQuery against any session eligible for a Reveal query and also against the test session. In it, $\mathcal{A}_F$ provides a session identifier, a parameter p and gets back $f_p(k)$ where f_p is an element of F and k is the session key of the named session.
- We change the test session experiment: A test session can be chosen under the same rules as in the CK model but the regular indistinguishability-based Test query is replaced with a *functional test* FTest. This can be an arbitrary test taking the form of an interactive game between a challenger and attacker $\mathcal{A}_F$, where the challenger's input is the test session key and a random bit b, and the output of $\mathcal{A}_F$ is a bit b'. We say that $\mathcal{A}_F$ wins if $b = b'$. The only condition on a functional test is that if one replaces the input to the challenger with a random independent key then the advantage of $\mathcal{A}_F$ in winning the game is negligible.

We allow the functional queries to use a shared state (formally accommodated through the parameter p that can be seen as a generalized second input to the function) and also allow the functional test to share such state. This adds generality to our treatment and is needed, in particular, for the formalization of stateful authenticated encryption as used in the definition of ACCE security.

Functional CK model. The security definition in this functional model remains the same as in the original CK model (end of Section 2.1) except that the notion of winning for the attacker changes from the indistinguishability Test experiment to a given functional test FTest. We will refer to this notion as Functional CK security and we will denote the MA and UA variants by F-MA and F-UA (as in the regular CK model of Section 2, the main difference between UA and MA is that only clients have a defined peer and only client sessions can be chosen as test sessions).

Remark: For sessions other than the test session, the functional queries seem redundant as the attacker can always query these sessions via a SK reveal query and compute the functional query by itself. However, note that information learned via functional queries on a session can help the attacker choose that session as the test session (running a reveal query on a session would disqualify that session for testing). Moreover, quantifying security via the number of functional queries in an attack (e.g., number of encrypt and decrypt queries) can be an important measure. On the other hand, one may simplify the model by assuming that the attacker chooses its test session at random (at the cost of a number-of-sessions factor loss in the attacker's advantage) in which case functional queries are not needed, except for the test session and as part of defining a functional test.

Remark: We note that the above formalism via a bit-guessing game can be generalized. First, one can replace such a game with any experiment (not necessarily bit-guessing) where the winning probability of $\mathcal{A}_F$ on a run over a random independent key is negligible. Second, it would be enough to require that the difference between the success of the attacker on a run over the real session key be negligibly higher than on a run over a random key (even if the latter case is not in itself negligible). Finally, note that the functional approach can be applied to other key exchange models such as the Bellare-Rogaway [1]; a notion similar to functional queries was considered by Shoup [34]. One may also be able to use the ”suitable for” notion from Brzuska et al. [8] as a replacement for our functional tests. The two notions differ in their approaches but have a motivation in common, namely, replacing pure key indistinguishability with more limited functionalities where the use of the session key during the KE process does not void the security of the task at hand.

Secure channels via functional security. As discussed above, we are interested in freeing our results from the specific details of a secure channels model or its implementation. The idea is that any such formalism can be framed via the above functional model. Specifically, when referring to secure channels we will define two abstract functional queries, Encrypt and Decrypt, and will assume a given functional test. Encrypt and Decrypt will have the regular functionality of encrypting and decrypting messages but their implementation can take different forms (e.g., they can be defined via general authenticated encryption or with a more specific scheme, they can be stateless or stateful, apply to full messages or fragments, etc.). The functional test can also take different forms, e.g. [20, 15]. For example, [20] defines its model, ACCE security, through a semantic security game that mimics the game used to define the security of the underlying (stateful) authenticated encryption scheme [31] and relies on the fact that no attacker can win the latter game if the key in use is fresh and random. Our formalism can represent such a model by instantiating the functional test via this particular ACCE game.

6.2 SIGMAC with encrypted CSM

We consider a variant of the SIGMAC compiler in which the CSM message is encrypted with the session key K_s from the same session to which the CSM message belongs. We show that any application whose security can be defined via a functional test as in Section 6.1, and where the encryption and decryption of CSM are included in the functional queries of the model, enjoys mutual authentication in the sense of F-MA. As an application, instantiating our abstract formalism with a secure channels model (e.g., [20, 15]), we get that the session key validated via SIGMAC with encrypted CSM implements secure channels with mutual authentication (it is assumed that the secure channels mechanism enforces domain separation between the encryption of CSM and encryption of application data, for example as the ACCE model does via stateful encryption).

Let $\Pi_2{}'$ be a protocol resulting from applying SIGMAC with encrypted CSM to a UA-secure Π_1 protocol. (It is possible to include further information under this encryption such as the client's identity or certificate to protect it from eavesdroppers.) The encryption of CSM is generated by the client and the corresponding decryption and verification is performed by the receiving server. We prove this modified compiler following the original proof of Theorem 6 but we replace the regular indistinguishability-based security test with a functional test as defined in Section 6.1; we also assume Encrypt and Decrypt operations as part of the allowed functional queries[12]. We prove:

[12]Note that the attacker does not provide the message CSM for encryption, but rather the functional query is defined as the application of Encrypt to the value CSM generated by the client and possibly unknown to the attacker.

Lemma 10. *The SIGMAC compiler with message CSM encrypted under session key K_s and applied to a treplication-secure UA-secure protocol (under the conditions of Theorem 6) results in a F-MA-secure protocol where the session indistinguishability test is replaced with a functional test (defined by the protocol) and the* Encrypt *and* Decrypt *operations applied to CSM are considered as valid functional queries on session key K_s.*

Proof (sketch). The proof follows the same simulation argument as the proof of Theorem 6 with two important adjustments. The first refers to the handling of CSM encryption and decryption. As before, SIM learns keys K_s and K_a by invoking $\mathcal{A}_1$ to issue reveal queries to the appropriate sessions, but now SIM also uses K_s to encrypt outgoing CSM messages from client's sessions as well as to decrypt incoming CSM messages in servers' sessions. The second change regards replacing the Test query in the original model with a functional test FTest defined by the protocol. The actions of SIM related to the choice and handling of the test session are the same as in the original simulation's step 4 with the addition that the key K_s derived from the real-or-random key K (that $\mathcal{A}_1$ received as a response to its Test query) is now used also for the encryption and decryption of the CSM message. When F-MA attacker $\mathcal{A}_2{}'$ running against $\Pi_2{}'$ chooses its test session (if this is not the guess session then SIM aborts) and issues a FTest query, SIM chooses a random bit b and uses this bit and the real-or-random K as its inputs to the functional test experiment. When $\mathcal{A}_2{}'$ stops with output bit b', SIM instructs $\mathcal{A}_1$ to output "real" if $b = b'$ and "random" otherwise.

We have that in the case that K is real then the run of $\mathcal{A}_2{}'$ as orchestrated by SIM corresponds to a real run of $\mathcal{A}_2{}'$. If we assume that $\mathcal{A}_2{}'$ wins the real game with non-negligible advantage then this is the case here when K is real. On the other hand, if K is random, then $\mathcal{A}_2{}'$ is being tested on a functional test with a (pseudo) random key, hence by definition, $\mathcal{A}_2{}'$ has only negligible winning advantage. This translates into a non-negligible advantage of $\mathcal{A}_1$ in responding correctly to the real-or-random test, in contradiction to the UA-security of Π_1. The formalization of this argument is standard and omitted from this proof outline.

□

Application to ACCE security. As previously stated our functional security model allows us to frame secure channel formalisms. One specific instance is the ACCE model from [20] that has been successfully applied to the analysis of TLS protocols (but see the refined definitions of [15]). We omit the ACCE details here and informally claim the following corollary. Please consult [20] for the model (including a definition of pre-accept and post-accept phases), the definition of the Encrypt and Decrypt operations (that we represent as functional queries), and their security game that instantiates our functional test.

Corollary 11 (informal)**.** *Let Π be a protocol resulting from the application of the SIGMAC compiler with encrypted CSM to a treplication-secure UA-secure protocol, and let* Encrypt *and* Decrypt *denote a stateful encryption scheme as considered by the ACCE model [20]. Consider protocol Π' that runs Π in the pre-accept phase of ACCE and uses the resultant session key as the key for the* Encrypt *and* Decrypt *operations in the post-accept phase as well as for the encryption of CSM* [13]*. Then, Π' is secure according to the (mutual authentication) ACCE model.*

[13]It is assumed that the Encrypt state is initialized and used first for encrypting the CSM message and then applied to messages exchanged in the post-accept phase.

6.3 Security under post-handshake authentication

Here we address the third case discussed in this section's introduction, namely, where data protected under the session key is exchanged between server and client after the server computed the session key but before the client sends the CSM message. This case, known as *"post-handshake client authentication"* in the context of TLS 1.3, requires a weakening of the security guarantees as the early use of the session key voids key indistinguishability. Clearly, since the protocol is UA-secure even without the CSM message then the data exchanged prior to the sending of CSM is UA-secure. But what notion of security can one claim once the client authenticates? Fortunately, using a simple adaptation of the proof of the SIGMAC compiler to the present setting, we are able to show that the data exchanged before the sending of CSM (as well as later data) can be considered to be protected by mutually authenticated secure channels, with the assurance for the server delayed until it verifies the client's CSM message. We frame this result in our more general functional setting (Section 6.1).

Let Π_2 be a protocol resulting from applying SIGMAC to a UA-secure protocol Π_1, namely, Π_2 consists of running Π_1 and adding to it the CSM message from client to server with a defined sid^* value. Let $\Pi_2{}'$ be a modification of Π_2 where between the last message of Π_1 and the CSM message from the client, data (referred to as *pre-authentication data*) is exchanged encrypted under the session key K_s generated at the end of the Π_1 run. We frame this protocol in our functional setting by modeling the encryption and decryption operations applied to the pre-authentication data with abstract $\mathsf{Encrypt}$ and $\mathsf{Decrypt}$ functional queries.

We say that a *functional test is consistent with a set of functional queries* if the advantage of any adversary to win the test with a random key remains negligible even when allowed to issue functional queries from this set.

Lemma 12. *Given a treplication-secure UA protocol* Π_1 *(i.e., one that satisfies the conditions of Theorem 6), Protocol* $\Pi_2{}'$ *as defined above is F-MA-secure with respect to any functional test consistent with the* $\mathsf{Encrypt}$ *and* $\mathsf{Decrypt}$ *functional queries defined by the protocol.*

Proof (sketch). The proof follows the original proof of Theorem 6 with some important adjustments in the simulator SIM's actions. When the F-MA attacker $\mathcal{A}_2{}'$ against $\Pi_2{}'$ invokes an $\mathsf{Encrypt}$ or $\mathsf{Decrypt}$ query at a non-test session (S, sid), then do nothing if the session is not yet complete. Else, reveal the session and use its value K_s with the $\mathsf{Encrypt}/\mathsf{Decrypt}$ queries (if K_s was previously revealed, use it). Similarly, for queries issued at a (C, S, sid) session, SIM reveals that session to obtain K_s and respond to the queries. When $\mathcal{A}_2{}'$ invokes (C, S, sid) to send CSM, reveal this session if K_a is not yet revealed. The guessed test session is chosen as in the proof of Theorem 6. If it is of the form (C, S, sid) or (S, C, sid), as soon as the server sends its last UA message in session (S, sid), SIM delivers this message to C and invokes $\mathcal{A}_1$ to issue Test against session (C, S, sid) in Π_1. Let K be the response to that query (i.e., the real-or-random key). SIM derives from it keys K_a and K_s, and uses K_s to respond to functional queries $\mathsf{Encrypt}$ and $\mathsf{Decrypt}$ against the test session. When attacker $\mathcal{A}_2{}'$ chooses its test session (if this is not the guess session then SIM aborts) and issues a FTest query, SIM chooses a random bit b and uses this bit and K_s, derived from the real-or-random K, as its inputs to the functional test experiment. When $\mathcal{A}_2{}'$ stops with output bit b', SIM instructs $\mathcal{A}_1$ to output "real" if $b = b'$ and "random" otherwise.

We have that in the case that K is real (and so is K_s) then the run of $\mathcal{A}_2{}'$ as orchestrated by SIM corresponds to a real run of $\mathcal{A}_2{}'$ against protocol $\Pi_2{}'$. If we assume that $\mathcal{A}_2{}'$ wins the real game with non-negligible advantage then this is the case here when K is real. On the other hand, if K is random, then $\mathcal{A}_2{}'$ is being tested on a functional test with a (pseudo) random key K_s, hence by definition, $\mathcal{A}_2{}'$ has only negligible winning advantage. This translates into a non-negligible advantage of $\mathcal{A}_1$ in responding correctly to the real-or-random test, in contradiction to the UA-security of Π_1. The formalization of this argument is standard and omitted from this proof outline. □

Application to ACCE security. A corollary of this lemma is that if we instantiate the functional test with a security experiment for a given "secure channels" model then we obtain that the encrypted pre-authentication data attains the same protection as guaranteed by the secure channels model. Illustrating this in the ACCE model [20] needs some adjustment since ACCE separates key exchange (pre-accept phase) and encrypted message exchange (post-accept phase) while in our case these two are interleaved. What we are really interested in is the security of the encrypted message exchange protocol. To capture the latter form of security we could model the pre-accept phase with an (idealized) trusted distribution of pairwise session keys between honest parties. If a post-accept message exchange protocol has the property that when coupled with the idealized key exchange results in an ACCE secure protocol then we say that the message exchange protocol is *post-accept ACCE secure*. In this setting, what the above lemma says is that the encrypted pre-authentication data in protocol $\Pi_2{}'$ is post-accept ACCE secure. (Note that the ACCE model is defined with mutual authentication which is ensured here, although the assurance for the server is delayed until it verifies the client's CSM message.) We omit a more formal treatment here.

On multiple client authentications. TLS 1.3 allows for multiple post-handshake client authentication executions, with same or different client certificates, in the same session. The above results can be applied also to this context. We note, however, that if the same client certificate is used repeatedly then there must be a new server's nonce (or other freshness value) included under the signature and MAC or otherwise a trivial replay attack applies. TLS 1.3 includes such a value, called a `certificate_request_context`.

6.3.1 Security under post-handshake authentication with encrypted CSM.

One can extend the result from Lemma 12 to the case where in addition to the encryption of pre-authentication data, the CSM message itself is encrypted as in Section 6.2 (this corresponds to the TLS 1.3 specification). The simulation combines the simulators from the proofs of Lemma 10 and Lemma 12 using the key K_s to encrypt both pre-authentication data and the CSM message. Note that the encrypted CSM is delivered only after the encryption of pre-authentication data, hence in a stateful encryption its ciphertext will depend on prior encryptions. The simulation will also deliver the encrypted CSM at this time, hence it will encrypt it correctly (a crucial point is that there is no need to send CSM out of order in the simulation).

Acknowledgment.

This paper has benefited enormously from collaboration with Eric Rescorla and Hoeteck Wee on the ongoing design of TLS 1.3, and from enlightening inputs from Björn Tackmann, Douglas Stebila, Felix Günther, Karthik Bhargavan, Kenny Paterson, Markulf Kohlweiss and many of the other participants in the TRON workshop and the TLS working group.

References

[1] M. Bellare and P. Rogaway. Entity authentication and key distribution. In D. R. Stinson, editor, *CRYPTO*, volume 773 of *Lecture Notes in Computer Science*, pages 232–249. Springer, 1993. ISBN 3-540-57766-1.

[2] M. Bellare, R. Canetti, and H. Krawczyk. A modular approach to the design and analysis of authentication and key exchange protocols (extended abstract). In *30th ACM STOC*, pages 419–428. ACM Press, May 1998.

[3] B. Beurdouche, K. Bhargavan, A. Delignat-Lavaud, C. Fournet, M. Kohlweiss, A. Pironti, P. Strub, and J. K. Zinzindohoue. A messy state of the union: Taming the composite state machines of TLS. In *IEEE Symposium on Security and Privacy*, 2015.

[4] K. Bhargavan and G. Leurent. Transcript collision attacks: Breaking authentication in tls, IKE and SSH. In *23nd Annual Network and Distributed System Security Symposium, NDSS 2016, San Diego, California, USA, February 21-24, 2016*. The Internet Society, 2016. URL `http://www.internetsociety.org/events/ndss-symposium-2016`.

[5] K. Bhargavan, C. Fournet, M. Kohlweiss, A. Pironti, and P.-Y. Strub. Implementing TLS with verified cryptographic security. In *IEEE Symposium on Security and Privacy*, 2013. URL `http://mitls.rocq.inria.fr/`.

[6] K. Bhargavan, A. Delignat-Lavaud, C. Fournet, A. Pironti, and P. Strub. Triple handshakes and cookie cutters: Breaking and fixing authentication over TLS. In *2014 IEEE Symposium on Security and Privacy, SP*, pages 98–113, 2014.

[7] K. Bhargavan, A. Delignat-Lavaud, and A. Pironti. Verified contributive channel bindings for compound authentication. In *NDSS*, 2015.

[8] C. Brzuska, M. Fischlin, N. P. Smart, B. Warinschi, and S. C. Williams. Less is more: relaxed yet composable security notions for key exchange. *Int. J. Inf. Sec.*, 12(4):267–297, 2013. Cryptology ePrint Archive, Report 2012/242.

[9] R. Canetti and H. Krawczyk. Analysis of key-exchange protocols and their use for building secure channels. In *EUROCRYPT*, pages 453–474, 2001. See also Cryptology ePrint Archive, Report 2001/040.

[10] R. Canetti and H. Krawczyk. Universally composable notions of key exchange and secure channels. In *EUROCRYPT*, pages 337–351, 2002. See also Cryptology ePrint Archive, Report 2002/059.

[11] C. Cremers, M. Horvat, S. Scott, and T. van der Merwe. Automated verification of TLS 1.3: 0-RTT, resumption and delayed authentication. In *IEEE S&P 2016.*, 2016.

[12] B. Dowling, M. Fischlin, F. Günther, and D. Stebila. A cryptographic analysis of the TLS 1.3 handshake protocol candidates. In *ACM CCS*, 2015. Also, Cryptology ePrint Archive, Report 2015/914.

[13] B. Dowling, M. Fischlin, F. Günther, and D. Stebila. A cryptographic analysis of the TLS 1.3 draft-10 full and pre-shared key handshake protocol. Cryptology ePrint Archive, Report 2016/081, 2016.

[14] M. Fischlin and F. Günther. Multi-stage key exchange and the case of Google's QUIC protocol. In *ACM CCS*, pages 1193–1204, 2014.

[15] M. Fischlin, F. Günther, G. A. Marson, and K. G. Paterson. Data is a stream: Security of stream-based

channels. In R. Gennaro and M. J. B. Robshaw, editors, *CRYPTO 2015, Part II*, volume 9216 of *LNCS*, pages 545–564. Springer, Heidelberg, Aug. 2015. doi: 10.1007/978-3-662-48000-7_27.

[16] I. Goldberg, D. Stebila, and B. Ustaoglu. Anonymity and one-way authentication in key exchange protocols. *Des. Codes Cryptography*, 67(2):245–269, 2013. doi: 10.1007/s10623-011-9604-z. URL http://dx.doi.org/10.1007/s10623-011-9604-z.

[17] S. Halevi and H. Krawczyk. Public-key cryptography and password protocols. *ACM Transactions on Information and System Security*, 2(3):230–268, Aug. 1999.

[18] S. Halevi and H. Krawczyk. One-pass HMQV and asymmetric key-wrapping. In *PKC 2011*, pages 317–334, 2011.

[19] T. Jager, F. Kohlar, S. Schäge, and J. Schwenk. Generic compilers for authenticated key exchange. In M. Abe, editor, *ASIACRYPT 2010*, volume 6477 of *LNCS*, pages 232–249. Springer, Heidelberg, Dec. 2010.

[20] T. Jager, F. Kohlar, S. Schäge, and J. Schwenk. On the security of TLS-DHE in the standard model. In *CRYPTO*, pages 273–293, 2012. Also Cryptology ePrint Archive, Report 2011/219.

[21] F. Kohlar, S. Schäge, and J. Schwenk. On the security of TLS-DH and TLS-RSA in the standard model. Cryptology ePrint Archive, Report 2013/367, 2013. http://eprint.iacr.org/.

[22] M. Kohlweiss, U. Maurer, C. Onete, B. Tackmann, and D. Venturi. (De-)constructing TLS. Cryptology ePrint Archive, Report 2014/020, 2014. revised Apr 2015.

[23] M. Kohlweiss, U. Maurer, C. Onete, B. Tackmann, and D. Venturi. (de-)constructing TLS 1.3. In *Progress in Cryptology - INDOCRYPT 2015 - 16th International Conference on Cryptology in India, Bangalore, India, December 6-9, 2015, Proceedings*, pages 85–102, 2015.

[24] H. Krawczyk. SIGMA: The "SIGn-and-MAc" approach to authenticated Diffie-Hellman and its use in the IKE protocols. In *CRYPTO*, pages 400–425, 2003.

[25] H. Krawczyk and H. Wee. The OPTLS protocol and TLS 1.3. In *EuroS&P*, 2016.

[26] H. Krawczyk, K. G. Paterson, and H. Wee. On the security of the TLS protocol: A systematic analysis. In *CRYPTO (1)*, pages 429–448, 2013. Also, Cryptology ePrint Archive, Report 2013/339.

[27] A. Langley and W.-T. Chang. QUIC crypto, 2013. URL http://tinyurl.com/lrrjyjs.

[28] R. Lychev, S. Jero, A. Boldyreva, and C. Nita-Rotaru. How secure and quick is QUIC? Provable security and performance analyses. In *IEEE Symposium on Security and Privacy*, pages 214–231, 2015.

[29] U. Maurer, B. Tackmann, and S. Coretti. Key exchange with unilateral authentication: Composable security definition and modular protocol design. *IACR Cryptology ePrint Archive*, 2013:555, 2013. URL http://eprint.iacr.org/2013/555.

[30] P. Morrissey, N. P. Smart, and B. Warinschi. A modular security analysis of the TLS handshake protocol. In *ASIACRYPT*, pages 55–73, 2008.

[31] K. G. Paterson, T. Ristenpart, and T. Shrimpton. Tag size does matter: Attacks and proofs for the TLS record protocol. In *ASIACRYPT*, pages 372–389, 2011.

[32] M. D. Raimondo, R. Gennaro, and H. Krawczyk. Deniable authentication and key exchange. In *ACM CCS*, 2006.

[33] E. Rescorla. The transport layer security (TLS) protocol version 1.3 (draft 13), Dec. 2015. URL https://tools.ietf.org/html/draft-ietf-tls-tls13-13.

[34] V. Shoup. On formal models for secure key exchange. Cryptology ePrint Archive, Report 1999/012, 1999. http://eprint.iacr.org/.