An extended abstract of this paper appears in the proceedings of CRYPTO'23. This is the full version.

On Optimal Tightness for Key Exchange with Full Forward Secrecy via Key Confirmation

Kai Gellert[1], Kristian Gjøsteen[2], Håkon Jacobsen[3,4], and Tibor Jager*[1]

[1]University of Wuppertal, {kai.gellert,jager}@uni-wuppertal.de, [2]Norwegian University of Science and Technology, kristian.gjosteen@ntnu.no, [3]Thales Norway, [4]University of Oslo, hakon.jacobsen@its.uio.no

August 11, 2023

Abstract

A standard paradigm for building key exchange protocols with *full* forward secrecy (and *explicit* authentication) is to add key confirmation messages to an underlying protocol having only *weak* forward secrecy (and *implicit* authentication). Somewhat surprisingly, we show through an impossibility result that this simple trick must nevertheless incur a linear tightness loss in the number of parties for many natural protocols. This includes Krawczyk's HMQV protocol (CRYPTO 2005) and the protocol of Cohn-Gordon et al. (CRYPTO 2019).

Cohn-Gordon et al. gave a very efficient underlying protocol with *weak* forward secrecy having a linear security loss, and showed that this is optimal for certain reductions. However, they also claimed that full forward secrecy could be achieved by adding key confirmation messages, and *without any additional loss*. Our impossibility result disproves this claim, showing that their approach, in fact, has an overall *quadratic* loss.

Motivated by this predicament we seek to restore the original linear loss claim of Cohn-Gordon et al. by using a different proof strategy. Specifically, we start by lowering the goal for the underlying protocol with weak forward secrecy, to a *selective* security notion where the adversary must commit to a long-term key it cannot reveal. This allows a *tight* reduction rather than a linear loss reduction. Next, we show that the protocol can be upgraded to full forward secrecy using key confirmation messages with a linear tightness loss, even when starting from the weaker selective security notion. Thus, our approach yields an *overall* tightness loss for the fully forward-secret protocol that is only linear, as originally claimed. Finally, we confirm that the underlying protocol of Cohn-Gordon et al. can indeed be proven selectively secure, tightly.

*Supported by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme, grant agreement 802823.

1 Introduction

A security reduction is said to be *tight* if it preserves the security of the object being reduced to. The benefit of a tight reduction is that it allows to closely relate the security of a complex object to a simpler and, hopefully, more easy to analyze object. Moreover, a tight reduction allows cryptographic schemes to be instantiated with optimal parameters in a theoretically sound way. Unfortunately, for key exchange protocols the security reductions have historically been inordinately *non-tight*. But recent works have started to address these deficiencies either by doing more careful analyses of existing protocols, or by proposing new protocols more suitable for tighter reductions. Typically the approach has either been to use digital signatures with tight multi-user security [BHJ+15, GJ18, PQR21, HJK+21], or signature-less protocols based on some variant of Diffie-Hellman [CCG+19, JKRS21]. While the latter tend to be more efficient than the signature-based approaches in practice, the comparison isn't completely fair since the signature-based protocols provide *full* forward secrecy and *explicit* authentication (AKE) while the protocols based on Diffie-Hellman ones only give *weak* forward secrecy and *implicit* authentication.

Forward secrecy, authentication and key confirmation. *Full* forward secrecy refers to the ability of a protocol to provide security of session keys even if the long-term secret key of a party is leaked [BG20]. It is considered an essential standard security goal for modern key exchange protocols. *Weak* forward secrecy achieves this property under the assumption that the adversary does not actively interfere with the protocol messages in the sessions it attacks.

The notions of weak and full forward secrecy are also intimately connected to authentication [BPR00, BG11, dFW20]. *Explicit* authentication guarantees that the intended communication partner is indeed "online", having actively participated in the protocol and derived the session key. *Implicit* authentication, on the other hand, only guarantees that the intended peer will be able to derive the same session key, not that it actually has participated in the protocol. Typically, for key exchange protocols having only implicit authentication, an adversary can efficiently impersonate Alice towards Bob, in the sense that it sends messages on behalf of Alice and such that Bob derives a session key that he believes is suitable for communicating with Alice, but the adversary will still not be able to distinguish this session key from a random one.

A natural question then is how to upgrade a protocol from weak forward secrecy and implicit authentication to full forward secrecy and explicit authentication, while still maintaining tightness and efficiency. Probably the most obvious idea is to have the participants send *key confirmation* messages derived from the session key. This solution is simple, efficient, and has already been treated in multiple works [BPR00, Kra05, Yan13, FGSW16, CCG+19, dFW20]. Key confirmation ensures that the other protocol participant has indeed computed the same session key, turning an implicitly authenticated protocol into an explicitly authenticated protocol [dFW20]. It also serves a dual role of upgrading weak forward secrecy to full forward secrecy. The HMQV-C [Kra05] protocol

is a notable example of this usage. In this paper we use key confirmation in both senses above.

Finally, while implicit (resp. explicit) authentication often corresponds to weak (resp. full) forward secrecy, we note that these are in fact separate notions. Protocols having explicit authentication but no forward secrecy are common (see for instance the AKEP1 protocol in [BR94, Fig. 2]). Examples of protocols achieving full forward security without explicit authentication are given in [BG11, Protocol 4] and [CF15, Fig. 3].

Does key confirmation preserve tightness? Suppose Π is an arbitrary key exchange protocol providing weak forward secrecy and implicit authentication. The protocol participants exchange a session key using Π and from this derive key confirmation messages as well as a new session key using a pseudorandom function (PRF). They then exchange and verify the confirmation messages before outputting the new session key. Call this extended protocol Π^+. Intuitively, protocol Π^+ should achieve explicit authentication via a *tight* reduction to the implicit authentication of protocol Π as well as the multi-user security of the PRF. Indeed, this is the claim of Theorem 6 in [CCG+19]. Unfortunately, this claim turns out to be wrong. In fact, as we will show, for certain natural protocols, such as the protocol from Cohn-Gordon et al. [CCG+19] and HMQV [Kra05], adding key confirmation messages like this or in any other deterministic way must *necessarily* lose a factor of U, where U is the number parties in the protocol. Hence, it seems that the notions of weak forward secrecy and implicit authentication are too weak to be *tightly* upgraded to full forward secrecy and explicit authentication by simply adding key confirmation messages. Interestingly and surprisingly, we will however also argue that an even weaker *selective* security notion is sufficient to obtain security with the same linear loss, which provides a new approach to obtain full forward secrecy with optimal linear tightness loss.

The flaw in Cohn-Gordon et al. [CCG+19]. The high-level idea of the security reduction from protocol Π^+ to protocol Π in [CCG+19] is as follows. The reduction uses Test or Reveal queries to get the session keys from Π and uses these to simulate the key confirmation messages of protocol Π^+. However, the reduction must decide which session keys it will reveal and which it will issue a Test query for. The trivial standard strategy would be to guess which session the adversary will test, but this cannot be deployed in [CCG+19] as it would immediately incur a linear security loss in the number of users *times* the number of sessions per user. Instead, the reduction proceeds as follows: once a session has reached an accepting state in the underlying AKE protocol Π, the reduction will base its decision on which query to use on the *current* freshness of the session. If the session is not fresh, it will issue a Reveal query. If the session is fresh, it will issue a Test query.

The problem with this strategy is that the freshness notion is with respect to protocol Π, which is only guaranteeing *weak* forward secrecy. Unfortunately

this notion is too weak (i.e., too restrictive) to accommodate the reduction. More specifically, in a weak forward secrecy model the adversary is forbidden from both being active in a Test session and revealing the long-term secret of its peer. This is due to a classic attack described by [BPR00] and [Kra05] (see [BG11] and [CF15] for further discussions).

In this attack the adversary $\mathcal{A}$ impersonates Alice towards Bob by creating the DH share g^x on her behalf. Once Bob receives this message he creates its own DH share g^y and accepts in protocol Π. Since $\mathcal{A}$ has not (yet) revealed the long-term key of Alice, Bob is at this point still fresh in protocol Π according to weak forward secrecy. Consequently, the reduction will issue a Test query in order to simulate its key confirmation message. However, if $\mathcal{A}$ now reveals the long-term key of Alice, then Bob will no longer be fresh (in protocol Π). At this point the reduction is stuck. This means that the reduction in Theorem 6 of Cohn-Gordon et al. [CCG+19] does not work.

1.1 Our contributions

While the reduction of [CCG+19] does not work, can the result nevertheless be salvaged? On the one hand, we show that a tight reduction from full forward secrecy and explicit authentication to weak forward secrecy and implicit security is impossible for a large class of compilers and protocols of interest for practical applications. In particular, this includes the common key confirmation message compiler discussed above and the key exchange protocol of [CCG+19]. We prove this using a meta-reduction described in more detail below.

On the other hand, by considering what the actual *end goal* of [CCG+19] is, we can in fact recover the intended result by a rearranging of arguments. That is, the end goal is to create an as efficient as possible key exchange protocol having full forward secrecy and explicit authentication, with optimal tightness. Here, tightness is with respect to the lowest-level building block of the protocol. In the case of [CCG+19] this is the strong Diffie-Hellman (stDH) assumption [ABR01]. It was shown in [CCG+19] that a large class of DH-based implicitly authenticated key exchange protocols must lose a factor of U when reducing to stDH, where U is the number of parties. If the reduction from Π^+ to Π had been tight, as mistakenly claimed in [CCG+19], then the overall result would have been a protocol Π^+ with full forward secrecy and an optimal tightness loss of U to the stDH assumption. However, in light of our impossibility result, the best one can hope for using this approach is a loss of U^2, since there is a tightness loss of U going from Π^+ to Π and a tightness loss of U going from Π to stDH.

But this begs the question: if we know from the beginning that we at least have to lose a factor of U, is there some other way of structuring our arguments in order to avoid a quadratic loss? The solution is to first reduce the security of protocol Π^+ to an even *weaker* notion of implicit security for protocol Π, taking the "hit" of U here. Then, we show that this weaker notion for Π can be reduced further to stDH but now *tightly*. Thus, overall we obtain a modular reduction from Π^+ down to stDH losing only a factor of U.

What is this weaker notion for Π? It is a type of *selective security* game

where the adversary must commit to a single party it will not reveal the long-term key of. This is related to the selective security notion from [KPW13], but differs in two important ways. First, the requirement that one long-term key must stay *unrevealed*—rather than simply being involved in some event—makes the two notions technically incomparable (see Remark 3.8). Second, in [KPW13] the adversary commits to *both* parties *and their sessions* involved in the event. This incurs a quadratic security loss, making it unsuitable for our purposes.

In summary, our main results are:

- We give a generic impossibility result showing that no security proof for adding key confirmation to a weakly forward-secret key exchange protocol can avoid a loss factor of U (Section 6).
- We provide an optimal security proof (i.e., with a linear loss in U) for adding key confirmation, which reduces to a weaker security notion for the underlying key exchange protocol (Section 4). This weaker notion allows us to avoid a tightness loss when proving the underlying protocol secure.
- Finally, we give a tight proof of the CCGJJ protocol [CCG+19] under the weaker notion, showing that the overall strategy achieves the end goal (Section 5).

One important consequence of our work is that future key exchange protocols having only weak forward secrecy can now be designed towards the goal of selective key secrecy, not full key secrecy. As shown by the analysis of CCGJJ [CCG+19], this may simplify proofs significantly.

Basic idea of the impossibility result. Our impossibility result shows essentially that if one constructs a protocol Π^+ from an underlying implicitly authenticated protocol Π by extending Π with two additional key confirmation messages, and if the security analysis of Π^+ includes a reduction $\mathcal{R}$ to the security of Π, then $\mathcal{R}$ loses a factor which is at least linear in the number U of parties. The basic idea of the argument is as follows.

We first define a (hypothetical) adversary $\mathcal{A}$, which proceeds in four steps:

1. First $\mathcal{A}$ receives the public keys $\mathsf{pk}_1, \ldots, \mathsf{pk}_U$ of all parties.
2. Then it interacts with $\mathcal{R}$ to create a session $s_{i,j}$ of protocol Π^+ for every pair of parties i, j. In all of these sessions the protocol is executed until $\mathcal{R}$ outputs the first of the two key confirmation messages.

   Note that $\mathcal{R}$ may simulate messages of Π^+ that correspond to messages of the underlying protocol Π by relaying these messages to its own security experiment. However, $\mathcal{R}$ also has to simulate the first key confirmation messages, which depend on the session key k of protocol Π.
3. Finally, $\mathcal{A}$ reveals the long-term secret keys of all but one party, receiving sk_i for all $i \in \{1, \ldots, U\} \setminus \{i^*\}$, where i^* is chosen at random by $\mathcal{A}$. Then $\mathcal{A}$

uses these secret keys to verify all key confirmation messages received from $\mathcal{R}$ for all sessions $s_{i,j}$ with $i \neq i^*$. If at least one of these key confirmation messages is invalid, then $\mathcal{A}$ terminates.

4. If all key confirmation messages are correct, then $\mathcal{A}$ breaks the security of Π^+ in a target session $s_{i^*,j}$ for some j.

$\mathcal{A}$ is a valid adversary that breaks Π^+ in the security experiment with maximal advantage. The choice of i^* is perfectly hidden from the reduction until Step 3 of $\mathcal{A}$, as all queries in Step 2 are independent of i^*. Note in particular that we can trivially simulate $\mathcal{A}$, if $\mathcal{R}$ outputs at least one invalid key confirmation message for any session $s_{i,j}$, $i \neq i^*$. Furthermore, note that we can always simulate the first three steps of $\mathcal{A}$ efficiently.

We will essentially argue that the reduction $\mathcal{R}$ is only able to simulate all key confirmation messages of sessions $s_{i,j}$ of parties $i \neq i^*$ properly, if it asks its security experiment to reveal the corresponding session keys $k_{i,j}$. However, at the same time $\mathcal{R}$ must not ask its security experiment to reveal the session key $k_{i^*,j}$ of the target session $s_{i^*,j}$, as otherwise it cannot leverage $\mathcal{A}$ to break the security of this session. Hence, the reduction faces the challenge that it has to "predict" i^* already in Step 2 of the adversary, in order to make sure that the key confirmation messages are simulated correctly, but still $\mathcal{A}$ can be leveraged to break the security of Π. Since i^* is chosen uniformly from $\{1, \ldots, U\}$, this yields a linear loss in U.

We stress that this sketch of the impossibility result is simplified, the actual formal result is more involved and subtle. For instance, in Section 6.1 we formulate precise conditions on which classes of reductions, protocols Π, and which constructions of Π^+ are covered by the impossibility result. These will cover the construction from [CCG+19] but also many other natural constructions.

The common way of arguing that a reduction $\mathcal{R}$ does not "need" $\mathcal{A}$ in certain cases is to perform a meta-reduction where $\mathcal{A}$ can efficiently be simulated in these cases. Normally, the standard approach of meta-reductions used in many prior works, such as [HJK12, LW14, BJLS16], is to *rewind* $\mathcal{R}$ in order to be able to simulate $\mathcal{A}$ properly. Unfortunately, these results are usually only able to rule out reductions to *non-interactive* hardness assumptions. In contrast, the assumption that Π is secure is *interactive*. By rewinding $\mathcal{R}$ and running it multiple times with different queries from the "snapshot" state, we might cause $\mathcal{R}$ to make a sequence of queries that is not allowed in the key exchange security experiment of Π, such as revealing and then testing the same session s. Hence, we need to find another argument that avoids rewinding.

2 Definitions

The formalism and definitions we use to model key exchange protocols are adapted from de Saint Guilhelm et al. [dFW20]. Unlike the traditional Bellare–Rogaway [BR94, BR95, BPR00] and (e)CK models [CK01, LLM07], security in this model is not formulated as a single all-in-one game that implicitly captures

all the properties a protocol should have. Instead, security is split into many smaller definitions that each captures a single "atomic" security property. This leads to a slight increase in the number of definitions, as well as the number proofs one have to carry out in order to establish a protocol as "secure". On the other hand, the advantage of this approach is that each definition/property is much simpler and focused, and the corresponding proofs similarly simple.

2.1 Syntax

A *key exchange protocol* is a tuple of algorithms $(\mathsf{KeyGen}, \mathsf{Init}, \mathsf{Run})$ where KeyGen is the long-term key generation algorithm; Init creates a *session state* at party i having intended peer j and role role, and returns this session's initial message (empty if a responder role); and Run takes as input a session state st and a message m and outputs an updated state st' and response message m'.

Session state. A *session state* st consists of the following variables.

- $\mathsf{accept} \in \{\mathsf{true}, \mathsf{false}, \perp\}$ – indicates the status of the key exchange run; initialized to $\perp$ and indicates a running, non-completed, session.
- $\mathsf{key} \in \{0,1\}^* \cup \{\perp\}$ – the local session key derived during the key exchange run; set once $\mathsf{accept} = \mathsf{true}$.
- $\mathsf{role} \in \{\mathsf{init}, \mathsf{resp}\}$ – the role of the session in the key exchange run.
- party – the party identity to which this session belongs.
- peer – the party identity of the intended peer for this key exchange run.
- sk – the secret long-term key of the party this session belongs to.
- pk – the public long-term key of the intended peer of the session.
- $\mathsf{transcript}$ – the (ordered) transcript of all messages sent and received by session s. We use $\mathsf{transcript}^-$ to denote the transcript minus the last message.
- aux – auxiliary protocol specific state, such as internal randomness and ephemeral values.

Security experiment We shall use the generic formal experiment $\mathbf{Exp}^{\mathsf{Pred}}_{\Pi,U}(\mathcal{A})$ given in Fig. 1 to define the various security properties of a key exchange protocol (see Section 3). The experiment is parameterized on a *security predicate* Pred that captures the security property being modeled. The experiment uses a number of counters, variables and collections for bookkeeping purposes.

- $\mathsf{query\_ctr}$ – incremented for each query made by the adversary. Used to order events in time; needed to define (full) forward secrecy.

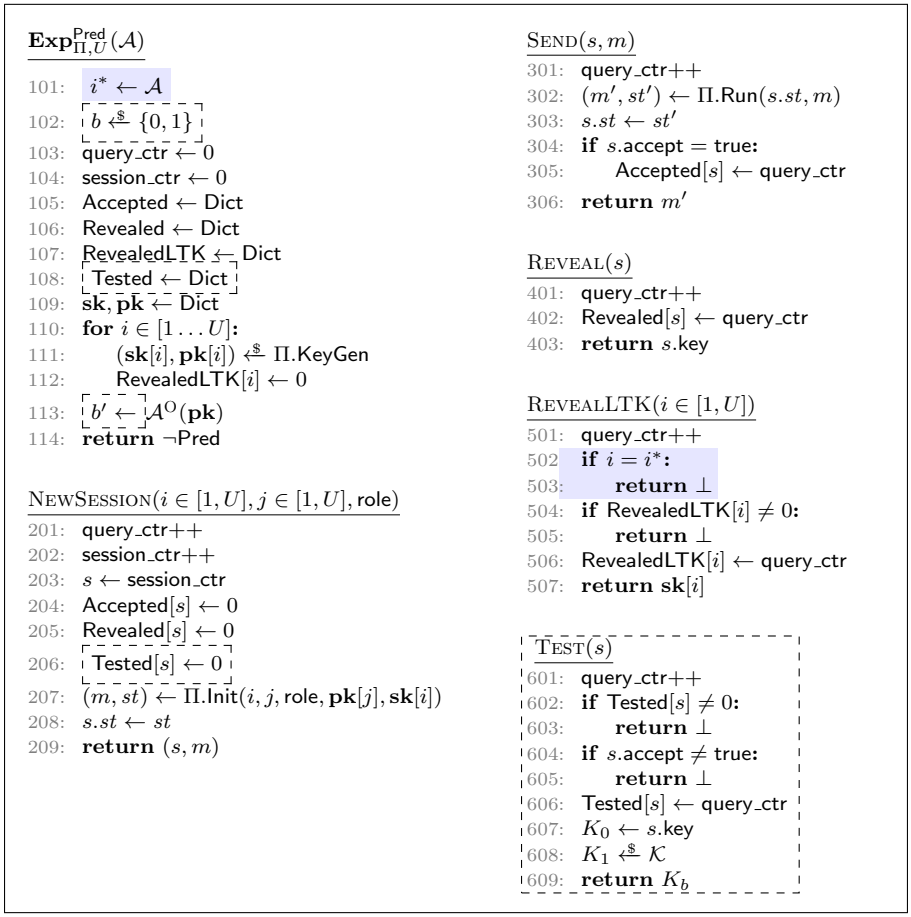

Figure 1: Generic experiment parameterized on predicate Pred, where $\mathcal{A}$ can make the queries in $\mathrm{O} = \{\textsc{NewSession}, \textsc{Send}, \textsc{Reveal}, \textsc{RevealLTK}, \textsc{Test}\}$. Code in dashed boxes is only for the key secrecy game; code in filled boxes is only for the selective key secrecy game. The notation $s.st \leftarrow st'$ means to assign all the variables in st' to the corresponding variables associated with session s. Dict defines an associative array.

- session_ctr – incremented for each new session created. Each session state is associated with a unique session number which functions as an administrative label for that session (state). The session number is also given to the adversary which can use it as an opaque handle to refer to a given session in its queries. We use the notation "s.x" to refer to the variable x of the session state identified by the administrative session number s. Note that the adversary cannot "dereference" a session number in order

to obtain internal variables of the session state.

- Accepted, Tested, Revealed, RevealedLTK – associative arrays that records when a session accepted, was tested, or when its session or long-term key was revealed.

Common predicates. It will be useful to introduce a number of predicates on the security experiment.

Definition 2.1 (Origin sessions)**.** A (possibly non-accepted) session s' is an *origin-session* for an accepted session s if predicate $\mathsf{Orig}(s, s')$ holds true, where

$$\mathsf{Orig}(s, s') \iff s'.\mathsf{transcript} \in \{s.\mathsf{transcript}, s.\mathsf{transcript}^{-}\}. \tag{1}$$

Definition 2.2 (Partnering)**.** Two sessions s, s' are *partners* if they have matching conversations; that is, if the predicate $\mathsf{Partner}(s, s')$ holds true, where

$$\mathsf{Partner}(s, s') \iff s.\mathsf{transcript} = s'.\mathsf{transcript}. \tag{2}$$

Like [dFW20] we do not require partners to agree upon each other's identities. This is an authentication property which will be covered by other definitions in Section 3. Unlike [dFW20] we use matching conversations instead of abstract session identifiers as our partnering mechanism. This is mainly done for the sake of concreteness and is not a fundamental difference, although certain well-known pitfalls need to be avoided when using matching conversations [LS17].

Definition 2.3 (SameKey)**.** The predicate $\mathsf{SameKey}(s, s')$ holds true if the sessions both have established a session key and they are equal, that is

$$\mathsf{SameKey}(s, s') \iff [s.\mathsf{key} = s'.\mathsf{key} \neq \bot]. \tag{3}$$

Definition 2.4 (Authentication fresh)**.** A session is *authentication fresh* if the long-term key of its intended peer has not been revealed, that is:

$$\mathsf{aFresh}(s) \iff \mathsf{RevealedLTK}[s.\mathsf{peer}] = 0. \tag{4}$$

Finally, we define freshness predicates used for the key secrecy games. These come in two flavors: *weak forward secrecy* and *full forward secrecy* [BPR00]. Common to both is that the adversary cannot reveal the session key of a tested session or its partner. The difference is how long-term key leakage is handled. For weak forward secrecy the adversary is forbidden from revealing the long-term key of a session's peer if it was actively interfering in the protocol run of the session (indicated by the lack of an origin-session for the session in question). For full forward secrecy this restriction is lifted, provided the leak happened *after* the session in question accepted.

Definition 2.5 (Session key freshness)**.** Let $s.\mathsf{peer} = j$. The $\mathsf{kFreshWFS}(s)$ (resp. $\mathsf{kFreshFFS}(s)$) predicate hold if:

$$\mathsf{Revealed}[s] = 0 \tag{5}$$

$$\forall s' :: \mathsf{Partner}(s, s') \implies \mathsf{Revealed}[s'] = 0 \wedge \mathsf{Tested}[s'] = 0 \tag{6}$$

$$\text{(wFS)} \quad \{s' \mid \mathsf{Orig}(s, s')\} = \emptyset \implies \mathsf{aFresh}(s) \tag{7}$$

$$\text{(fFS)} \quad \{s' \mid \mathsf{Orig}(s, s')\} = \emptyset \implies \mathsf{aFresh}(s) \vee (\mathsf{RevealedLTK}[j] > \mathsf{Accepted}[s]) \tag{8}$$

3 Protocol security properties

This section defines the security properties a secure key exchange protocol ought to have. The breakdown follows that of [dFW20] and consists of: soundness properties (match and key-match soundness); various authentication properties (implicit/explicit key and entity authentication); and session key secrecy. An application will typically require all of these properties. Refer to [dFW20] for further discussion and background.

3.1 Match soundness

Match soundness is primarily a sanity check on the choice of partnering mechanism. Namely, partnered sessions should derive the same session key (9); and sessions will at most have one partner (10).

Definition 3.1 (Match soundness)**.** The Match predicate evaluates to 1 iff $\forall s, s', s''$:

$$\mathsf{Partner}(s, s') \implies \mathsf{SameKey}(s, s') \tag{9}$$

$$(\mathsf{Partner}(s, s') \wedge \mathsf{Partner}(s, s'')) \implies s' = s'' \tag{10}$$

The *match soundness advantage* of an adversary $\mathcal{A}$ is

$$\mathbf{Adv}^{\mathsf{Match}}_{\Pi,U}(\mathcal{A}) \stackrel{\text{def}}{=} \Pr[\mathbf{Exp}^{\mathsf{Match}}_{\Pi,U}(\mathcal{A}) \Rightarrow 1] \tag{11}$$

3.2 Key-match soundness

Key-match soundness (KMSound) is basically the converse of Match soundness. While Match soundness says that partners should have equal session keys, KMSound says that sessions having equal session keys should be partners.

Definition 3.2 (Key-match soundness)**.** The KMSound predicate evaluates to 1 if and only if

$$\forall s :: (\mathsf{aFresh}(s) \wedge s.\mathsf{accept}) \implies \forall s' :: (\mathsf{SameKey}(s, s') \implies \mathsf{Partner}(s, s')) \tag{12}$$

The *key-match soundness advantage* of an adversary $\mathcal{A}$ is

$$\mathbf{Adv}^{\mathsf{KMSound}}_{\Pi,U}(\mathcal{A}) \stackrel{\text{def}}{=} \Pr[\mathbf{Exp}^{\mathsf{KMSound}}_{\Pi,U}(\mathcal{A}) \Rightarrow 1]. \tag{13}$$

3.3 Implicit key authentication

Implicit key authentication stipulates that two sessions that derive the same session key should agree upon *whom* they are sharing this key with.

Definition 3.3 (Implicit key authentication)**.** The iKeyAuth predicate evaluates to 1 if and only if

$$\forall s :: s.\mathsf{accept} \implies \forall s' :: (\mathsf{SameKey}(s, s') \implies s.\mathsf{peer} = s'.\mathsf{party})$$

The *implicit key authentication advantage* of an adversary $\mathcal{A}$ is

$$\mathbf{Adv}^{\mathsf{iKeyAuth}}_{\Pi,U}(\mathcal{A}) \stackrel{\text{def}}{=} \Pr[\mathbf{Exp}^{\mathsf{iKeyAuth}}_{\Pi,U}(\mathcal{A}) \Rightarrow 1]. \tag{14}$$

3.4 Explicit key authentication

Explicit key authentication stipulates that any two sessions that derive the same session key should agree upon whom they are sharing this key with (as for implicit key authentication), and as long as the session is authentication fresh some other session deriving the same session key should exist.

Obviously, the session that sends the last message can never guarantee that this message arrives at its destination, which means that this session can only achieve the notion of *almost-full* key authentication, namely that an origin session should exist and any origin session that has derived a session key has derived the same key. A session that receives the last message, however, can guarantee that another session exists that has derived the same key, and thereby achieve *full* key authentication.

Let $\mathcal{L}_{\mathsf{rcv}}$ denote the collection of all sessions that *receives* the last message of the protocol, and let $\mathcal{L}_{\mathsf{send}}$ denote the collection of all sessions that *sends* the last message of the protocol.

Definition 3.4 (Explicit key authentication)**.** The fexKeyAuth predicate (resp. afexKeyAuth predicate) evaluates to 1 if and only if

$$\forall s \in \mathcal{L}_{\mathsf{rcv}}\,(\text{resp. } \mathcal{L}_{\mathsf{send}}) :: s.\mathsf{accept} \implies \forall s' :: (\mathsf{SameKey}(s, s') \Rightarrow s.\mathsf{peer} = s'.\mathsf{party})$$

$$\wedge$$

$$\text{(full)} \qquad \mathsf{aFresh}(s) \Rightarrow \exists s' :: \mathsf{SameKey}(s, s')$$

$$\text{(almost-full)} \qquad \mathsf{aFresh}(s) \Rightarrow \exists s' :: \Big(\mathsf{Orig}(s, s') \wedge [s'.\mathsf{key} \neq \bot \implies \mathsf{SameKey}(s, s')]\Big)$$

The *full (resp. almost-full) explicit key authentication advantage* of $\mathcal{A}$ is

$$\mathbf{Adv}^{\mathsf{fexKeyAuth}}_{\Pi,U}(\mathcal{A}) \stackrel{\text{def}}{=} \Pr[\mathbf{Exp}^{\mathsf{fexKeyAuth}}_{\Pi,U}(\mathcal{A}) \Rightarrow 1] \tag{15}$$

$$\mathbf{Adv}^{\mathsf{afexKeyAuth}}_{\Pi,U}(\mathcal{A}) \stackrel{\text{def}}{=} \Pr[\mathbf{Exp}^{\mathsf{afexKeyAuth}}_{\Pi,U}(\mathcal{A}) \Rightarrow 1] \tag{16}$$

3.5 Explicit entity authentication

Explicit *entity* authentication is almost identical to explicit *key* authentication, the only difference being that the former is based on the Partner predicate while the latter is based on the SameKey predicate. Basically, explicit key authentication says that if a session with an honest peer accepts then there *is* some other session holding the same session key, while explicit entity authentication says that if a session with an honest peer accepts then it *has* a partner session.

Explicit key authentication and explicit entity authentication are closely related, as shown in [dFW20] and further expounded in Appendix A.2.

Definition 3.5 (Explicit entity authentication)**.** The fexEntAuth predicate (resp. afexEntAuth predicate) evaluates to 1 if and only if

$$\forall s \in \mathcal{L}_{\mathsf{rcv}}\,(\text{resp. } \mathcal{L}_{\mathsf{send}}) :: s.\mathsf{accept} \implies \forall s' ::(\mathsf{Partner}(s,s') \implies s.\mathsf{peer} = s'.\mathsf{party}) \wedge \mathsf{aFresh}(s) \implies \exists s' :: \mathsf{Partner}(s,s')$$

(full)

$$\text{(almost-full)} \quad \mathsf{aFresh}(s) \implies \exists s' :: \Big(\mathsf{Orig}(s,s') \wedge [s'.\mathsf{accept} \implies \mathsf{Partner}(s,s')]\Big)$$

The *full (resp. almost-full) explicit entity authentication advantage* of $\mathcal{A}$ is

$$\mathbf{Adv}^{\mathsf{fexEntAuth}}_{\Pi,U}(\mathcal{A}) \stackrel{\text{def}}{=} \Pr[\mathbf{Exp}^{\mathsf{fexEntAuth}}_{\Pi,U}(\mathcal{A}) \Rightarrow 1] \tag{17}$$

$$\mathbf{Adv}^{\mathsf{afexEntAuth}}_{\Pi,U}(\mathcal{A}) \stackrel{\text{def}}{=} \Pr[\mathbf{Exp}^{\mathsf{afexEntAuth}}_{\Pi,U}(\mathcal{A}) \Rightarrow 1] \tag{18}$$

3.6 Key secrecy

Key secrecy is defined as usual with the adversary using a Test query to get the real session key or a random key of a session. The adversary may make multiple test queries, and they all share the same challenge bit, so that either all Test queries return real keys, or all Test queries return random (and independently) sampled keys. Our experiment does not prevent the adversary from making Test queries for sessions that are not key fresh, so we need to account for this in the definition of advantage (called the *penalty-style* in [RZ18]).

Definition 3.6 (Key secrecy)**.** If $\forall s \in \mathsf{Tested} :: \mathsf{kFreshWFS}(s) = \mathsf{true}$ (resp. $\mathsf{kFreshFFS}(s) = \mathsf{true}$), the KeySecWFS (resp. KeySecFFS) predicate returns 1 if and only if $b' = b$. Else it returns b. The *weak (resp. full) forward key secrecy advantage* of an adversary $\mathcal{A}$ is

$$\mathbf{Adv}^{\mathsf{KeySecWFS}}_{\Pi,U}(\mathcal{A}) \stackrel{\text{def}}{=} \Big|2 \cdot \Pr[\mathbf{Exp}^{\mathsf{KeySecWFS}}_{\Pi,U}(\mathcal{A}) \Rightarrow 1] - 1\Big| \tag{19}$$

$$\mathbf{Adv}^{\mathsf{KeySecFFS}}_{\Pi,U}(\mathcal{A}) \stackrel{\text{def}}{=} \Big|2 \cdot \Pr[\mathbf{Exp}^{\mathsf{KeySecFFS}}_{\Pi,U}(\mathcal{A}) \Rightarrow 1] - 1\Big| \tag{20}$$

Selective key secrecy. The *selective* key secrecy experiment is defined over the experiment given in Fig. 1, where now the code inside the blue boxes is included. In the selective security experiment the adversary has to commit to one party it will not reveal the long-term key of throughout the game.

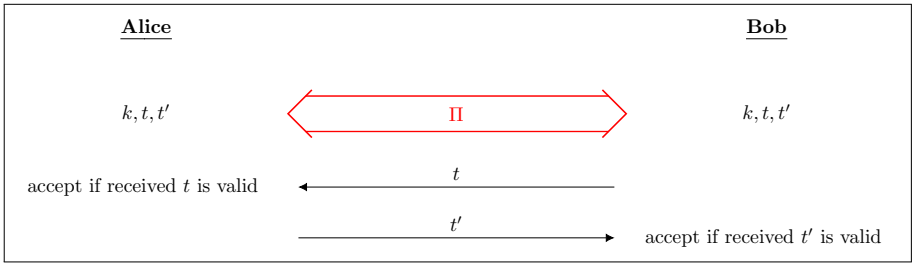

Figure 2: Protocol Π^+ obtained by extending protocol Π with key confirmation tags. All session variables in Π^+ are inherited from Π, except for accept which is defined as shown. The session sending the last message in protocol Π sends tag t, and the session receiving the last message in protocol Π sends tag t'.

Definition 3.7 (Selective key secrecy)**.** If $\forall s \in \mathsf{Tested} :: \mathsf{kFreshWFS}(s) = \mathsf{true}$, the SelKeySecWFS predicate returns 1 if and only if $b' = b$. Else it returns b. The *selective key secrecy advantage* of an adversary $\mathcal{A}$ is

$$\mathbf{Adv}^{\mathsf{SelKeySecWFS}}_{\Pi,U}(\mathcal{A}) \stackrel{\text{def}}{=} \left| 2 \cdot \Pr[\mathbf{Exp}^{\mathsf{SelKeySecWFS}}_{\Pi,U}(\mathcal{A}) \Rightarrow 1] - 1 \right| . \tag{21}$$

Remark 3.8. Contrary to what one might expect, ordinary key secrecy does *not* trivially reduce to selective key secrecy via a standard guessing argument (with a tightness loss of U). In particular, an adversary that starts by revealing *all* long-term keys will make a reduction to selective key secrecy unable to simulate the one key it committed to. This makes our selective security notion incomparable to the selective notion of [KPW13].

4 The security of adding key confirmation

Let Π denote an arbitrary key exchange protocol, and let Π^+ denote the protocol that extends Π by adding key confirmation messages from each side as illustrated in Fig. 2. Conventionally, the key confirmation messages are derived from the session key of Π using a PRF (and possibly a MAC) but in order to simplify the later analysis we assume that Π produces session keys of the form (k, t, t') directly. Protocol Π^+ is then derived from Π simply by defining its session key to be k, and the key confirmation tags to be t and t'. Using this trick we can relate the security of protocol Π^+ purely to the security of protocol Π without having to rely on PRFs or MACs.

Unfortunately, defining Π^+ in terms of the key triple output by Π introduces one technicality. We will often want to make an assertion of the form "if s and s' have equal keys in protocol Π^+ (meaning k), then they also have equal keys in protocol Π (meaning (k, t, t'))". While this assertion easily follows in practice[1], in the generality we have presented Π and Π^+ above the assertion

[1] For example if (k, t, t') is derived from the session transcript using a function for which

does not automatically follow. To cleanly state and prove our generic results we therefore introduce the implication "equal $k \implies$ equal (k, t, t')" as an explicit security property.

To this end, let $\mathrm{prefix} : \{0,1\}^* \to \{0,1\}^*$ be a function that returns a prefix of a particular length (left unspecified) from its argument and define

$$\mathsf{SamePrefix}(s, s') \iff [s.\mathsf{key}, s'.\mathsf{key} \neq \bot \wedge \mathrm{prefix}(s.\mathsf{key}) = \mathrm{prefix}(s'.\mathsf{key})]. \quad (22)$$

Definition 4.1 (Same prefix security)**.** The PreEqAllEq predicate evaluates to 1 if and only if

$$\forall s, s' : \mathsf{SamePrefix}(s, s') \implies \mathsf{SameKey}(s, s'). \quad (23)$$

The *same prefix advantage* of $\mathcal{A}$ is

$$\mathbf{Adv}^{\mathsf{PreEqAllEq}}_{\Pi,U}(\mathcal{A}) \stackrel{\mathrm{def}}{=} \Pr[\mathbf{Exp}^{\mathsf{PreEqAllEq}}_{\Pi,U}(\mathcal{A}) \Rightarrow 1]. \quad (24)$$

Remark 4.2. As mentioned above, proving same prefix security for a concrete protocol will typically be straightforward assuming the keys are derived using a reasonable function. However, it is also possible to avoid the notion altogether (even for completely generic protocols) using the proof technique of Lemma 4.6 in Section 4.2. Specifically, if s and s' have equal keys in protocol Π^+ but not in protocol Π, then this allows to break the (selective) key secrecy of protocol Π, albeit with a tightness loss in the number of parties U.

4.1 Main result

We now state the first main theorem of the paper: a protocol with weak forward secrecy can be upgraded to full forward secrecy by adding key confirmation messages with a linear security loss in the number of parties. The second main theorem of the paper is that this linear security loss is unavoidable for a larger class of compilers (see Section 6).

Theorem 4.3. *Let $\mathcal{A}$ be an adversary against key secrecy for Π^+. Then there exist adversaries $\mathcal{B}_1, \mathcal{B}_2, \ldots, \mathcal{B}_6$, all with about the same runtime as $\mathcal{A}$, such that*

$$\begin{aligned}\mathbf{Adv}^{\mathsf{KeySecFFS}}_{\Pi^+,U}(\mathcal{A}) \leq\ & 4 \cdot U \cdot \mathbf{Adv}^{\mathsf{SelKeySecWFS}}_{\Pi,U}(\mathcal{B}_1) + 8 \cdot \mathbf{Adv}^{\mathsf{iKeyAuth}}_{\Pi,U}(\mathcal{B}_2) + \frac{4US}{2^{\mathsf{taglen}}} \\ & + 4 \cdot \mathbf{Adv}^{\mathsf{Match}}_{\Pi,U}(\mathcal{B}_3) + 4 \cdot \mathbf{Adv}^{\mathsf{KMSound}}_{\Pi,U}(\mathcal{B}_4) \\ & + 12 \cdot \mathbf{Adv}^{\mathsf{PreEqAllEq}}_{\Pi,U}(\mathcal{B}_5) + 4 \cdot \mathbf{Adv}^{\mathsf{KeySecWFS}}_{\Pi,U}(\mathcal{B}_6),\end{aligned}$$

where taglen *is the length of the key confirmation tags used by Π^+ and S is the number of sessions.*

The outline of the proof of Theorem 4.3 is shown in Fig. 3. At a high level the proof consists of two parts: one where all accepting sessions with peers

getting a collision just in k is unlikely, such as an extendable-output function or a random oracle

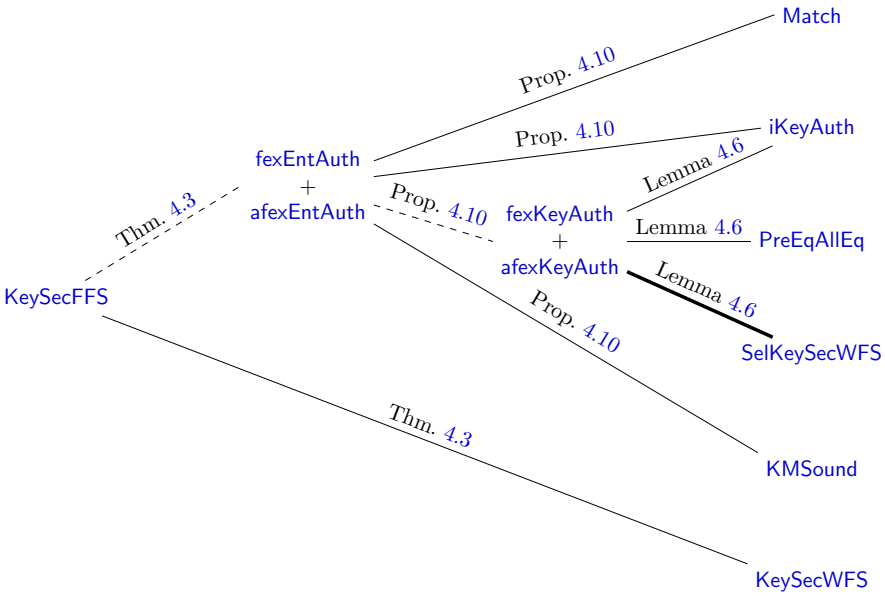

Figure 3: Steps in the proof of Theorem 4.3. A dashed line from X to Y means that Y is not part of the statement of X but only an intermediate step inside its proof. A bold line from X to Y means that the reduction loses a factor of U. Note that some reductions include additional intermediate steps not shown.

whose long-term keys are unrevealed have an origin session, and one where they don't. In the first case full forward key secrecy of protocol Π^+ reduces straightforwardly to the weak forward key secrecy of protocol Π. The main challenge is to deal with the second case, namely to prove that protocol Π^+ achieves explicit entity authentication. In fact, the main technical tool for this is to prove that Π^+ achieves explicit *key* authentication, which is where we use the *selective* key secrecy notion. The proof of explicit key authentication is the focus of Section 4.2.

Proof (of Theorem 4.3). We prove that

$$\mathbf{Adv}^{\mathsf{KeySecFFS}}_{\Pi^+,U}(\mathcal{A}) \leq 2 \cdot \mathbf{Adv}^{\mathsf{fexEntAuth}}_{\Pi^+,U}(\mathcal{A}) + 2 \cdot \mathbf{Adv}^{\mathsf{afexEntAuth}}_{\Pi^+,U}(\mathcal{A}) + 4 \cdot \mathbf{Adv}^{\mathsf{KeySecWFS}}_{\Pi,U}(\mathcal{B}_6) \tag{25}$$

from which the result will follow by Proposition 4.10.

Let Win_{G_i} denote that $\mathcal{A}$ wins in Game i, i.e., $b' = b$ and all TEST sessions are key fresh.

Game 0. This is the original key secrecy game for protocol Π^+.

Game 1. In this game all sessions s having honest peers (i.e., $\mathsf{aFresh}(s) = \mathsf{true}$), but not having an origin session reject all tags. In particular, this means that these sessions will never accept in protocol Π^+.

Claim 4.4.

$$\Pr[\mathsf{Win}_{G_0}] \leq \Pr[\mathsf{Win}_{G_1}] + \mathbf{Adv}^{\mathsf{fexEntAuth}}_{\Pi^+,U}(\mathcal{A}) + \mathbf{Adv}^{\mathsf{afexEntAuth}}_{\Pi^+,U}(\mathcal{A}) \tag{26}$$

Proof. Let E be the event that a session with an honest peer accepts in protocol Π^+, but without having an origin session. Since Game 0 and Game 1 are identical unless event E occurs it is sufficient to bound $\Pr[E]$.

Suppose s is a session that triggers event E. In particular, (i) $s.\mathsf{accept} = \mathsf{true}$, (ii) $\mathsf{aFresh}(s) = \mathsf{true}$, and (iii) $\{s' \mid \mathsf{Orig}(s', s)\} = \emptyset$. By the last property s also doesn't have a partner, and by the first two properties this means that either fexEntAuth is violated (in case $s \in \mathcal{L}_{\mathsf{rcv}}$) or afexEntAuth is violated (in case $s \in \mathcal{L}_{\mathsf{send}}$). □

Game 2. In this game all kFreshWFS sessions *which accept* have their session keys replaced by random.

Claim 4.5.

$$\Pr[\mathsf{Win}_{G_1}] \leq \Pr[\mathsf{Win}_{G_2}] + 2 \cdot \mathbf{Adv}^{\mathsf{KeySecWFS}}_{\Pi,U}(\mathcal{B}_6). \tag{27}$$

Proof. Algorithm $\mathcal{B}_6$ begins by drawing a random bit b_{sim}, then simulates the following game for $\mathcal{A}$. Whenever $\mathcal{A}$ makes a query pertaining to protocol Π, then $\mathcal{B}_6$ forwards it to its own (weak forward secrecy) game. When a session s accepts in protocol Π, then $\mathcal{B}_6$ simulates protocol Π^+ for $\mathcal{A}$ as follows.

- If s is not kFreshWFS in protocol Π then $\mathcal{B}_6$ issues a $\textsc{Reveal}(s)$ query to its key secrecy game and uses the returned key (k, t, t') to simulate s's tags in protocol Π^+.
- If s is kFreshWFS, but does not have an origin session (in Π), then $\mathcal{B}_6$ issues a $\textsc{Reveal}(s)$ query to its key secrecy game and uses the returned key (k, t, t') to simulate the tag *sent* by s. If s *receives* a tag, then it is simply rejected.
- If s is kFreshWFS and has an origin session s' (in Π), then,
  - if s' haven't accepted in Π yet (and thus haven't been issued a $\textsc{Test}$ or $\textsc{Reveal}$ query), then $\mathcal{B}_6$ issues a $\textsc{Test}$ query to s to obtain a key (k, t, t');
  - if, on the other hand, s' have already accepted, then by the previous cases $\mathcal{B}_6$ must already have issued a $\textsc{Test}$ or $\textsc{Reveal}$ query to s', which returned a key (k, t, t').

  In either case, $\mathcal{B}_6$ uses the returned key (k, t, t') to simulate both how s sends *and* receives tags.

To answer $\mathcal{A}$'s REVEAL queries (in Π^+), $\mathcal{B}_6$ uses the "k" element of the tuples it obtained above. To answer $\mathcal{A}$'s REVEALLTK queries, $\mathcal{B}_6$ simply forward these to its own game. To answer $\mathcal{A}$'s TEST queries, $\mathcal{B}_6$ answers as follows.

- If the session has already been tested, or has not accepted yet, return $\perp$.
- If $b_{sim} = 0$ then $\mathcal{B}_6$ returns key from the (k, t, t') tuple it previously obtained for this session, as described above.
- If $b_{sim} = 1$ then $\mathcal{B}_6$ returns a random key $\widetilde{k}$.

Finally, when $\mathcal{A}$ stops and outputs a bit b', then $\mathcal{B}_6$ outputs 1 to its own key secrecy game if and only if $b' = b_{sim}$.[2]

We first claim that if the secret bit in $\mathcal{B}_6$'s key secrecy game is 0 (hence $\mathcal{B}_6$'s TEST queries are answered with real session keys), then $\mathcal{B}_6$ perfectly simulates either Game 1 or Game 2, depending on the bit b_{sim}.

Note first that if s is not kFreshWFS, then $\mathcal{B}_6$ obtains s's actual key in protocol Π, hence simulates protocol Π^+ correctly. Second, if s is kFreshWFS, but does not have an origin session, then s rejects any tag, and thus never accepts in protocol Π^+. This is exactly what happens after the change in Game 1. Finally, if s is kFreshWFS and has an origin session, then it behaves exactly as in Game 1 if $b_{sim} = 0$ (since then $\mathcal{B}_6$ is using the actual key from Π) and exactly as in Game 2 if $b_{sim} = 1$ (since then $\mathcal{B}_6$ is using a completely random key).

On the other hand, if the secret bit in $\mathcal{B}_6$'s key secrecy game is 1, then $\mathcal{B}_6$ simulates Game 2 independently of what b_{sim} is. Specifically, this means that b_{sim} is completely hidden from $\mathcal{A}$ in this case.

Consequently, assuming the secret bit in $\mathcal{B}_6$'s key secrecy game is b, we have

$$\begin{aligned}
\Pr[\mathcal{B}_6 \text{ wins}] &= \Pr[\mathcal{B}_6 \text{ wins} \mid b = 0 \wedge b_{sim} = 0] \cdot \frac{1}{4} \\
&\quad + \Pr[\mathcal{B}_6 \text{ wins} \mid b = 0 \wedge b_{sim} = 1] \cdot \frac{1}{4} && (28) \\
&\quad + \Pr[\mathcal{B}_6 \text{ wins} \mid b = 1] \cdot \frac{1}{2} \\
&= \Pr[\mathsf{Win}_{G_1}] \cdot \frac{1}{4} + (1 - \Pr[\mathsf{Win}_{G_2}]) \cdot \frac{1}{4} + \frac{1}{4} && (29) \\
&= \Pr[\mathsf{Win}_{G_1}] \cdot \frac{1}{4} - \Pr[\mathsf{Win}_{G_2}] \cdot \frac{1}{4} + \frac{1}{2}. && (30)
\end{aligned}$$

Hence,

$$\mathbf{Adv}^{\mathsf{KeySecWFS}}_{\Pi,U}(\mathcal{B}_6) = |2 \cdot \Pr[\mathcal{B}_6 \text{ wins}] - 1| = \frac{1}{2} \cdot |\Pr[\mathsf{Win}_{G_1}] - \Pr[\mathsf{Win}_{G_2}]| \quad (31)$$

which proves the claim. □

[2] Assuming all tested sessions are key fresh (according to kFreshFFS). Otherwise $\mathcal{B}_6$ simply outputs a random bit.

Concluding the proof of Theorem 4.3. By the change in Game 2 all session keys of kFreshWFS sessions are now random, hence $\Pr[\mathsf{Win}_{G_2}] = 1/2$. Combining the bounds from (26) and (27), and multiplying by 2, yields (25), from which Theorem 4.3 follows by Proposition 4.10. □

4.2 Implicit to explicit key authentication

In this section, we establish that explicit key authentication can be based on *selective* key secrecy, implicit key authentication, and same prefix security. This is a key technical result needed to restore the tight security of the explicitly authenticated protocol of [CCG+19]. The use of selective security may also have further applications in constructing highly efficient explicitly authenticated key exchange protocols with full forward secrecy in the future.

Lemma 4.6. *Let $\mathcal{A}$ be an adversary against full (resp. almost full) explicit key authentication for Π^+. Then there exists an adversary $\mathcal{B}_2$ against selective key secrecy and an adversary $\mathcal{B}_1$ against implicit key authentication and same prefix security, both with the same runtime as $\mathcal{A}$, such that*

$$\mathbf{Adv}^{\mathsf{fexKeyAuth}}_{\Pi^+,U}(\mathcal{A}) \leq \mathbf{Adv}^{\mathsf{iKeyAuth}}_{\Pi,U}(\mathcal{B}_1)+\mathbf{Adv}^{\mathsf{PreEqAllEq}}_{\Pi,U}(\mathcal{B}_1)+U\cdot\mathbf{Adv}^{\mathsf{SelKeySecWFS}}_{\Pi,U}(\mathcal{B}_2)+\frac{US}{2^{\mathsf{taglen}}},$$

$$\mathbf{Adv}^{\mathsf{afexKeyAuth}}_{\Pi^+,U}(\mathcal{A}) \leq \mathbf{Adv}^{\mathsf{iKeyAuth}}_{\Pi,U}(\mathcal{B}_1)+\mathbf{Adv}^{\mathsf{PreEqAllEq}}_{\Pi,U}(\mathcal{B}_1)+U\cdot\mathbf{Adv}^{\mathsf{SelKeySecWFS}}_{\Pi,U}(\mathcal{B}_2)+\frac{US}{2^{\mathsf{taglen}}},$$

where taglen *is the length of the key confirmation tags used by Π^+ and S is the number of sessions.*

Since the proofs of full and almost full key authentication are virtually identical, we only prove the fist bound. We need to deal with two cases. The first case considers attacks on explicit authentication that result from breaking implicit authentication of the underlying protocol Π. This case does not incur a tightness loss.

The second case considers attacks on explicit authentication that rely on breaking the weak forward secrecy of the underlying protocol Π. The important point is that in order to break explicit authentication, the partner long-term key must be unrevealed at the point in time where authentication is broken. This means that the session will be fresh at the time authentication is broken, which means that we can deduce the challenge bit at the point in time where authentication is broken. Any subsequent reveal of the partner long-term key can therefore be ignored.

Proof. The proof is structured as a sequence of games. Let Win_{G_i} denote the event that $\mathcal{A}$ wins in Game i. Winning in this case means that full explicit key authentication in (15) from Definition 3.4 does not hold.

Game 0. This is the original game for protocol Π^+. We have that

$$\mathbf{Adv}^{\mathsf{fexKeyAuth}}_{\Pi^+,U}(\mathcal{A}) = \Pr[\mathsf{Win}_{G_0}]. \tag{32}$$

Game 1. We modify the game so that if (15) does not hold for the Π part of a session of Π^+, then that session never accepts. Let Except_{G_1} be the event that this happens.

It is immediate that until Except_{G_1} happens, Game 1 proceeds exactly as Game 0, so

$$|\Pr[\mathsf{Win}_{G_1}] - \Pr[\mathsf{Win}_{G_0}]| \leq \Pr[\mathsf{Except}_{G_1}]. \tag{33}$$

We create an adversary $\mathcal{B}_1$ against implicit key authentication for Π that runs a copy of $\mathcal{A}$ and uses its experiment to run the Π part of Π^+. When a session of Π outputs a session key, $\mathcal{B}_1$ reveals the session key and uses that to simulate sending and receiving the key confirmation messages. Let $\mathsf{Win}_{\mathcal{B}_1}$ denote the probability that $\mathcal{B}_1$ wins.

It is immediate that $\mathcal{B}_1$ and its experiment together simulate the experiment in Game 0 perfectly with respect to the copy of $\mathcal{A}$ run by $\mathcal{B}_1$. Since SameKey for Π^+ implies SamePrefix for Π, if (15) does not hold for Π^+ in an execution, either it will not hold when we consider the game as an execution of Π, or PreEqAllEq will not hold when we consider the game as an execution of Π. In other words,

$$\Pr[\mathsf{Except}_{G_1}] \leq \Pr[\mathsf{Win}_{\mathcal{B}_1}^{\mathsf{iKeyAuth}}] + \Pr[\mathsf{Win}_{\mathcal{B}_1}^{\mathsf{PreEqAllEq}}]. \tag{34}$$

Game 2. We modify the game by sampling $j \in \{1, 2, \ldots, U\}$ at the start. Let Win'_{G_2} be the event that Win_{G_2} happens and one session for which authentication is broken has the jth key as its peer's public key. Clearly,

$$\Pr[\mathsf{Win}'_{G_2}] \geq \frac{1}{U} \Pr[\mathsf{Win}_{G_2}] = \frac{1}{U} \Pr[\mathsf{Win}_{G_1}]. \tag{35}$$

Game 3. We modify the game so that if (15) holds for a session of Π^+ that has the jth key as its peer public key but it has no origin session, then that session samples random tags to use for the Π^+ part of the protocol, instead of the tags output by Π.

It is immediate that

$$\Pr[\mathsf{Win}'_{G_3}] \leq \frac{S}{2^{\mathsf{taglen}}}. \tag{36}$$

We create an adversary $\mathcal{B}_2$ against selective key secrecy for Π that runs a copy of $\mathcal{A}$ and uses its experiment to run the Π part of Π^+, simulating the sending and receiving of key confirmation messages as modified in Game 2, further modified as follows:

- At the start, $\mathcal{B}_2$ selects an integer $j \in \{1, 2, \ldots, U\}$.
- When a session of Π, using the ith key as its peer key, outputs a session key, (15) holds for the session and it has no origin session, then:
  - If $i \neq j$, then $\mathcal{B}_2$ reveals the session key of the session and uses that key to simulate the Π^+ part of the session.
  - If $i = j$, then $\mathcal{B}_2$ tests the Π instance and uses that key to simulate the Π^+ part of the session.

- If $\mathcal{A}$ reveals the jth long-term key, $\mathcal{B}_2$ outputs 0 and stops.

If $\mathcal{A}$ breaks authentication for a session with the jth key as its peer key, $\mathcal{B}_2$ outputs 1, otherwise $\mathcal{B}_2$ outputs 0.

Let $\mathsf{Win}'_{\mathcal{B}_2,b}$ denote the event that $\mathcal{B}_2$ outputs 1, when its experiment has the secret bit b. We have that

$$\mathbf{Adv}^{\mathsf{SelKeySecWFS}}_{\Pi,U}(\mathcal{B}_2) = |\Pr[\mathsf{Win}_{\mathcal{B}_2,0}] - \Pr[\mathsf{Win}_{\mathcal{B}_2,1}]|. \tag{37}$$

If the experiment's secret bit $b = 0$, then $\mathcal{B}_2$ perfectly simulates Game 2 with respect to the Win'_{G_2} event, since the only observable difference is that $\mathcal{B}_2$ terminates when Win'_{G_2} no longer can occur (when the jth long-term key is revealed), so

$$\Pr[\mathsf{Win}_{\mathcal{B}_2,0}] = \Pr[\mathsf{Win}'_{G_2}]. \tag{38}$$

If the experiment's secret bit $b = 1$, then $\mathcal{B}_2$ perfectly simulates Game 3 with respect to the Win'_{G_3} event, again because of termination, so

$$\Pr[\mathsf{Win}_{\mathcal{B}_2,1}] = \Pr[\mathsf{Win}'_{G_3}]. \tag{39}$$

The claim follows from (32)–(39). □

4.3 Additional security reductions

Lemma 4.6 is *the* key result needed to show that protocol Π^+ achieves full forward secrecy (and explicit authentication) from the weakly forward-secret (and implicitly authenticated) protocol Π in a way that only incurs a tightness loss of U. From this result all the other security properties defined in Section 3 follow in a straightforward and modular way as we demonstrate in the following subsections. Moreover, none of these reductions lose more than a factor of U (in fact, most of the reductions are fully tight; those that are not only accrue the U term as a result of invoking Lemma 4.6).

Match soundness. Match soundness of protocol Π^+ follows directly from match soundness of Π.

Proposition 4.7. *Let $\mathcal{A}$ be an adversary against match security for Π^+. Then there exists an adversary $\mathcal{B}$ against match security for Π with the same runtime as $\mathcal{A}$, such that*

$$\mathbf{Adv}^{\mathsf{Match}}_{\Pi^+,U}(\mathcal{A}) \leq \mathbf{Adv}^{\mathsf{Match}}_{\Pi,U}(\mathcal{B}).$$

Proof. If any condition on the left hand side in Definition 3.1 (partnering via matching conversations) is satisfied for protocol Π^+ then it is also satisfied for protocol Π since the transcript of Π is a prefix of the transcript for Π^+. Hence any violation in Π^+ implies a violation in Π.

The adversary $\mathcal{B}$ against Π therefore trivially simulates an execution of Π^+ by revealing session keys and simulating the key confirmation messages. This simulation is perfect and does not require extra resources. □

Key-match soundness. Key-match soundness breaks down into two cases depending on whether or not s and s' have the same key in protocol Π. Here we again use the same-prefix security notion PreEqAllEq as in Lemma 4.6.

Proposition 4.8. *Let $\mathcal{A}$ be an adversary against key match soundness for Π^+. Then there exists an adversary $\mathcal{B}$ against key match soundness, match security and same-prefix security for Π with the same runtime as $\mathcal{A}$, such that*

$$\mathbf{Adv}^{\mathsf{KMSound}}_{\Pi^+,U}(\mathcal{A}) \leq \mathbf{Adv}^{\mathsf{KMSound}}_{\Pi,U}(\mathcal{B}) + \mathbf{Adv}^{\mathsf{PreEqAllEq}}_{\Pi,U}(\mathcal{B}).$$

Proof. Suppose s and s' are such that key-match soundness (Definition 3.2) is violated for protocol Π^+. That is, s has accepted, $\mathsf{aFresh}(s) = \mathsf{true}$, $\mathsf{SameKey}(s, s') = \mathsf{true}$, but $\mathsf{Partner}(s, s') = \mathsf{false}$. Note that all of these predicates are relative to protocol Π^+.

We consider two cases:

1. s and s' also have the same key (k, t, t') in protocol Π;
2. s and s' do not have the same key in protocol Π.

In case 1, s and s' have the same key confirmation tags t, t', hence for them not to be partners in Π^+, they cannot be partners in protocol Π (recall that partnering is based on matching conversations). But this then violates key-match soundness of Π. In case 2, PreEqAllEq fails for protocol Π. □

Implicit key authentication. The proof of implicit key authentication follows the exact same structure as the key-match soundness proof.

Proposition 4.9. *Let $\mathcal{A}$ be an adversary against implicit key authentication for Π^+. Then there exists an adversary $\mathcal{B}$ against implicit key authentication, match security and same-prefix security for Π with the same runtime as $\mathcal{A}$, such that*

$$\mathbf{Adv}^{\mathsf{iKeyAuth}}_{\Pi^+,U}(\mathcal{A}) \leq \mathbf{Adv}^{\mathsf{iKeyAuth}}_{\Pi,U}(\mathcal{B}) + \mathbf{Adv}^{\mathsf{PreEqAllEq}}_{\Pi,U}(\mathcal{B}).$$

Proof. Suppose s and s' are such that implicit key authentication (Definition 3.3) is violated for protocol Π^+. That is, s has accepted, $\mathsf{aFresh}(s) = \mathsf{true}$, $\mathsf{SameKey}(s, s') = \mathsf{true}$, but $s.\mathsf{peer} \neq s'.\mathsf{party}$. Note that all of these predicates are relative to protocol Π^+.

We consider two cases:

1. s and s' also have the same key (k, t, t') in protocol Π;
2. s and s' do not have the same key in protocol Π.

In case 1 implicit key authentication is also violated for protocol Π. In case 2, PreEqAllEq fails for protocol Π. □

Explicit entity authentication. Proving explicit entity authentication is straightforward, though it relies on a technical result from Appendix A.2.

Proposition 4.10. *Let $\mathcal{A}$ be an adversary against full (resp. almost-full) explicit entity authentication for Π^+. Then there exists adversaries $\mathcal{B}_1, \mathcal{B}_2, \ldots, \mathcal{B}_6$ against, respectively, match security, implicit key authentication, key match soundness, key secrecy, selective key secrecy and implicit key authentication for Π, all with the same runtime as $\mathcal{A}$, such that*

$$\begin{aligned}\mathbf{Adv}^{\mathsf{fexEntAuth}}_{\Pi^+,U}(\mathcal{A}) \leq \mathbf{Adv}^{\mathsf{Match}}_{\Pi,U}(\mathcal{B}_1) + 2 \cdot \mathbf{Adv}^{\mathsf{iKeyAuth}}_{\Pi,U}(\mathcal{B}_2) + \mathbf{Adv}^{\mathsf{KMSound}}_{\Pi,U}(\mathcal{B}_3) \\ + 3 \cdot \mathbf{Adv}^{\mathsf{PreEqAllEq}}_{\Pi,U}(\mathcal{B}_4) + \frac{US}{2^{\mathsf{taglen}}} + U \cdot \mathbf{Adv}^{\mathsf{SelKeySecWFS}}_{\Pi,U}(\mathcal{B}_5)\end{aligned}$$

$$\begin{aligned}\mathbf{Adv}^{\mathsf{afexEntAuth}}_{\Pi^+,U}(\mathcal{A}) \leq \mathbf{Adv}^{\mathsf{Match}}_{\Pi,U}(\mathcal{B}_1) + 2 \cdot \mathbf{Adv}^{\mathsf{iKeyAuth}}_{\Pi,U}(\mathcal{B}_2) + \mathbf{Adv}^{\mathsf{KMSound}}_{\Pi,U}(\mathcal{B}_3) \\ + 3 \cdot \mathbf{Adv}^{\mathsf{PreEqAllEq}}_{\Pi,U}(\mathcal{B}_4) + \frac{US}{2^{\mathsf{taglen}}} + U \cdot \mathbf{Adv}^{\mathsf{SelKeySecWFS}}_{\Pi,U}(\mathcal{B}_5)\end{aligned}$$

Proof. The proofs of full and almost-full explicit entity authentication are virtually identical so we only give a proof of the former.

By Proposition A.2 we have (note that Π^+ appears on both sides of the inequality)

$$\mathbf{Adv}^{\mathsf{fexEntAuth}}_{\Pi^+,U}(\mathcal{A}) \leq \mathbf{Adv}^{\mathsf{Match}}_{\Pi^+,U}(\mathcal{A}) + \mathbf{Adv}^{\mathsf{iKeyAuth}}_{\Pi^+,U}(\mathcal{A}) + \mathbf{Adv}^{\mathsf{KMSound}}_{\Pi^+,U}(\mathcal{A}) + \mathbf{Adv}^{\mathsf{fexKeyAuth}}_{\Pi^+,U}(\mathcal{A})$$

Proposition 4.10 now follows by bounding all the individual terms on the right using, respectively, Proposition 4.7, Proposition 4.9, Proposition 4.8, and Lemma 4.6. □

5 The CCGJJ protocol

The CCGJJ protocol [CCG+19], shown in Fig. 4, is a highly efficient implicitly authenticated key exchange protocol with optimal tightness. We use this protocol to illustrate our framework, which means we need to prove it satisfies the various security properties defined in Section 3.

We begin by proving that the protocol has the basic properties we want, in particular match soundness, key match soundness and the same prefix property.

Proposition 5.1. *Let $\mathcal{A}$ be an adversary against the CCGJJ protocol. Then*

$$\mathbf{Adv}^{\mathsf{Match}}_{\mathrm{CCGJJ},U}(\mathcal{A}) \leq \frac{S^2}{2^{|\mathsf{key}|}}, \quad \mathbf{Adv}^{\mathsf{KMSound}}_{\mathrm{CCGJJ},U}(\mathcal{A}) \leq \frac{S^2}{|\mathbb{G}|} \quad \textit{and} \quad \mathbf{Adv}^{\mathsf{iKeyAuth}}_{\mathrm{CCGJJ},U}(\mathcal{A}) \leq \frac{S^2}{2^{|\mathsf{key}|}}.$$

Proof. Since the KDF used to compute the session key includes the transcript and the secrets included are fully determined by the public values, it follows that any two sessions that are partners will compute the same session key, and a session will compute the same key as any origin session.

Match soundness could fail if a session has more than one partner, but this will only happen if two sessions choose the same randomness.

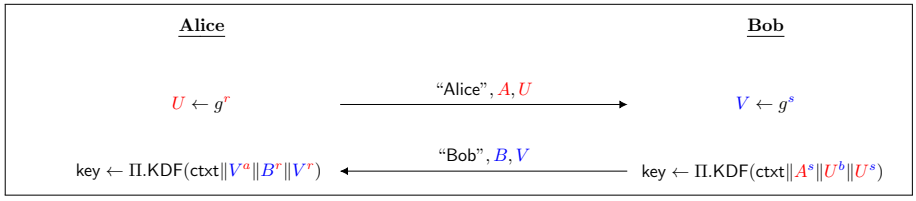

Figure 4: The CCGJJ protocol from [CCG+19] for a prime-ordered group $\mathbb{G}$ with generator g. Alice has secret long-term key a with public key $A \leftarrow g^a$; Bob has secret long-term key b with public key $B \leftarrow g^b$. Their context ctxt contains their names, their public keys and the two messages U and V. We include names and public keys in the messages; in practice these may be communicated in other ways.

Since the information in the transcript, and in particular names, is included in the KDF that computes the key, two sessions that compute the same session key must either be partners/agree on identities or there must be a collision in the KDF.

In either case, the claims follows from a birthday bound. □

Proposition 5.2. *Let $\mathcal{A}$ be an adversary against the CCGJJ protocol. Then*

$$\mathbf{Adv}^{\mathsf{PreEqAllEq}}_{\mathrm{CCGJJ},U}(\mathcal{A}) \leq \frac{S^2}{2^{|\mathsf{key}|-2\mathsf{taglen}}}.$$

Proof. Two sessions have identical prefixes but distinct keys only if the data hashed is distinct and there is a partial collision in the KDF. Since we model the KDF as a random oracle, the birthday bound applies. □

Proposition 5.3. *Let $\mathcal{A}$ be an adversary against selective key secrecy for CCGJJ. Then there exists adversaries $\mathcal{B}_1$, $\mathcal{B}_2$ and $\mathcal{B}_3$ against strong Diffie-Hellman (in group $\mathbb{G}$ with generator g) with essentially the same runtime as $\mathcal{A}$ such that*

$$\mathbf{Adv}^{\mathsf{SelKeySecWFS}}_{\mathrm{CCGJJ},U}(\mathcal{A}) \leq \mathbf{Adv}^{\mathsf{stDH}}_{\mathbb{G},g}(\mathcal{B}_1) + \mathbf{Adv}^{\mathsf{stDH}}_{\mathbb{G},g}(\mathcal{B}_2) + \mathbf{Adv}^{\mathsf{stDH}}_{\mathbb{G},g}(\mathcal{B}_3) + \frac{US^2}{|\mathbb{G}|}.$$

The proof of Proposition 5.3 closely follows the structure of the proof in [CCG+19], so we only sketch the argument, with emphasis on the differences, which entirely relate to avoiding hybrid arguments involving the users. (Also note that the proof in [CCG+19] proves more than just key secrecy, since they use a different security model.)

Proof (sketch). Note that all of the adversaries we construct below have the same runtime as $\mathcal{A}$ and the number of DH oracle calls is essentially bounded by the number of random oracle queries made by $\mathcal{A}$.

Recall from the proof in [CCG+19] that there are five classes of sessions: (I) initiator sessions that have an origin session/partner that it agrees on identities

with; (II) other initiator sessions that are authentication fresh when they accept; (III) responder sessions that have an origin session that it agrees on identities with; (IV) other responder sessions that are authentication fresh when they accept; and (V) sessions where the intended peer's long-term key has been revealed.

The proof proceeds as a sequence of games, where the initial (zeroth) game is the selective key secrecy game. The first game prevents two honest sessions from having the same randomness. The second game changes to lazy evaluation of the random oracle $\Pi.\mathsf{KDF}$. We use the birthday bound to bound the advantage loss from the first change, while the second change is unobservable.

The third game modifies type IV responder sessions whose peer is the selected party so that it never modifies the random oracle to be consistent with the session key. This is only observable if the adversary makes the corresponding hash query. By embedding our challenge DH tuple into the public key of the selected party and (rerandomized) into the message sent by type IV responder sessions, we get a strong Diffie-Hellman adversary $\mathcal{B}_1$ that succeeds whenever such a hash query is made. This adversary uses its DH oracle to recognize and reprogram hash queries related to sessions running as the selected party. *Unlike in [CCG+19], the strong Diffie-Hellman adversary does not have to guess a party whose secret long-term key will not be revealed, so it does not lose a factor U in advantage.*

The fourth game modifies type III responder sessions so that they never modify the random oracle to be consistent with the session key. This is only observable if the adversary makes the corresponding hash query. By embedding our challenge DH tuple (rerandomized) into the initial messages of type I and II initiator sessions and (rerandomized) into the responder messages of type III responder sessions, we get a strong Diffie-Hellman adversary $\mathcal{B}_2$ that succeeds whenever such a hash query is made. This adversary uses its DH oracle to recognize and reprogram hash queries related to type II initiator sessions.

The fifth game modifies type II initiator sessions whose peer is the selected party so that they never modify the random oracle to be consistent with the session key. This is only observable if the adversary makes the corresponding hash query. By embedding our challenge DH tuple into the public key of the selected party and (rerandomized) into the message sent by type I or II initiator sessions, we get a strong Diffie-Hellman adversary $\mathcal{B}_3$ that succeeds whenever such a hash query is made. This adversary uses its DH oracle to recognize and reprogram hash queries related to sessions running as the selected party. *Again, unlike in [CCG+19], the strong Diffie-Hellman adversary does not have to guess a party whose secret long-term key will not be revealed, so it does not lose a factor U in advantage.*

At this point, we observe that every session key fresh session fails to reprogram the random oracle to be consistent with its session key, which means that every testable session key is independent of anything the adversary has observed. It follows that the adversary's advantage in this game is 0, and the claim follows. □

6 Impossibility of tightly secure explicit authentication via key confirmation

In this section we show that a large, natural, and widely-used class of compilers for turning implicitly authenticated protocols into explicitly authenticated protocols, inevitably must incur a linear security loss in the number of parties. The class includes the generic compiler from [CCG+19] which was incorrectly claimed to achieve tight security but also the MAC-based approach to turn HMQV into HMQV-C [Kra05] (which does not give explicit security bounds) and the compiler by Yang [Yan13] (which has a linear loss in the number of parties times sessions per party).

6.1 Requirements on Π and Π^+

We want to consider *generic* approaches that turn *any* implicitly authenticated key exchange protocol Π into an explicitly authenticated protocol Π^+. To this end, we will in the sequel focus on underlying protocols Π and constructions Π^+ that satisfy certain requirements that we define in this section.

Messages of Π are independent of the secret long-term keys. We consider protocols Π where the messages sent are *independent* of the parties' secret long-term keys. More precisely, with respect to the protocol syntax in Fig. 1, the Init algorithm does not use a secret long-term key sk as input, i.e.,

$$(m, st) \leftarrow \Pi.\mathsf{Init}(i, j, \mathsf{role}, \mathsf{pk}, -) \tag{40}$$

and any subsequent updates of the session state st by the Run algorithm only depend on the received message and the current session state (minus the secret long-term key). However, we still want to allow the final session key to depend on the secret long-term key, so we model this by associating a session key derivation algorithm to protocol Π that explicitly takes as input the secret long-term key. That is, the session key k is computed as

$$k \leftarrow \Pi.\mathsf{KDF}(st, \mathsf{sk}), \tag{41}$$

where st is the session's state (minus the secret long-term key), and sk is the secret long-term key.

The messages of Π must be independent of the secret long-term key because we will construct an efficient adversary which has to send messages on behalf of other parties without knowing their secret long-term keys. This includes typical implicitly authenticated protocols, such as protocols where each party sends a group element g^x for random $x \leftarrow \mathbb{Z}_p$ and the long-term secret keys are only used during key derivation at the end of the protocol. Many protocols are of this form, in particular the implicitly authenticated variant of the CCGJJ19 protocol [CCG+19], as well as the HMQV protocol [Kra05].

Remark 6.1. One class of protocols which is *not* covered by this assumption are those based on digital signatures, such as the signed Diffie-Hellman protocol. Digitally signing messages is thus a way to circumvent our impossibility result. However, note that implicitly authenticated key exchange protocols, such as [CCG+19, Kra05], typically avoid the use of digital signatures since they add very significant overhead (w.r.t. computation and communication) to the protocol. This holds in particular when tightness is considered, where the most efficient signature schemes with fully tight multi-user security (in the random oracle model) [DGJL21] are significantly more expensive than corresponding schemes without tight security.

An extension to NAXOS-like protocols [LLM07], where parties send a group element g^x where $x \leftarrow H(\mathsf{sk}, r)$ depends on the secret key of the sending party and some randomness r, as well as to reductions in the random oracle model, is discussed in Section 6.3.

Π^+ adds key confirmation messages derived from the Π session key. Given an n-message protocol Π we define the key confirmation extension protocol Π^+ to be the $(n+1)$-message protocol consisting of the n messages of protocol Π, but where the n-th message is extended with a key confirmation tag, and then a final $(n+1)$-th key confirmation message is added to the end. The key confirmation tags t, t' are derived deterministically from the session key and message transcript T of protocol Π using algorithms $\Pi^+.\mathsf{Conf}$ and $\Pi^+.\mathsf{Conf}'$, respectively. In practice $\Pi^+.\mathsf{Conf}, \Pi^+.\mathsf{Conf}'$ will be PRFs applied to the message transcript of Π using the session key from Π as the key but formally the security property we require is that they preserve the entropy of their input.[3]

Definition 6.2. We say that a function $F\colon \mathcal{K} \times \{0,1\}^* \to \{0,1\}^\tau$ is *δ-entropy-preserving* if for every $t \in \{0,1\}^\tau$ and every string $T \in \{0,1\}^*$ it holds that

$$\Pr_{k \xleftarrow{\$} \mathcal{K}}[F(k,T) = t] \leq \delta. \tag{42}$$

Finally, we assume that the session key in Π^+ is also derived deterministically from the session key and message transcript T of protocol Π using another algorithm $\Pi^+.\mathsf{KDF}$.

Remark 6.3. We have defined Π^+ such that the first key confirmation message is sent along with the last message of Π, and then the second key confirmation message is sent as a reply. This adds *one* extra message to Π, that is, Π^+ is an $(n+1)$-message protocol. Alternatively, we could have defined Π^+ so that the first key confirmation message is sent as a *reply* to the n-th message. This would have added *two* messages to Π, making Π^+ an $(n+2)$-message protocol. We consider the former approach more natural, and this is the approach used in CCGJJ19 [CCG+19] and HMQV-C [Kra05]. The latter approach is used by Yang [Yan13]. Even though we only treat the former variant here, our results apply equally to both variants.

[3] Technically, in our proof we only need the entropy preserving property when producing the *first* tag t, i.e., we only need Definition 6.2 to hold for function $\Pi^+.\mathsf{Conf}$.

Security of Π^+ from Π via a valid black-box reduction. We want to consider generic constructions of a fully forward secret (and explicitly authenticated) protocol Π^+ based on the assumption that the underlying Π is a weakly forward secret (and implicitly authenticated) protocol, plus possibly some additional assumptions, e.g., on the primitives used to create the key confirmation messages, and so on. This excludes artificial constructions of Π^+ which run Π as a redundant subroutine, but where security is achieved in a completely different way, such that Π is actually superfluous. The most natural way to establish security of Π^+ based on the security of Π is to have a security analysis of Π^+ which includes (possibly among other arguments and reductions) at least one reduction $\mathcal{R}$ to the KeySecWFS security of Π. We will argue that such a reduction cannot be tight. The full security analysis of Π^+ might include further arguments, such as reductions to the security of primitives used in the key confirmation messages.

More precisely, we assume that the security proof of Π^+ includes a black-box reduction $\mathcal{R}$, which treats the adversary $\mathcal{A}$ as a black box by submitting inputs and receiving outputs from $\mathcal{A}$ as specified in the explicitly authenticated security model, and which is able to leverage any successful $\mathcal{A}$ (independently of how $\mathcal{A}$ works internally) to break the KeySecWFS security of Π. Reduction $\mathcal{R}$ has access to the KeySecWFS security experiment of protocol Π, and to an adversary $\mathcal{A}$ on the KeySecFFS security of Π^+. We require that for every party i in the Π^+ security experiment, there exists a unique corresponding party i' in the Π security experiment, and that $\mathcal{R}$ relays all messages of Π between the adversary and its security experiment. Hence, for every session $s_{i,j}$ of Π^+ there exists a unique session $s'_{i,j}$ of Π. The additional key confirmation messages of Π^+ are simulated by $\mathcal{R}$ (in any arbitrary way).

We will also assume that the reduction is always "valid", i.e., it never makes any trivially invalid queries in its KeySecWFS security experiment. Examples of such invalid queries are, for instance, asking $\textsc{Reveal}(s)$ and $\textsc{Test}(s)$ against the same session s. We assume that a reduction rather aborts instead of making invalid queries. Obviously, any reduction $\mathcal{R}'$ that does not satisfy this can be generically transformed into a reduction $\mathcal{R}$ that does, with essentially the same running time and advantage, by simply putting a wrapper around $\mathcal{R}'$ that relays all queries and their replies, but terminates $\mathcal{R}'$ when it asks the first trivially invalid query.

Π has unique and efficiently verifiable secret keys. We assume that the public key pk of every party running protocol Π *uniquely* determines a matching secret key sk, and that one can efficiently and perfectly verify that a given sk matches a given pk. Note that this also holds for many typical implicitly authenticated high-efficiency protocols, in particular the CCGJJ19 protocol [CCG+19], but also HMQV [Kra05], NAXOS [LLM07], and many more, where a public key is a group element y of a group of order p and the matching secret key is the unique $x \in \mathbb{Z}_p$ such that $g^x = y$.

It is known that it is generally difficult to reduce the security of a protocol Π

with unique secret keys tightly to the hardness of some non-interactive complexity assumption, due to the general impossibility results of Bader *et al.* [BJLS16]. However, note here that our impossibility result is not about the tightness of security proofs for Π and reductions to non-interactive hardness assumptions, but rather about the tightness of reducing the security of a protocol Π^+ with key confirmation to the security of some underlying protocol Π (which then may or may not have a tight reduction to some hardness assumption). Hence, it is independent of the question whether Π has a tight security proof under some (non-interactive) hardness assumption or not. Note also that in order to rule out tight generic constructions of explicit authentication via key confirmation, it is sufficient to rule out such constructions for protocols with unique secret keys, as a generic construction should work in particular for protocols with unique keys.

6.2 Impossibility result

Theorem 6.4. *Let Π be an AKE protocol and let Π^+ be an AKE protocol constructed by extending Π with δ-entropy-preserving key confirmation; let $\mathcal{R}$ be a reduction which converts any adversary $\mathcal{A}$ against the* KeySecFFS *security of Π^+ into an adversary $\mathcal{R}(\mathcal{A})$ against the* KeySecWFS *security of Π; and assume that Π, Π^+, and $\mathcal{R}$ satisfy all the requirements described in Section 6.1. Then we can construct an efficient adversary $\mathcal{B}$ against the* KeySecWFS *security of Π such that*

$$\mathbf{Adv}^{\mathsf{KeySecWFS}}_{\Pi,U}(\mathcal{B}) \geq \mathbf{Adv}^{\mathsf{KeySecWFS}}_{\Pi,U}(\mathcal{R}(\mathcal{A})) - \frac{1}{U} - \frac{1}{2^{|\mathsf{key}|}} - \delta, \tag{43}$$

where U is the number of parties, and $|\mathsf{key}|$ is the length of the session keys in Π^+.[4]

Interpretation of Theorem 6.4. We can assume that $2^{-|\mathsf{key}|}$ and δ are negligibly small. Hence, the theorem states that if $\mathbf{Adv}^{\mathsf{KeySecWFS}}_{\Pi,U}(\mathcal{R}(\mathcal{A}))$ is significantly greater than $1/U$, then $\mathcal{B}$ breaks the security of Π with non-negligible advantage $\mathbf{Adv}^{\mathsf{KeySecWFS}}_{\Pi,U}(\mathcal{B})$. But if Π is KeySecWFS-secure, this $\mathcal{B}$ cannot exist. Therefore the advantage $\mathbf{Adv}^{\mathsf{KeySecWFS}}_{\Pi,U}(\mathcal{R}(\mathcal{A}))$ of the reduction $\mathcal{R}$ cannot be significantly greater than $1/U$. However, the reduction is a black-box reduction and supposed to work even with an adversary with advantage close to 1 (such as the hypothetical adversary $\mathcal{A}$ described below). Hence, the reduction must lose a factor of at least U.

Theorem 6.4 is formulated in terms of key secrecy, as it considers reductions leveraging an adversary $\mathcal{A}$ breaking the full forward secrecy (KeySecFFS) of Π^+. However, the impossibility result could equally have been phrased in terms of entity authentication. To see this, note that while the proof of Theorem 6.4 describes a meta-reduction which simulates a hypothetical adversary $\mathcal{A}$ breaking key secrecy, in Step 4 of this hypothetical adversary $\mathcal{A}$ also breaks entity authentication. Consequently, Theorem 6 of [CCG+19], which is technically a

[4] The proof of Theorem 6.4 has been slightly simplified compared to the conference version.

claim about entity authentication, not key secrecy, cannot be correct by (the proof of) Theorem 6.4.

Proof. As common in proofs based on meta-reductions [HJK12, LW14, BJLS16] we first describe an (inefficient) hypothetical adversary $\mathcal{A}$. Then we explain how it is possible to efficiently simulate this hypothetical adversary for any reduction $\mathcal{R}$ that is "too tight", which yields a contradiction. We call this $\mathcal{A}$-simulating meta-reduction $\mathcal{B}$.

Description of the hypothetical adversary $\mathcal{A}$. Consider the following (hypothetical) adversary $\mathcal{A}$ against the kFreshFFS security of protocol Π^+.

1. $\mathcal{A}$ receives a dictionary **pk** containing all the public keys of all users. It picks $i^* \xleftarrow{\$} \{1, \ldots, U\}$ at random.

2. $\mathcal{A}$ initiates U protocol runs as follows. For every party i it executes one run of Π^+ with party $j = i + 1 \bmod U$ (in the remainder of this section we use $\{1, \ldots, U\}$ as the representatives of integers modulo U, so $j \in \{1, \ldots, U\}$), up to the point where the experiment outputs the *first* key confirmation message on behalf of j. $\mathcal{A}$ does not respond with the second key confirmation message.

   More precisely, if Π is an n-message protocol for n even, then $\mathcal{A}$ impersonates every party i as an initiator by querying

   $$s_{i,j} \leftarrow \textsc{NewSession}(j, i, \mathsf{resp}) \tag{44}$$

   to create a session id $s_{i,j}$. Note that in our notation the session identifier $s_{i,j}$ refers to a session at party j where the adversary impersonates party i towards it in $\mathcal{A}$'s experiment. Then $\mathcal{A}$ computes

   $$(m, st_{i,j}) \leftarrow \Pi.\mathsf{Init}(i, j, \mathsf{init}, \mathsf{pk}_j, -) \tag{45}$$

   and queries $\textsc{Send}(s_{i,j}, m)$ in order to send m to session $s_{i,j}$ on behalf of party i. Here we use the assumption that the protocol messages of Π are independent of the secret long-term key sk_i.

   If Π is a two-message protocol, then the experiment will respond with the second message of Π and the key confirmation message on behalf of j. If Π has more than two messages (i.e., $n \in \{4, 6, 8, \ldots\}$), then $\mathcal{A}$ continues to simulate all further messages of Π using $st_{i,j}$ on behalf of i by appropriate $\textsc{Send}$ queries, until the experiment outputs the first key confirmation message.

   If Π is an n-message protocol for n odd, then $\mathcal{A}$ proceeds similarly, except that instead of sending the first protocol message, it queries $(m, s_{i,j}) \leftarrow \textsc{NewSession}(j, i, \mathsf{init})$ in order to receive the first protocol message m from j to i and a corresponding session id $s_{i,j}$.

   In total $\mathcal{A}$ obtains U key confirmation messages from its security experiment.

3. So far all queries of $\mathcal{A}$ were independent of i^*. Now $\mathcal{A}$ obtains the long-term keys sk_i of all parties except i^*, by querying RevealLTK(i) for all $i \neq i^*$. The adversary aborts if anything is wrong. More precisely:

   (a) $\mathcal{A}$ checks whether all secret keys returned by the experiment match the public keys and aborts if not. Here we use that Π has unique and efficiently verifiable secret keys.

   Intuitively, this forces a reduction $\mathcal{R}$ simulating the experiment to relay all RevealLTK queries to the security experiment of Π, as otherwise we are able to leverage $\mathcal{R}$ to break Π, as $\mathcal{R}$ is able to output a non-revealed, yet valid secret key. We prove this intuition below.

   (b) For all $i \neq i^*$ and $j = i + 1 \bmod U$, $\mathcal{A}$ uses sk_i to derive the session key k of $s_{i,j}$ in protocol Π. Specifically, since the messages and session states are independent of the secret long-term keys in protocol Π, $\mathcal{A}$ can recreate $s_{i,j}$'s final session state st. This can then be combined with the secret long-term key to compute the session key k in protocol Π as:

$$k \leftarrow \Pi.\mathsf{KDF}(st, \mathsf{sk}_i). \tag{46}$$

   From k and the protocol message transcript $T_{i,j}$, $\mathcal{A}$ computes the first key confirmation messages as

$$t \leftarrow \Pi^+.\mathsf{Conf}(k, T_{i,j}). \tag{47}$$

   $\mathcal{A}$ now checks that the first key confirmation message t matches the key confirmation message it received earlier from the security experiment for session $s_{i,j}$. $\mathcal{A}$ aborts if any key confirmation message is incorrect.

   Intuitively, this forces a reduction $\mathcal{R}$ simulating the security experiment to produce correct key confirmation messages for all sessions where $\mathcal{R}$ simulates party j in a session with a party $i \neq i^*$. Below we will argue that this is difficult for a reduction without predicting the index i^*, which incurs a linear security loss.

4. This last step is the "hypothetical" part of the adversary. $\mathcal{A}$ somehow computes the unique value sk_{i^*} that corresponds to the secret key of party i^*. We intentionally do not specify precisely how this is done, since a black-box reduction $\mathcal{R}$ should be able to leverage any adversary that accomplishes this. $\mathcal{A}$ then uses sk_{i^*} to finish the key exchange protocol on behalf of i^* with party $j^* = i^* + 1 \bmod U$. In particular, this involves deriving the session key k in protocol Π for session s_{i^*,j^*} as described in Step 3b), and using this to compute the first key confirmation tag as

$$t \leftarrow \Pi^+.\mathsf{Conf}(k, T_{i^*,j^*}). \tag{48}$$

   If t matches the tag that $\mathcal{A}$ received from the security experiment for this run in Step 2, then $\mathcal{A}$ derives the second key confirmation message t' of

Π^+ and sends it to session s_{i^*,j^*}. Otherwise, $\mathcal{A}$ aborts. In case of a match the second key confirmation message is computed as

$$t' \leftarrow \Pi^+.\mathsf{Conf}'(k, T_{i^*,j^*}). \tag{49}$$

Note that t' is a valid key confirmation message so s_{i^*,j^*} will accept in protocol Π^+ upon receiving it.[5] Since s_{i^*,j^*} is fresh according to kFreshFFS it is eligible for a TEST query. Now $\mathcal{A}$ asks TEST(s_{i^*,j^*}) and receives back a key k', which is either the "real" session key or a random key. Locally $\mathcal{A}$ computes s_{i^*,j^*}'s session key in protocol Π^+ as

$$k^+ \leftarrow \Pi^+.\mathsf{KDF}(k, T_{i^*,j^*}) \tag{50}$$

and outputs 1 if $k' = k^+$, and 0 otherwise.

Note that $\mathcal{A}$ is a correct (hypothetical) adversary against the KeySecFFS security of Π^+ with

$$\mathbf{Adv}^{\mathsf{KeySecFFS}}_{\Pi^+,U}(\mathcal{A}) = 1 - 1/2^{|\mathsf{key}|}. \tag{51}$$

The term $1/2^{|\mathsf{key}|}$ is the probability that a random key k' equals the "real" session key k_{i^*,j^*} "by accident", which is the only case where $\mathcal{A}$ answers incorrectly. Note that this is the best possible advantage that an adversary that asks only a single TEST query can achieve in the security experiment. Hence, any black-box reduction $\mathcal{R}$ that works for any correct adversary should, in particular, work for $\mathcal{A}$. As common in proofs based on meta-reductions, such as [Cor02, HJK12, LW14, BJLS16], our hypothetical adversary is not efficient, but we will show how it can be efficiently *simulated*, if the reduction $\mathcal{R}$ is too tight. This yields that either the reduction must be non-tight, or the underlying hardness assumption (i.e., that Π is secure in an implicitly authenticated sense) must be wrong. Since we assume that Π is secure (as otherwise any reduction to the security of Π is meaningless and trivial, anyway), we can conclude that $\mathcal{R}$ must be non-tight.

Construction of meta-reduction $\mathcal{B}$. Adversary $\mathcal{B}$ runs $\mathcal{R}$ as a subroutine. It relays all queries between $\mathcal{R}$ and the KeySecWFS security experiment of Π and simulates the hypothetical adversary $\mathcal{A}$ towards $\mathcal{R}$.

Recall from Section 6.1 that for every party i of the Π^+ experiment, there exists a unique corresponding party i' in the Π security experiment. Recall also that $\mathcal{R}$ relays all messages of Π between the adversary and its security experiment, such that for every session $s_{i,j}$ of Π^+ there exists a unique session $s'_{i,j}$ of Π, and we assume that $\mathcal{R}$ does not make any invalid queries in its security experiment that make it trivially "lose" the KeySecFFS security experiment. Consider the following events that may occur in the execution of $\mathcal{B}$.

[5] This already breaks entity authentication of Π^+ because the long-term key of s_{i^*,j^*}'s peer has not been revealed. However, since our focus in Theorem 6.4 is on forward security we let $\mathcal{A}$ continue.

Incorrect secret key sk_i with $i \neq i^*$. Let $\mathsf{IncorrectSK}$ denote the event that $\mathcal{R}$ outputs at least one secret key such that $\mathcal{A}$ aborts in Step 3a). That is, one of the secret long-term keys for party i with $i \neq i^*$ returned by $\mathcal{R}$ is not correct.

Note that $\mathcal{B}$ can efficiently check whether $\mathsf{IncorrectSK}$ occurs and trivially simulate $\mathcal{A}$ in this case. Hence, it remains to consider $\mathcal{B}$ when the event $\neg\mathsf{IncorrectSK}$ occurs.

No RevealLTK(i) for some $i \neq i^*$. Let $\mathsf{NotAllRevLTK}(i)$ denote the event that $\neg\mathsf{IncorrectSK}$ occurs, and additionally there exists $i \neq i^*$ such that $\mathcal{R}$ outputs sk_i, but it has not queried $\textsc{RevealLTK}(i)$ to its security experiment when $\mathcal{A}$ reaches Step 3a).

Note that in this case the adversary $\mathcal{A}$ simulated by $\mathcal{B}$ obtains from $\mathcal{R}$ a secret long-term key sk_i for a party i, such that $\textsc{RevealLTK}(i)$ was not asked in the KeySecWFS security experiment. Note also that $\mathcal{B}$ can efficiently check whether $\mathsf{NotAllRevLTK}(i)$ occurs, and for which party i.

If $\mathsf{NotAllRevLTK}(i)$ occurs, then we let $\mathcal{B}$ stop $\mathcal{R}$ before it issues any further queries to the KeySecWFS security experiment. Then $\mathcal{B}$ creates an additional new session $s_{i,j}$ in the KeySecWFS experiment of Π, impersonating i in a session with some other party j, and using the secret long-term key sk_i of the non-revealed i to obtain the corresponding session key $k_{i,j}$ in Π. Then, it asks $\textsc{Test}(s_{i,j})$, which is a valid query because the session is fresh, obtaining a challenge key k. $\mathcal{B}$ returns 1 if and only if $k_{i,j} = k$, breaking the KeySecWFS security.

Incorrect key confirmation message for $s_{i,j}$ with $i \neq i^*$. Let $\mathsf{IncorrectConf}$ denote the event that $\neg\mathsf{IncorrectSK}$ occurs, and additionally $\mathcal{R}$ outputs at least one key confirmation message such that $\mathcal{A}$ aborts in Step 3b). That is, one of the key confirmation messages for $s_{i,j}$ with $i \neq i^*$ returned by $\mathcal{R}$ is not correct.

Note that, due to $\neg\mathsf{IncorrectSK}$, $\mathcal{B}$ can efficiently detect whether $\mathsf{IncorrectConf}$ occurs, and trivially simulate $\mathcal{A}$ in this case. Hence, it remains to consider the case where event $\neg\mathsf{IncorrectConf}$ occurs, i.e., $\mathcal{R}$ outputs correct key confirmation messages for all $s_{i,j}$ with $i \neq i^*$.

Reveal(s_{i^*,j^*}) before Step 4 of $\mathcal{A}$. Let $\mathsf{TargetRevealed}$ denote the event that $\mathcal{R}$ queries $\textsc{Reveal}(s_{i^*,j^*})$ before Step 4 of $\mathcal{A}$, where $j^* = i^* + 1 \bmod U$ (again, we use $\{1,\ldots,U\}$ as the representatives of integers modulo U, so $j^* \in \{1,\ldots,U\}$).

Recall that adversary $\mathcal{B}$ runs $\mathcal{R}$ as a subroutine by relaying all queries between $\mathcal{R}$ and the KeySecWFS security experiment of Π. Hence, if $\mathcal{R}$ queries $\textsc{Reveal}(s_{i^*,j^*})$ and receives in reply the session key k of this session of Π, then $\mathcal{B}$ sees this query and therefore obtains the session key k of the "target" session of Π that corresponds to s_{i^*,j^*} as well.

Note that the only inefficient computation of the hypothetical adversary $\mathcal{A}$ is the computation of k in its Step 4. Since $\mathcal{B}$ obtains k through the $\textsc{Reveal}(s_{i^*,j^*})$ query of $\mathcal{R}$, provided that $\mathsf{TargetRevealed}$ occurs, $\mathcal{B}$ is able to efficiently simulate all four steps of adversary $\mathcal{A}$.

In the sequel let

$$\Delta := \neg\mathsf{IncorrectSK} \cap \neg\mathsf{NotAllRevLTK}(i) \cap \neg\mathsf{IncorrectConf} \cap \neg\mathsf{TargetRevealed} \quad (52)$$

We now consider two different events, both in the case where Δ occurs and thus $\mathcal{B}$ cannot trivially simulate $\mathcal{A}$ in the ways explained above. In the first case, $\mathcal{R}$ outputs a valid key confirmation message for session s_{i^*,j^*} without making a $\textsc{Test}(s_{i^*,j^*})$ query *before* the end of Step 2. Note that due to $\neg\mathsf{TargetRevealed}$ $\mathcal{R}$ also does not query $\textsc{Reveal}(s_{i^*,j^*})$. Intuitively, in this case, $\mathcal{R}$ has to predict a *correct* key confirmation message, which $\mathcal{B}$ can use as a test value to break the KeySecWFS security of Π.

In the second case, we consider the complementary event, where $\mathcal{R}$ either outputs an invalid key confirmation message for session s_{i^*,j^*}, or makes a $\textsc{Test}(s_{i^*,j^*})$ query *before* the end of Step 2. Intuitively, this requires $\mathcal{R}$ to predict i^* already before Step 3 of $\mathcal{A}$, which happens with probability at most $1/U$. Recall here that i^* is chosen uniformly random and the view of $\mathcal{R}$ is independent of i^* until Step 3.

Valid key confirmation for s_{i^*,j^*} and no Test(s_{i^*,j^*}). In this case event Δ occurs, and additionally $\mathcal{R}$ outputs a *correct* key confirmation message for session s_{i^*,j^*} in Step 2 of $\mathcal{A}$, and $\mathcal{R}$ has not asked a $\textsc{Test}(s_{i^*,j^*})$ query for this session *before* the end of Step 2 (i.e., before Step 3 starts).

Note that $\mathcal{R}$ also did not query $\textsc{Reveal}(s_{i^*,j^*})$ before Step 4 of $\mathcal{A}$, due to event $\neg\mathsf{TargetRevealed}$, thus, in particular it did not query $\textsc{Reveal}(s_{i^*,j^*})$ in Step 2 of $\mathcal{A}$. Yet, $\mathcal{R}$ is able to simulate a *correct* key confirmation message for session s_{i^*,j^*}. We argue that this enables $\mathcal{B}$ to break the security of Π in the KeySecWFS experiment.

Concretely, $\mathcal{B}$ proceeds as follows. Let $\widetilde{t}$ denote the correct key confirmation message simulated by $\mathcal{R}$. If Δ occurs and $\mathcal{R}$ does not query $\textsc{Test}(s_{i^*,j^*})$ before the end of Step 2, then $\mathcal{B}$ runs $\mathcal{R}$ until it queries $\textsc{Test}(s_{i^*,j^*})$. If $\mathcal{R}$ does not query $\textsc{Test}(s_{i^*,j^*})$, then it runs $\mathcal{A}$ until the end of Step 3 of $\mathcal{A}$, and then $\mathcal{B}$ queries $\textsc{Test}(s_{i^*,j^*})$ to the KeySecWFS security experiment on behalf of $\mathcal{R}$. Either way, $\mathcal{B}$ immediately stops $\mathcal{R}$ after the $\textsc{Test}(s_{i^*,j^*})$ query.

The KeySecWFS security experiment returns a challenge key k. Then $\mathcal{B}$ computes $t \leftarrow \Pi^+.\mathsf{Conf}(k, T_{i^*,j^*})$ and checks whether $t = \widetilde{t}$. If this holds, then it returns 0 to the KeySecWFS experiment (i.e., k is a "real" key). If $t \neq \widetilde{t}$, then it returns 1.

If k is the "real" key, then we have $t = \widetilde{t}$ and $\mathcal{B}$ outputs the correct bit 0 to the KeySecWFS experiment. If k is a "random" key, then we might

still have $t = \widetilde{t}$ by accident, i.e., a false positive. This may happen either if the random key chosen by the KeySecWFS experiment is equal to the real key, which happens with probability $2^{-|\mathsf{key}|}$. Or even for a random key k we might have $\widetilde{t} = \Pi^+.\mathsf{Conf}(k, T_{i^*,j^*})$. However, due to the δ-entropy-preserving property of $\Pi^+.\mathsf{Conf}$ (see Definition 6.2) this happens with probability at most δ. Hence, the false positive probability of $\mathcal{B}$ in this case is at most $2^{-|\mathsf{key}|} + \delta$.

Invalid key confirmation for s_{i^*,j^*} or Test(s_{i^*,j^*}). This is the final remaining case, where Δ occurs, but either $\mathcal{R}$ outputs an *incorrect* key confirmation message for session s_{i^*,j^*} in Step 2 of $\mathcal{A}$, or it asks $\textsc{Test}(s_{i^*,j^*})$ query for this session *before* the end of Step 2 (i.e., before Step 3 starts).

In this case our meta-reduction $\mathcal{B}$ fails. We claim that

$$\Pr[\mathcal{B} \text{ fails}] \leq \frac{1}{U}. \tag{53}$$

To see this, note that $\mathcal{R}$ outputs valid key confirmation messages for all $s_{i,j}$ with $i \neq i^*$, since $\neg\mathsf{IncorrectConf}$ occurs. However, since $\mathcal{R}$ is independent of i^* until Step 2 starts, it can only output an *incorrect* key confirmation message for session s_{i^*,j^*} in Step 2 by predicting i^*, which happens with probability at most $1/U$.

Furthermore, since $\neg\mathsf{IncorrectSK}$ occurs, $\mathcal{R}$ outputs all long-term secret keys sk_i, with $i \neq i^*$, correctly. Furthermore, since $\neg\mathsf{NotAllRevLTK}(i)$ occurs, $\mathcal{R}$ asks $\textsc{RevealLTK}(i)$ for all $i \neq i^*$. However, if it makes a $\textsc{Test}(s_{i^*,j^*})$ query before Step 3, then it must predict the i^* for which a $\textsc{RevealLTK}(i^*)$ query is never asked already in Step 2. This happens with probability at most $1/U$.

This bounds the probability that $\mathcal{B}$ fails to $1/U$, as desired. □

6.3 Discussion of extensions and generalizations

Extension to NAXOS-like protocols. One limitation of the impossibility result is that it requires that all protocol messages are independent of the long-term secret key. As already discussed, this holds for many implicitly-authenticated, in particular those aiming at maximal efficiency. However, one interesting class of protocols that are unfortunately excluded are NAXOS-like protocols [LLM07], where parties send a group element g^x where $x = H(sk, r)$ depends on the secret key of the sending party and some randomness r.

We expect that Theorem 6.4 can also be generalized to such protocols, though. Recall that our hypothetical adversary $\mathcal{A}$ establishes protocol sessions between all parties in its Step 2. However, since it did not yet reveal the secret long-term key of any party at this step, it cannot compute $x = H(sk, r)$ for the real secret key sk of a party. Observe, though that in such NAXOS-like protocols a party receiving g^x is not able to verify that $x = H(sk, r)$, because the receiving party also does not know sk (and also not r). Even though a

reduction $\mathcal{R}$ might somehow be able to check consistency of x (e.g., using the random oracle queries made by $\mathcal{A}$), it would still have to continue the key exchange protocol like the real security experiment. Hence, $\mathcal{A}$ could simply pick x at random and send g^x to $\mathcal{R}$, and $\mathcal{R}$ would have to continue the protocol and send a key confirmation message at the end, which leads to a similar security loss as in the proof of Theorem 6.4, with almost exactly the same argument that it essentially requires $\mathcal{R}$ to "predict" the index i^* chosen by $\mathcal{A}$ already in Step 2 of $\mathcal{A}$.

The reason why we did not consider this extension is because it seems we would either have to consider specific protocols concretely, such as NAXOS specifically, which would reduce the generality of the result, or alternatively we would have to define a general notion of "efficient simulatability" of secret-key-dependent protocol messages. We refrained from the former to obtain a general impossibility result which explains the core reason of the inherent security loss of standard ways to do key confirmation, and from the latter because the argument in the proof of Theorem 6.4 is already relatively complex due to the inherent complexity of key exchange security models, and we preferred an as-clean-as-possible and more rigorous argument over full generality.

Extension to key confirmation in the random oracle model. Note that the argument in the proof of Theorem 6.4 uses the key confirmation as a "test value" to check whether a given challenge key k is real or random. This exploits that the tag produced by $\mathcal{R}$ in the tested session is computed deterministically from the real session key k' and the public protocol message transcript T_n as

$$t \leftarrow \Pi^+.\mathsf{Conf}(k', T_n). \tag{54}$$

In particular, the argument does not rely on random oracles.

Given that most highly-efficient implicitly authenticated protocols are proven secure in the random oracle model, one might ask whether it is possible to give a tightly-secure construction of Π^+ from Π in the random oracle model. For instance, one could consider computing the key confirmation message as

$$t \leftarrow H(k', T_n). \tag{55}$$

By modeling H as a random oracle the reduction $\mathcal{R}$ could potentially avoid committing to the key confirmation messages it simulates by just sending random strings $\widetilde{t}$ that can later be "explained" by re-programming H.

Unfortunately, we expect this approach to have an inherent tightness loss too. The reason for this is that the reduction needs to simulate the random oracle H consistently. But in order to achieve this, $\mathcal{R}$ would have to be able to distinguish a random oracle query $H(k', T_n)$ using the "real" key (in which case it would have to return $\widetilde{t}$) from a query $H(k'', T_n)$ using an independent string k''. Since k' is the session key of Π, the reduction would thus have to be able to distinguish session keys of Π from random, which should give rise to another attacker $\mathcal{B}$ on Π that proceeds as follows:

1. $\mathcal{B}$ is again a meta-reduction, which runs $\mathcal{R}$ as a subroutine, relays all queries between $\mathcal{R}$ and its security experiment, and simulates our hypothetical adversary $\mathcal{A}$ until the end of Step 2.

2. Then $\mathcal{B}$ picks an arbitrary random session $s_{i,j}$ and queries $\textsc{Test}(s_{i,j})$ to the security experiment of Π, receving back a challenge key k.

3. Now $\mathcal{B}$ issues many random oracle queries of the form $H(k_i, T_n)$ to $\mathcal{R}$, where $k_1, ..., k_Q$ are chosen at random, but $k_\ell := k$ is defined as the challenge key k for some random index $\ell \xleftarrow{\$} \{1, \ldots, Q\}$.

4. Now either $\mathcal{R}$ is able to distinguish the query with a "real" key k from a "random" one. In this case, $\mathcal{B}$ can also distinguish the real key from a random one, by checking whether $m_{n+1} = H(k_\ell, T_n)$.

   Or $\mathcal{R}$ is not able to distinguish the query with a "real" key k from a "random" one. In this case, $\mathcal{R}$ will fail with probability $1 - 1/Q$, so that once again we can simulate $\mathcal{A}$ efficiently because it aborts if $\mathcal{R}$ fails.

The above proof idea is only a sketch, and we expect a rigorous proof to be significantly more complex and subtle. In this paper we focused on the simpler and cleaner case of ruling out tight standard model constructions, which is also what was claimed in [CCG+19].

References

[ABR01] Michel Abdalla, Mihir Bellare, and Phillip Rogaway. The oracle Diffie-Hellman assumptions and an analysis of DHIES. In David Naccache, editor, *Topics in Cryptology – CT-RSA 2001*, volume 2020 of *Lecture Notes in Computer Science*, pages 143–158, San Francisco, CA, USA, April 8–12, 2001. Springer, Heidelberg, Germany. doi:10.1007/3-540-45353-9_12. 4

[BG11] Colin Boyd and Juan Manuel González Nieto. On forward secrecy in one-round key exchange. In Liqun Chen, editor, *13th IMA International Conference on Cryptography and Coding*, volume 7089 of *Lecture Notes in Computer Science*, pages 451–468, Oxford, UK, December 12–15, 2011. Springer, Heidelberg, Germany. 2, 3, 4

[BG20] Colin Boyd and Kai Gellert. A Modern View on Forward Security. *The Computer Journal*, 64(4):639–652, 08 2020. arXiv:https://academic.oup.com/comjnl/article-pdf/64/4/639/37161647/bxaa104.pdf, doi:10.1093/comjnl/bxaa104. 2

[BHJ+15] Christoph Bader, Dennis Hofheinz, Tibor Jager, Eike Kiltz, and Yong Li. Tightly-secure authenticated key exchange. In Yevgeniy Dodis and Jesper Buus Nielsen, editors, *TCC 2015: 12th Theory of Cryptography Conference, Part I*, volume 9014 of *Lecture Notes in Computer Science*, pages 629–658, Warsaw, Poland,

March 23–25, 2015. Springer, Heidelberg, Germany. doi:10.1007/978-3-662-46494-6_26. 2

[BJLS16] Christoph Bader, Tibor Jager, Yong Li, and Sven Schäge. On the impossibility of tight cryptographic reductions. In Marc Fischlin and Jean-Sébastien Coron, editors, *Advances in Cryptology – EUROCRYPT 2016, Part II*, volume 9666 of *Lecture Notes in Computer Science*, pages 273–304, Vienna, Austria, May 8–12, 2016. Springer, Heidelberg, Germany. doi:10.1007/978-3-662-49896-5_10. 6, 28, 29, 31

[BPR00] Mihir Bellare, David Pointcheval, and Phillip Rogaway. Authenticated key exchange secure against dictionary attacks. In Bart Preneel, editor, *Advances in Cryptology – EUROCRYPT 2000*, volume 1807 of *Lecture Notes in Computer Science*, pages 139–155, Bruges, Belgium, May 14–18, 2000. Springer, Heidelberg, Germany. doi:10.1007/3-540-45539-6_11. 2, 4, 6, 9

[BR94] Mihir Bellare and Phillip Rogaway. Entity authentication and key distribution. In Douglas R. Stinson, editor, *Advances in Cryptology – CRYPTO'93*, volume 773 of *Lecture Notes in Computer Science*, pages 232–249, Santa Barbara, CA, USA, August 22–26, 1994. Springer, Heidelberg, Germany. doi:10.1007/3-540-48329-2_21. 3, 6

[BR95] Mihir Bellare and Phillip Rogaway. Provably secure session key distribution: The three party case. In *27th Annual ACM Symposium on Theory of Computing*, pages 57–66, Las Vegas, NV, USA, May 29 – June 1, 1995. ACM Press. doi:10.1145/225058.225084. 6

[CCG+19] Katriel Cohn-Gordon, Cas Cremers, Kristian Gjøsteen, Håkon Jacobsen, and Tibor Jager. Highly efficient key exchange protocols with optimal tightness. In Alexandra Boldyreva and Daniele Micciancio, editors, *Advances in Cryptology – CRYPTO 2019, Part III*, volume 11694 of *Lecture Notes in Computer Science*, pages 767–797, Santa Barbara, CA, USA, August 18–22, 2019. Springer, Heidelberg, Germany. doi:10.1007/978-3-030-26954-8_25. 2, 3, 4, 5, 6, 18, 22, 23, 24, 25, 26, 27, 28, 36

[CF15] Cas Cremers and Michèle Feltz. Beyond eCK: perfect forward secrecy under actor compromise and ephemeral-key reveal. *Des. Codes Cryptogr.*, 74(1):183–218, 2015. URL: https://doi.org/10.1007/s10623-013-9852-1. 3, 4

[CK01] Ran Canetti and Hugo Krawczyk. Analysis of key-exchange protocols and their use for building secure channels. In Birgit Pfitzmann, editor, *Advances in Cryptology – EUROCRYPT 2001*, volume 2045 of *Lecture Notes in Computer Science*, pages 453–474,

Innsbruck, Austria, May 6–10, 2001. Springer, Heidelberg, Germany. doi:10.1007/3-540-44987-6_28. 6

[Cor02] Jean-Sébastien Coron. Optimal security proofs for PSS and other signature schemes. In Lars R. Knudsen, editor, *Advances in Cryptology – EUROCRYPT 2002*, volume 2332 of *Lecture Notes in Computer Science*, pages 272–287, Amsterdam, The Netherlands, April 28 – May 2, 2002. Springer, Heidelberg, Germany. doi:10.1007/3-540-46035-7_18. 31

[dFW19] Cyprien Delpech de Saint Guilhem, Marc Fischlin, and Bogdan Warinschi. Authentication in key-exchange: Definitions, relations and composition. Cryptology ePrint Archive, Report 2019/1203, 2019. https://eprint.iacr.org/2019/1203. 41

[dFW20] Cyprien de Saint Guilhem, Marc Fischlin, and Bogdan Warinschi. Authentication in key-exchange: Definitions, relations and composition. In Limin Jia and Ralf Küsters, editors, *CSF 2020: IEEE 33st Computer Security Foundations Symposium*, pages 288–303, Boston, MA, USA, jun 22-26 2020. IEEE Computer Society Press. doi:10.1109/CSF49147.2020.00028. 2, 6, 9, 10, 12, 40

[DGJL21] Denis Diemert, Kai Gellert, Tibor Jager, and Lin Lyu. More efficient digital signatures with tight multi-user security. In Juan Garay, editor, *PKC 2021: 24th International Conference on Theory and Practice of Public Key Cryptography, Part II*, volume 12711 of *Lecture Notes in Computer Science*, pages 1–31, Virtual Event, May 10–13, 2021. Springer, Heidelberg, Germany. doi:10.1007/978-3-030-75248-4_1. 26

[FGSW16] Marc Fischlin, Felix Günther, Benedikt Schmidt, and Bogdan Warinschi. Key confirmation in key exchange: A formal treatment and implications for TLS 1.3. In *2016 IEEE Symposium on Security and Privacy*, pages 452–469, San Jose, CA, USA, May 22–26, 2016. IEEE Computer Society Press. doi:10.1109/SP.2016.34. 2

[GJ18] Kristian Gjøsteen and Tibor Jager. Practical and tightly-secure digital signatures and authenticated key exchange. In Hovav Shacham and Alexandra Boldyreva, editors, *Advances in Cryptology – CRYPTO 2018, Part II*, volume 10992 of *Lecture Notes in Computer Science*, pages 95–125, Santa Barbara, CA, USA, August 19–23, 2018. Springer, Heidelberg, Germany. doi:10.1007/978-3-319-96881-0_4. 2

[HJK12] Dennis Hofheinz, Tibor Jager, and Edward Knapp. Waters signatures with optimal security reduction. In Marc Fischlin, Johannes Buchmann, and Mark Manulis, editors, *PKC 2012: 15th International Conference on Theory and Practice of Public Key Cryptogra-*

phy, volume 7293 of *Lecture Notes in Computer Science*, pages 66–83, Darmstadt, Germany, May 21–23, 2012. Springer, Heidelberg, Germany. doi:10.1007/978-3-642-30057-8_5. 6, 29, 31

[HJK+21] Shuai Han, Tibor Jager, Eike Kiltz, Shengli Liu, Jiaxin Pan, Doreen Riepel, and Sven Schäge. Authenticated key exchange and signatures with tight security in the standard model. In Tal Malkin and Chris Peikert, editors, *Advances in Cryptology – CRYPTO 2021, Part IV*, volume 12828 of *Lecture Notes in Computer Science*, pages 670–700, Virtual Event, August 16–20, 2021. Springer, Heidelberg, Germany. doi:10.1007/978-3-030-84259-8_23. 2

[JKRS21] Tibor Jager, Eike Kiltz, Doreen Riepel, and Sven Schäge. Tightly-secure authenticated key exchange, revisited. In Anne Canteaut and François-Xavier Standaert, editors, *Advances in Cryptology – EUROCRYPT 2021, Part I*, volume 12696 of *Lecture Notes in Computer Science*, pages 117–146, Zagreb, Croatia, October 17–21, 2021. Springer, Heidelberg, Germany. doi:10.1007/978-3-030-77870-5_5. 2

[KPW13] Hugo Krawczyk, Kenneth G. Paterson, and Hoeteck Wee. On the security of the TLS protocol: A systematic analysis. In Ran Canetti and Juan A. Garay, editors, *Advances in Cryptology – CRYPTO 2013, Part I*, volume 8042 of *Lecture Notes in Computer Science*, pages 429–448, Santa Barbara, CA, USA, August 18–22, 2013. Springer, Heidelberg, Germany. doi:10.1007/978-3-642-40041-4_24. 5, 13

[Kra05] Hugo Krawczyk. HMQV: A high-performance secure Diffie-Hellman protocol. In Victor Shoup, editor, *Advances in Cryptology – CRYPTO 2005*, volume 3621 of *Lecture Notes in Computer Science*, pages 546–566, Santa Barbara, CA, USA, August 14–18, 2005. Springer, Heidelberg, Germany. doi:10.1007/11535218_33. 2, 3, 4, 25, 26, 27

[LLM07] Brian A. LaMacchia, Kristin Lauter, and Anton Mityagin. Stronger security of authenticated key exchange. In Willy Susilo, Joseph K. Liu, and Yi Mu, editors, *ProvSec 2007: 1st International Conference on Provable Security*, volume 4784 of *Lecture Notes in Computer Science*, pages 1–16, Wollongong, Australia, November 1–2, 2007. Springer, Heidelberg, Germany. 6, 26, 27, 34

[LS17] Yong Li and Sven Schäge. No-match attacks and robust partnering definitions: Defining trivial attacks for security protocols is not trivial. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 2017: 24th Conference on Computer and Communications Security*, pages 1343–1360,

Dallas, TX, USA, October 31 – November 2, 2017. ACM Press. doi:10.1145/3133956.3134006. 9

[LW14] Allison B. Lewko and Brent Waters. Why proving HIBE systems secure is difficult. In Phong Q. Nguyen and Elisabeth Oswald, editors, *Advances in Cryptology – EUROCRYPT 2014*, volume 8441 of *Lecture Notes in Computer Science*, pages 58–76, Copenhagen, Denmark, May 11–15, 2014. Springer, Heidelberg, Germany. doi:10.1007/978-3-642-55220-5_4. 6, 29, 31

[PQR21] Jiaxin Pan, Chen Qian, and Magnus Ringerud. Signed diffie-hellman key exchange with tight security. In Kenneth G. Paterson, editor, *Topics in Cryptology – CT-RSA 2021*, volume 12704 of *Lecture Notes in Computer Science*, pages 201–226, Virtual Event, May 17–20, 2021. Springer, Heidelberg, Germany. doi:10.1007/978-3-030-75539-3_9. 2

[RZ18] Phillip Rogaway and Yusi Zhang. Simplifying game-based definitions - indistinguishability up to correctness and its application to stateful AE. In Hovav Shacham and Alexandra Boldyreva, editors, *Advances in Cryptology – CRYPTO 2018, Part II*, volume 10992 of *Lecture Notes in Computer Science*, pages 3–32, Santa Barbara, CA, USA, August 19–23, 2018. Springer, Heidelberg, Germany. doi:10.1007/978-3-319-96881-0_1. 12

[Yan13] Zheng Yang. Modelling simultaneous mutual authentication for authenticated key exchange. In *FPS*, volume 8352 of *Lecture Notes in Computer Science*, pages 46–62. Springer, 2013. 2, 25, 26

A Additional notes on definitions

A.1 Achieving key-match soundness.

We proved concretely (Proposition 5.1) that the CCGJJ protocol achieves key-match soundness based on the protocol's structure. Interestingly, de Saint Guilhem et al. [dFW20, Thm. 2] proved that key-match soundness generically follows from key secrecy and regular match security. Unfortunately, their proof has a large tightness gap since its main reduction needs to guess two sessions among all possible sessions. Avoiding this guess seems difficult to do in full generality. However, if we are willing to impose a small amount of structure on our protocols, as well as resorting to the random oracle model, we can prove it tightly. The structure we require is that session keys are derived using a random oracle, and that this random oracle call includes the protocol messages communicated up to this point. Other values will typically also be included into this call, such as Diffie-Hellman secrets, party identities, certificates, etc., but we only explicitly require the messages to be included. This gives the following intuitive result.

Proposition A.1. *Let* Π *be a protocol where session keys are derived using a random oracle, and where this random oracle call includes the messages up until that point, then*

$$\mathbf{Adv}^{\mathsf{KMSound}}_{\Pi,U}(\mathcal{A}) \leq \mathbf{Adv}^{\mathsf{SelKeySecWFS}}_{\Pi,U}(\mathcal{B}_1) + \mathbf{Adv}^{\mathsf{Match}}_{\Pi,U}(\mathcal{B}_2) + \frac{q_{RO}}{2^{|\mathsf{key}|}} \tag{56}$$

Note: *this assumes partnering based on matching conversations.*

A.2 Explicit entity authentication vs. explicit key authentication.

Here we prove that explicit *entity* authentication reduces to explicit *key* authentication. This is basically a restatement of Proposition 9 in [dFW19][6]. Note that Proposition A.2 describes a reduction for the *same* protocol Π unlike in Section 4 where the reductions are from the extended protocol Π^+ to the underlying protocol Π.

Proposition A.2. *Let* Π *be a protocol and* $\mathcal{A}$ *and adversary against* Π *for (almost-)full entity authentication. Then* $\mathcal{A}$ *is also an adversary against match soundness, implicit key authentication, key match soundness and (almost-)full key authentication, and*

$$\mathbf{Adv}^{\mathsf{fexEntAuth}}_{\Pi,U}(\mathcal{A}) \leq \mathbf{Adv}^{\mathsf{Match}}_{\Pi,U}(\mathcal{A}) + \mathbf{Adv}^{\mathsf{iKeyAuth}}_{\Pi,U}(\mathcal{A}) + \mathbf{Adv}^{\mathsf{KMSound}}_{\Pi,U}(\mathcal{A}) + \mathbf{Adv}^{\mathsf{fexKeyAuth}}_{\Pi,U}(\mathcal{A})$$

$$\mathbf{Adv}^{\mathsf{afexEntAuth}}_{\Pi,U}(\mathcal{A}) \leq \mathbf{Adv}^{\mathsf{Match}}_{\Pi,U}(\mathcal{A}) + \mathbf{Adv}^{\mathsf{iKeyAuth}}_{\Pi,U}(\mathcal{A}) + \mathbf{Adv}^{\mathsf{KMSound}}_{\Pi,U}(\mathcal{A}) + \mathbf{Adv}^{\mathsf{afexKeyAuth}}_{\Pi,U}(\mathcal{A})$$

Proof. The proofs of full and almost-full explicit entity authentication are identical so we only give a proof of the former.

Suppose $s \in \mathcal{L}_{\mathsf{rcv}}$ is a session that receives the last message of the protocol, and for which fexEntAuth is violated (so s has accepted). We consider two cases:

1. There is a session s' such that $\mathsf{Partner}(s, s') = \mathsf{true}$, but $s.\mathsf{peer} \neq s.\mathsf{party}$. We consider two further sub-cases:
   (a) $\mathsf{SameKey}(s, s') \neq \mathsf{true}$. In this case Match is violated.
   (b) $\mathsf{SameKey}(s, s') = \mathsf{true}$. In this case iKeyAuth is violated.
2. There is no session s' such that $\mathsf{Partner}(s, s') = \mathsf{true}$, but $\mathsf{aFresh}(s) = \mathsf{true}$. We consider two further sub-cases:
   (a) There is session s'' such that $\mathsf{SameKey}(s, s'') = \mathsf{true}$. In this case KMSound is violated.
   (b) There is no session s' such that $\mathsf{SameKey}(s, s') = \mathsf{true}$. In this case fexKeyAuth is violated.

□

[6] Although [dFW19, Prop. 9] appears to miss an iKeyAuth term.