Authentication in Key-Exchange: Definitions, Relations and Composition

Cyprien Delpech de Saint Guilhem[1,2], Marc Fischlin[3], and Bogdan Warinschi[2]

[1] imec-COSIC, KU Leuven, Belgium
[2] Dept Computer Science, University of Bristol, United Kingdom
[3] Computer Science, Technische Universität Darmstadt, Germany
cyprien.delpechdesaintguilhem@kuleuven.be, marc.fischlin@cryptoplexity.de, bogdan@cs.bris.ac.uk

Abstract. We present a systematic approach to define and study authentication notions in authenticated key-exchange protocols. We propose and use a flexible and expressive predicate-based definitional framework. Our definitions capture key and entity *authentication*, in both implicit and explicit variants, as well as key and entity *confirmation*, for authenticated key-exchange protocols. In particular, we capture critical notions in the authentication space such as *key-compromise impersonation* resistance and security against *unknown key-share* attacks. We first discuss these definitions within the Bellare–Rogaway model and then extend them to Canetti–Krawczyk-style models.
We then show two useful applications of our framework. First, we look at the authentication guarantees of three representative protocols to draw several useful lessons for protocol design. The core technical contribution of this paper is then to formally establish that composition of secure implicitly authenticated key-exchange with subsequent confirmation protocols yields explicit authentication guarantees. Without a formal separation of implicit and explicit authentication from secrecy, a proof of this folklore result could not have been established.

1 Introduction

The commonly expected level of security for authenticated key-exchange (AKE) protocols comprises two aspects. *Authentication* provides guarantees on the identities of the parties involved in the protocol execution. *Secrecy* promises that the key is not known by active adversaries. The two together ensure that the key is only known by the "right" parties.

The apparent simplicity of these informal definitions hides a complex landscape as both secrecy and authentication come in many related but distinct flavours. Authentication can be one-way or mutual; it can refer directly to the entities involved in the protocol run (*entity authentication*) or indirectly to the identities of the parties that hold the keys (*key authentication*); it can be *explicit* (i.e. be achieved during the protocol run) or *implicit* (i.e. rely on use of the session-key in other protocols). In addition, AKE protocols are often expected to guarantee *key-confirmation*, where a party which derives a key is convinced that it has also been derived by another session, to ensure that only properly shared keys are used in communication. Each of these properties may also come with different levels of strength due to the asymmetry in most two-party protocols.[4]

Starting with the seminal work of Bellare and Rogaway [BR94], research has made steady progress in formalizing and clarifying what some of these security notions signify. For example, the early definition proposed in [BR94] requires explicit entity authentication for all AKE protocols: when a session finishes its execution, there exists a unique session of the intended peer to which it is partnered.[5] In particular, this definition is applicable to arbitrary two-party protocols (i.e. it does not require that parties derive keys). Subsequent work resulted in numerous extensions and variations including changes to the

[4] One party always terminates first after sending the last message and is not able to confirm the message's correct delivery.

[5] Partnering was originally defined through matching conversations.

core partnering mechanism [BPR00,BR95,CK01,BFWW11,LS17] and extensions of the adversarial powers [CK01,LLM07,CF12]. Over time, definitions have shifted to guarantees associated only with keys, separated from identities [Kra05,BSWW13], and formalized other aspects not considered by original definitions e.g. various forms of forward secrecy [Kra05,CF12] and key-confirmation properties [FGSW16].

All these developments were guided chiefly by protocol design ideas and intuitive understanding of the security guarantees and therefore happened outside of a complete framework of security notions. Despite some works which attempt to systematize and understand the relative merits of the different models [CBH05a,Ust09,Cre11], emphasis was on protocol design and more pressing aspects (e.g. privacy) rather than on a thorough evaluation of the different guarantees (including authentication). This lack of comprehensive study has led to missing definitions, unclear relations between properties, and the use of folklore results which lack formal support. Perhaps surprisingly, core security notions such as *implicit authentication* have influenced the design of large classes of protocols while not being formally defined. Similarly, large classes of attacks on authentication, such as *unknown key-share* or *key-compromise impersonation*, have led many protocols to attempt to avoid them, but no security definition has been argued to properly defend against such attacks.

The case of implicit authentication. This discussion is best illustrated by the literature on implicit authentication which has guided protocol design for decades [MTI86,Kra05,CF12]. This line of research weakens the level of authentication provided. It requires that when a session derives a key, the adversary cannot force an unintended party to derive the same key.

Unfortunately, implicit authentication is tangled together with authentication and secrecy guarantee into single monolithic definitions which leads to several undesirable consequences. First, it makes the definitions themselves difficult to understand. The relation between the trust models for authentication and secrecy is not clear cut. For example, secrecy for some session when the intended partner is adversarially controlled does not make sense, whereas integrity guarantees are still desirable. This makes it non-obvious that the two notions can actually be compounded. Arguing that the definition captures the "right" guarantees requires a rather cumbersome reduction from an authentication attack to an attack against secrecy. It also makes the definition less portable; each change in the underlying execution model, e.g. varying the corruption model, requires dealing, unnecessarily, with the idiosyncrasies associated to secrecy (e.g. specific freshness notions).

Finally, it makes reasoning about properties related strictly to authentication cumbersome. A simple example is that it is difficult to justify that implicit authentication is a consequence of explicit authentication. More importantly, the lack of standalone definitions for authentication, separated from that of secrecy, makes it impossible to justify the folklore result that use of an implicitly authenticated key yields explicit authentication guarantees. This is a highly desirable property which allows for more efficient key-exchange protocol which delay the authentication guarantees to when keys are used rather than when they are agreed. This folklore composition, sometime used as alternative definition for implicit authentication, has no rigorous justification: there is no formal proof which establishes if, and under what conditions, the *use* of implicitly authenticated keys provides further authentication guarantees to parties.[6]

Our results. To address these gaps, we present a comprehensive study of the various forms of authentication in AKE protocols. We discuss how different combinations of properties relate to each other through implications and/or equivalence relations. We detail our results below.

Definitions. We identify and formally define a range of authentication properties. Figure 1 partially summarizes our definitional contributions; it shows that we cover implicit and explicit authentication, for both for keys and for entities, together with key and entity confirmation. Entity authentication and confirmation have not been defined separately from the other notions before in the literature and we do so here for completeness and symmetry, and to guide protocol design.

[6] Yang [Yan13] provides a compiler to add entity authentication to secure key exchange protocols via MACs and PRFs and a key refresh step, but this does not cover the case where the actual session key is used, e.g., in the subsequent channel protocol. Such a composed protocol clearly does not guarantee key secrecy anymore and one thus needs to argue along the authentication property alone to justify this property.

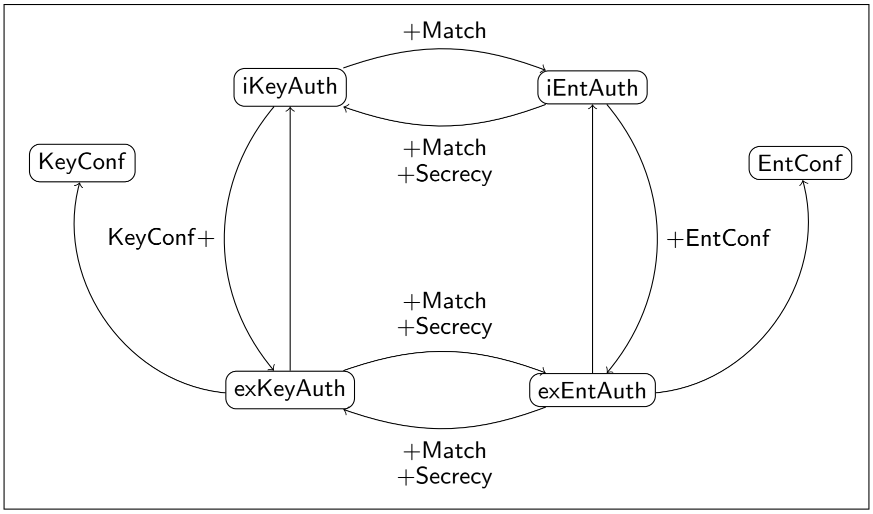


Fig. 1: Relationship of notions for implicit key and entity authentication (iKeyAuth and iEntAuth),explicit key and entity authentication (exKeyAuth and exEntAuth), key and entity confirmation (KeyConf and EntConf), Match security and key Secrecy. We note that key confirmation and explicit authentication definitions actually come in two flavours, full and almost-full, depending on which party receives the final protocol message. The implications hold accordingly for these sub versions.

Our definitions use a formalism inspired by the work of Rogaway and Stegers [RS09]. Specifically, we use logical statements to express when a desirable property is satisfied by the overall state of the protocol execution; security then demands that this property is satisfied with overwhelming probability. A key benefit of the approach is that these precise guarantees are unambiguous, unlike the English prose often used to describe authentication properties in AKE [MvOV97], and can be assessed by parsing the logical formalism.

Not evident from Figure 1 is that we cover both one-way and mutual authentication settings using a new and flexible mechanism to unify both versions. Another aspect not illustrated in Figure 1 is that some of the notions come in two flavours: “full” which is the stronger guarantee, offered to the party which receives the last message, and “almost-full” which is offered to the party that sends the last message (and therefore does not know if this message is delivered).

In our work we made several important choices about the definitional framework. The first concerns the adversarial powers and goals. While the rigorous formalization of adversarial goals is one of our main contributions, we are aware that these need to be placed in (and often depend on) the context of the available adversarial powers. A variety of choices exists, especially when deciding which parties the adversary is allowed to corrupt, and we opt for a more conservative scenario. We extend the Bellare–Rogaway (BR) model [BR94] by selectively allowing the adversary to compromise *any* of the two parties that engage in an authenticated session. One implication of this choice is that our definitions fully cover advanced properties like *key-compromise impersonation (KCI)* resistance [BJM97] and security against *unknown key-share* (UKS) attacks [BWM99]. We also fully allow the reveal of session keys as we consider that authentication should already hold upon *derivation* of the key. In a later section, we extend our definitions to the AKE models in the Canetti–Krawczyk style [CK01] which also include the reveal of ephemeral material. We show that our definitions naturally extend to this stronger adversarial model and that we can in fact separate out authentication from secrecy definitions in order to capture our stronger form of implicit authentication.

Another choice is how to capture matching sessions (sometimes called session partnering). Such sessions are deemed by the model to have communicated with one another and, among other implications, they are expected to derive the same key. The notion of partnering is crucial for defining secrecy of keys (technically to restrict the powers of the adversary in a meaningful way) but also impacts authentication. When authentication is required, then sessions deemed as matching should agree on identities. In the literature, there are two prevalent mechanisms that capture this idea: matching conversations and session identifiers. In this paper we use session identifiers – some related to keys, some related to identities – and demand that they meet a refined definition of Match security which was first proposed in earlier work [BFWW11] as "partnering security."

Such careful reflections about security notions and the logical formalism enable the intuitive understanding to be matched almost one-for-one in the security definitions, therefore guaranteeing that each stand-alone definition captures exactly what it is supposed to, something that could not be achieved with a single, secrecy-focused definition.

Relations. A second benefit of our predicate-based definitions is that the use of logic simplifies and facilitates the study of relations between the different properties. It is immediate that implicit authentication is a consequence of explicit authentication. Similarly, implicit key authentication together with key confirmation yield explicit key authentication.

The logical formulations also make it clear that although key authentication and entity authentication are obviously related, the relation is more subtle than appears at first. A clear separation is that the latter property is applicable to a larger class of protocols. Perhaps less clear is that for AKE protocols, entity authentication does not imply key authentication. Indeed, the authentication guarantee could come from protocol flows that are not involved in key generation; moreover, the key could be non-secret so authentication via keys would then be meaningless. However, we complete the study of relations between the different notions by showing formally that an equivalence holds for all protocols which satisfy Match security and for which the keys are private.

Protocols. To illustrate the different levels of key authentication we briefly discuss how three well-known protocols fare with respect to the notions we put forth. We first argue that for unauthenticated protocols like plain Diffie-Hellman the notions of (key and entity) authentication do not yield any meaningful guarantees.

More interestingly, we test our formulation of implicit authentication in the case of the HMQV protocol [Kra05]. Due to the prudent choices in our definitions, we find that we cannot use the original proof of secrecy in a black-box fashion since it holds for different adversarial powers. This indicates that existing security definitions for IAKE protocols do not straightforwardly provide the strongest implicit guarantees. Finally, we show that the much heavier TLS 1.3 protocol satisfies the strongest guarantee: explicit key authentication. Following our analysis, we derive some rules of thumb which can inform the design of protocols for AKE protocols. These analyses with our stand-alone definitions also bring out the assumptions that are necessary for authentication which turn out to be significantly weaker than those required for secrecy.

Composition result. As explained above, a protocol with implicit key authentication and key confirmation also satisfies explicit key authentication. This result can be used to prove the folklore (composition) result that *the use* of the key obtained from an implicitly authenticated protocol yields stronger authentication guarantees. Of course, the use of the key has to be meaningful, in the sense that only a party who possesses the key can successfully engage in the task. One can then regard the use of the key as a means to "strengthen" the key-exchange protocol to also provide key confirmation.

We formalize this idea as follows. We define the class of *key-confirming protocols* which, in addition to their basic functionality, provide a key confirmation guarantee upon a "successful" use of a key. Such protocols can be as simple as sending a MAC on some fixed message. For example, we argue that an authenticated channel also offers key-confirming guarantees. We then prove that running an implicitly authenticated key-exchange protocol followed by a key-confirming protocol results in an explicitly authenticated key-exchange protocol. The desired result follows by observing that the composition is still an implicitly authenticated protocol in addition to providing key confirmation.

2 Game-based security for AKE protocols

We use the framework of Brzuska et al. [BFWW11] for cryptographic games and describe its formulation of the BR model [BR94] for AKE protocols. We augment it to provide a flexible authentication framework and to capture key-confirmation as formalised by Fischlin et al. [FGSW16].

In the security game, the adversary interacts with parties executing the protocol via queries; these capture its capabilities in a real-world execution. The adversary's aim is to trigger an event defined as "bad" by the game whilst abiding by the game's limitations on queries. We use λ to denote the security parameter, let 1^λ be its unary representation and let $\mathsf{negl}(\lambda)$ denote an arbitrary negligible function. We denote by $\{0,1\}^*$ the set of all finite-length bit-strings.

Identities. We let n_i be the number of parties modelled by the game. Each party has a unique identity $i \in \mathbb{N}$ and we denote by $\mathcal{I} \subset \mathbb{N}$ the identity set of size n_i.

To separate identities who are expected to authenticate we specify a subset $\mathcal{S} \subseteq \mathcal{I}$; this is our first augmentation. This models real-world servers who have to convince clients of their identity by means of a certificate. These clients, modelled by the identities in $\mathcal{I} \setminus \mathcal{S}$, do not have such secret information. This modelling flexibly captures varying forms of authentication; a secure protocol for which $\mathcal{S} = \mathcal{I}$ provides mutual authentication for all sessions, whereas one for which $\mathcal{S} = \emptyset$ provides no authentication whatsoever; for $\mathcal{S}$ a non-empty proper subset of $\mathcal{I}$, it provides one-way authentication from identities in $\mathcal{S}$. We leave the specification of $\mathcal{S}$ within $\mathcal{I}$ as a design choice for protocols.

We assume that each party is aware of its own identity in an execution. We sometimes use id_A and id_B to distinguish between the identities of two parties A and B in an execution. We work in the *pre-specified peer model* [MU08] where each session knows its intended partner's identity from the start. We can restrict to the case that only identities of authenticating partners are known, e.g. if a client remains anonymous in a TLS connection.

Protocols and sessions. A protocol is a pair of algorithms $\pi = (\mathsf{kg}, \zeta)$ where $\mathsf{kg}(1^\lambda)$ is a randomized key generation algorithm and where ζ is the algorithm executed locally by parties engaging in the protocol. Local sessions are identified by a *local session identifier* $\ell \in \mathcal{I} \times \mathcal{I} \times \mathbb{Z}$ where $\ell = (i, j, k)$ refers to the k-th session of identity i with intended peer j. We use $\ell.\mathsf{id}$ and $\ell.\mathsf{pid}$ to refer to i and j respectively. We let n_s be the maximum value of k for any ℓ in a game.

We use the notion of session identifiers as was originally proposed by [BPR00] to "partner" two sessions as having engaged in the same execution. These are computed by the protocol itself, different from the *local* session identifier ℓ which is an artefact of the model. For simplicity we maintain the original approach of considering these session identifiers to be revealed upon acceptance.

2.1 Common game states

Security games maintain states to keep track of the execution of the protocol. This consists of five elements: a list LSID of valid local session identifiers, a list SST of protocol-related *session states*, a list LST of game-related *local session states*, a *game execution state* EST which contains global protocol-related information, and a *model state* MST which contains global game-related information, relevant to the security notion (e.g. a hidden bit) Our authentication games share the same execution, game and local states. Here we describe and augment the states from [BFWW11].

Game execution state. As in [BFWW11], the execution state EST contains a list $\mathcal{L}_{\mathsf{keys}} = \{(i, \mathsf{pk}_i, \mathsf{sk}_i, \delta_i)\}_{i \in \mathcal{I}}$ where $\delta_i \in \{\mathsf{honest}, \mathsf{corrupt}\}$ denotes whether sk_i has been corrupted.

Session state. The state of the local session with identifier $\ell = (i, j, k)$ is composed of the following:

- $(\mathsf{pk}_i, \mathsf{sk}_i)$, the long-term key pair of identity i, the "owner" of the session. This is initialised to $\ell.\mathsf{id}$'s key pair if $i \in \mathcal{S}$ or set to $\perp$ if $i \in \mathcal{I} \setminus \mathcal{S}$. One may think of the public keys as certified, in which case the party also holds a certificate cert_i for pk_i under its identity, but we omit details here.
- pk_j, the long-term public key of identity j, the intended peer of the session. This is initialised to $\ell.\mathsf{pid}$'s public key if $j \in \mathcal{S}$ or set to $\perp$ if $j \in \mathcal{I} \setminus \mathcal{S}$.
- $\mathsf{crypt} \in \{0,1\}^*$ is some protocol-specific private session state used to maintain secret values from one invocation to the next.
- $\mathsf{accept} \in \{\mathsf{true}, \mathsf{false}, \perp\}$ indicates whether the party has *accepted* or *rejected* the session as an succesful execution of π. Initially set to $\perp$, to signify *running*, accept may change to either true or false only upon termination. We assume the value of accept is public.
- $\mathsf{sid} \in \{0,1\}^* \cup \{\perp\}$, the session identifier as specified by the protocol. Initially set to $\perp$, it may be changed once to a non-trivial value. If the sid is different from $\perp$, then accept must be set to true, and vice versa,

if accept is set to true, then sid must become different from $\perp$. We assume that the value of sid is made public when accept is set to true.

- $\mathsf{key} \in \{0,1\}^* \cup \{\perp\}$ is the (session-)key locally derived during the execution. Initially set to $\perp$, it may be changed once to a non-trivial value. If the key is different from $\perp$, then accept must be set to true, and vice versa, if accept is set to true then, key must become different from $\perp$. We note that this implies that sessions must terminate with the same call to the protocol as that which sets the key and the sid, they cannot continue once the key is set.
- $\mathsf{kconf} \in \{\mathsf{full}, \mathsf{almost}, \mathsf{no}, \perp\}$ indicates the form of key confirmation that the owner expects to receive. This addition to the model captures the fact that one partner of a run always terminates first and therefore may not expect a full confirmation of the final session-key. The value of kconf is initialised to $\perp$ and set when the session is first activated.
- $\mathsf{kcid} \in \{0,1\}^* \cup \{\perp\}$ is a key-confirmation identifier, indicating sessions which will eventually derive the same key. Initially set to $\perp$, it may be changed once to a non-trivial value and may not be changed again. If key is different from $\perp$, then kcid must be different from $\perp$.

We write $\mathsf{SST}[\ell] = ((\mathsf{pk}_i, \mathsf{sk}_i), \mathsf{pk}_j, \mathsf{crypt}, \mathsf{accept}, \mathsf{sid}, \mathsf{key}, \mathsf{kconf}, \mathsf{kcid})$ to denote the session state of ℓ. We use the notation $\ell.\mathsf{sid}$ or $\ell.\mathsf{key}$ to refer to individual elements and use similar notation for the game, local session or model states.

This session state augments that of [BFWW11] with kconf and kcid from [FGSW16] along with some renaming. These are used to to modularly capture the formal definition of key confirmation of [FGSW16], similarly to our addition of $\mathcal{S}$ to capture different authentication directions. We refer to [BFWW11, Section 3] for a discussion on public session identifiers.

Local session state. The local session state consists of:

- $\delta_{\mathsf{ownr}} \in \{\mathsf{honest}, \mathsf{corrupt}\}$: denotes whether the owner of the session was corrupted before the session was completed (i.e. while $\ell.\mathsf{accept} = \perp$).
- $\delta_{\mathsf{peer}} \in \{\mathsf{honest}, \mathsf{corrupt}\}$: denotes whether the intended peer of the session was corrupted before the session was completed.
- $\delta_{\mathsf{sess}} \in \{\mathsf{fresh}, \mathsf{revealed}\}$: denotes whether the session-key for this session has been revealed to the adversary.

As in [BFWW11], keeping track of δ_{ownr} separately from δ_i allows sessions that terminated before their owner was corrupted to remain honest; this enables the modelling of *forward secrecy*. We write $\mathsf{LST}[\ell] = (\delta_{\mathsf{ownr}}, \delta_{\mathsf{peer}}, \delta_{\mathsf{sess}})$ for the local session state of session ℓ and use the notation $\ell.\delta_{\mathsf{sess}}$ to refer to individual elements.

Setup. Modelling protocols using these states requires the following procedures:

- $(\mathsf{SST}, \mathsf{EST}) \leftarrow \mathsf{setupE}(\mathsf{LSID}, \mathsf{kg}, 1^\lambda)$: for protocol-relevant components.
- $(\mathsf{LST}, \mathsf{MST}) \leftarrow \mathsf{setupG}(\mathsf{LSID}, \mathsf{SST}, \mathsf{EST}, 1^\lambda)$: for game-relevant components.

Our setupE is similar to that of [BFWW11] but it only generates long-term keys for the identities in $\mathcal{S}$ and initialises the new elements of the session state.

2.2 Session partnering

We define the partnering predicate using session identifiers.

Definition 2.1 (Partners). *We say that two sessions ℓ and ℓ' are* partners *if the predicate* $\mathsf{Partner}(\ell, \ell')$ *holds true, where*

$$\mathsf{Partner}(\ell, \ell') \iff \big[(\ell \neq \ell') \wedge (\ell.\mathsf{sid} = \ell'.\mathsf{sid} \neq \ \perp)\big].$$

Thus, to be partners, two sessions need to be administratively different and to both have set a non-trivial sid. This does not exclude the possibility that they belong to the same identity, i.e. that $\ell.\mathsf{id} = \ell'.\mathsf{id}$.

For correctness, we require that two sessions executing π without adversarial interaction derive identical sids upon accepting and are therefore partnered. We also require that two such sessions derive identical keys and kcids.

While this definition of partnering appears similar to that of “matching” sessions in CK-like models [CK01,Kra05,LLM07], it differs it two important aspects. The first is that it does not involve the sessions’ identities, thus separating out authentication notions. The second is that our sids are derived by the protocol itself rather than arbitrarily set by a higher layer. We note that our notion of sids superseeds that of matching conversations used for partnering in [Kra05,LLM07].

2.3 Adversarial interaction and common queries

The adversary $\mathcal{A}$ is a probabilistic polynomial-time (PPT) algorithm that interacts with a game through queries specified by a set Q. Upon receiving a query $q \in Q$, the game has a behaviour algorithm χ which takes q together with the state to return a response to $\mathcal{A}$.

Not every query is always valid; this is captured by the **Valid** predicate which the game evaluates each time a query q is received. Based on q and on the current state, **Valid** returns either true or false which determines if χ is executed on q.

In addition to the common states, our security games for authentication notions also share a query set Q. We specify here the Send, Reveal and $\mathsf{Corrupt}$ queries following the work of Brzuska et al. but, as we model KCI resistance, we modify the **Valid** predicate.

The Send query. Whatever the game, Q always includes the Send query. It takes an identifier $\ell \in \mathsf{LSID}$ and a message $m \in \{0,1\}^*$ as inputs and is processed by χ by running π on $\mathsf{SST}[\ell]$ with input m. This updates SST and returns a response m' which is given to $\mathcal{A}$ together with accept and also sid if m triggered the termination of the session. This gives control of the network to $\mathcal{A}$ and allows it to forward, alter, delay, create or delete messages.

The Reveal query. When $\mathcal{A}$ submits $\mathsf{Reveal}(\ell)$, this sets $\ell.\delta_{\mathsf{key}} \leftarrow \mathsf{revealed}$ and returns $\ell._{\mathsf{key}}$.

The Corrupt query. We formalise the $\mathsf{Corrupt}(i)$ query as follows. First, the value of δ_i in $\mathcal{L}_{\mathsf{keys}}$ is set to $\mathsf{corrupt}$. Then, for any session of the format $(i,*,*)$ for which $\mathsf{accept} = \perp$, we set $\delta_{\mathsf{ownr}} \leftarrow \mathsf{corrupt}$; for any session of the format $(*,i,*)$ which is still running, we set $\delta_{\mathsf{peer}} = \mathsf{corrupt}$. Finally, sk_i is returned to $\mathcal{A}$.

The Valid predicate. A significant difference to [BFWW11] is that our **Valid** predicate allows for the adversary to submit a Send query to a session whose owner has already been corrupted. This is crucial to model KCI resistance as this notion guarantees a security property to sessions whose owner was corrupted *before* they terminated. Furthermore, the **Valid** predicate returns false if a Reveal query is made to a session whose $\mathsf{key} = \perp$.

2.4 Winning condition and formal game definition

A game considers that $\mathcal{A}$ has won a security game, i.e. broken a security property of π, if it succeeds in triggering a "bad" event. This event is defined by a predicate P which is a logical statement evaluated on the state. We denote this by $b \leftarrow \mathsf{P}(\mathsf{LSID}, \mathsf{SST}, \mathsf{LST}, \mathsf{EST}, \mathsf{MST})$, where $b \in \{0,1\}$, and $b = 1$ signifies that $\mathcal{A}$ has successfully triggered the "bad" event. We therefore define a generic security experiment as follows.

Definition 2.2. *A game G maintains a state* $(\mathsf{LSID}, \mathsf{SST}, \mathsf{LST}, \mathsf{EST}, \mathsf{MST})$ *and is defined by the tuple* $(\mathsf{setupE}, \mathsf{setupG}, Q, \mathbf{Valid}, \chi, \mathsf{P})$. *An experiment parameterised by a protocol π, an adversary $\mathcal{A}$ and a game G is executed as follows.*

1. *The experiment runs* $(\mathsf{SST}, \mathsf{EST}) \leftarrow \mathsf{setupE}(\mathsf{LSID}, \mathsf{kg}, 1^\lambda)$ *and* $(\mathsf{LST}, \mathsf{MST}) \leftarrow \mathsf{setupG}(\mathsf{LSID}, \mathsf{SST}, \mathsf{EST}, 1^\lambda)$.
2. *The adversary submits queries from Q to the game which processes them with* **Valid** *and χ.*
3. *When $\mathcal{A}$ terminates, $b \leftarrow$* $\mathsf{P}(\mathsf{LSID}, \mathsf{SST}, \mathsf{LST}, \mathsf{EST}, \mathsf{MST})$ *is evaluated by the experiment which finally outputs b.*

We note that our definition of a game G includes more than [BFWW11, Definition 1], namely Q and P, as these also uniquely characterise it. We denote the experiment by $\mathrm{Exp}^{G}_{\mathcal{A},\pi}(1^\lambda)$ and we write $\mathrm{Exp}^{G}_{\mathcal{A},\pi}(1^\lambda) = b$.

2.5 Match security

Before authentication or secrecy, a "good" AKE protocol should first provide certain correctness and soundness guarantees. A Match-secure AKE protocol should ensure that:

1. Partner sessions derive the same key and kcid (properties 1 and 2 below);
2. at most two sessions derive the same sid (property 3);
3. sessions with the same kcid accept with the same key (property 4).

This guarantees disagreements cannot be created between partnered sessions. Formally, we define the following predicate.

Definition 2.3 (Match **predicate).** *The* Match *predicate evaluates to* 1 *iff* $\forall \ell, \ell', \ell'' \in \mathsf{LSID}$,

$$\begin{aligned} &(\mathsf{Partner}(\ell,\ell') \wedge \ell.\mathsf{key} \neq \perp \neq \ell'.\mathsf{key}) \implies \mathsf{Samekey}(\ell,\ell') && (1)\\ \wedge\ &(\mathsf{Partner}(\ell,\ell') \wedge \ell.\mathsf{kcid} \neq \perp \neq \ell'.\mathsf{kcid}) \implies \mathsf{Samekcid}(\ell,\ell') && (2)\\ \wedge\ &(\mathsf{Partner}(\ell,\ell') \wedge \mathsf{Partner}(\ell,\ell'')) \implies \ell' = \ell'' && (3)\\ \wedge\ &(\mathsf{Samekcid}(\ell,\ell') \wedge \ell.\mathsf{key} \neq \perp \neq \ell'.\mathsf{key}) \implies \mathsf{Samekey}(\ell,\ell'). && (4) \end{aligned}$$

where $\mathsf{Samekey}(\ell,\ell') \iff [\ell' \neq \ell \wedge \ell'.\mathsf{key} = \ell.\mathsf{key} \neq \perp]$ *and* $\mathsf{Samekcid}(\ell,\ell')$ *is defined analogously.*

We then define the Match security game G_{Match} in the sense of Definition 2.2 where π is an AKE protocol, the state, and the setupE algorithm are as in Section 2.1, the query set $Q = \{\mathsf{Send}, \mathsf{Reveal}, \mathsf{Corrupt}\}$ and the behaviour χ are as in Section 2.3, the setupG algorithm sets the LST of each session to $(\mathsf{honest}, \mathsf{honest}, \mathsf{fresh})$ and the predicate $P = \mathsf{Match}$. The advantage of an adversary $\mathcal{A}$ against the game G_{Match} with identity sets $\mathcal{I}, \mathcal{S}$ is written as

$$\mathrm{Adv}^{G_{\mathsf{Match}}}_{\mathcal{A},\pi,\mathcal{I},\mathcal{S}}(1^\lambda) = \Pr\left[\mathrm{Exp}^{G_{\mathsf{Match}}}_{\mathcal{A},\pi,\mathcal{I},\mathcal{S}}(1^\lambda) = 0\right].$$

Definition 2.4 (Match **security).** *An AKE protocol* π *is* Match*-secure for identity sets* $\mathcal{I}, \mathcal{S}$ *if, for all PPT adversaries* $\mathcal{A}$*,* $Adv^{G_{\mathsf{Match}}}_{\mathcal{A},\pi,\mathcal{I},\mathcal{S}}(1^\lambda) = \mathsf{negl}(\lambda)$.

Comparison to previous definitions. Considered in the context of the BR model, our definition of Match security refines that of [BFWW11] in two ways. We first incorporate the conditions of the $\mathsf{KCIDbind}$ predicate of [FGSW16] as conditions (2) and (4). As this work builds a unified model of AKE protocols with both authentication and key confirmation, it is reasonable to add this predicate to the definition of Match since it concerns notions of correctness and soundness, as the previous definition already did for keys.

Secondly, we remove the requirement that partnered sessions should agree on each other's identities. This condition implied that Match-secure AKE protocols already provided some form of authentication, albeit very weak. This mixed an authentication with design and soundness and we therefore remove it. We present separate definitions for authentication in the next section.

In the context of the CK-style models, as summarised in [Cre11], the usual first requirement of security is for matching sessions of uncorrupted identities to derive equal keys. We note that our notion of Match security would capture and extend this requirement if a $\mathsf{StateReveal}$ query were added to its game. We note that we do not restrict to uncorrupted identities, as we do not consider that the adversary takes control of sessions ℓ managed by the game; thus our Match predicate is only evaluated for honestly-behaving sessions.

3 Key authentication and confirmation

We present here our new predicate-based definitions for key authentication notions. We define three distinct flavours of authentication: *implicit*, *confirmation* and *explicit*. For each, we first discuss the intuitive understanding that motivates our definition and then give a formal statement. We then show that our definitions are consistent and that a protocol that combines implicit key authentication and key confirmation also provides explicit key authentication.

3.1 Implicit authentication

We take *implicit* to mean: "should there be a session ℓ' that possesses the same key as session ℓ, then the owner of session ℓ' *must* be the identity designated as the peer of session ℓ." This does not guarantee the secrecy of the key, nor whether such a session ℓ' exists. Equivalently, this means that any session whose owner is *not* designated by the peer of ℓ is incapable of deriving the same session-key. (Recall that the term "session" refers to sessions executed by the model and that this does not forbid the adversary from deriving the key itself.)

This informal notion of implicit authentication raises the question whether to only consider sessions which interact with an honest peer, or to also allow those with a corrupted peer. The impact of this distinction was first observed by Diffie et al. [DvOW92]. In their design of the station-to-station (STS) protocol they aimed to prevent an attack in which one can make an honest B believe it is sharing a key with a malicious E, whereas the actual other honest key holder A intends to communicate with B. This has later become known as an *unknown key-share* (UKS) attack [BWM99] which, ironically, was shown to apply to the STS protocol in the same work [BWM99].

For our formal definition, the question is then either to restrict the adversary's valid targets to the sessions that were executed with an honest peer, or to allow all sessions, even those that accepted with a corrupted peer, as valid targets. The first choice would comply with the idea stated in [BWM99]: "By definition, the provision of implicit key authentication is only considered in the case where B engages in the protocol with an honest entity (which E isn't)." The second choice, would instead lead to a definition where UKS attack scenarios even with dishonest peers are accounted for. Such a scenario would include a corrupt server causing a client to exchange a key with another, unintended, server. It is clear that this yields a stronger security guarantee and we therefore choose the second formulation in our definitions.

We stress that our model also captures *key-compromise impersonation* resistance [BJM97]—where the adversary knows the long-term key of a party A and tries to impersonate another party to A—since our formalization also allows the owner of target sessions to be corrupted.

Definition 3.1 (Implicit key authentication). *The* $\mathsf{iKeyAuth}$ *predicate evaluates to* 1 *if and only if*

$$\forall \ell \in \mathsf{LSID}, (\ell.\mathsf{pid} \in \mathcal{S} \land \ell.\mathsf{accept}) \implies \forall \ell' \in \mathsf{LSID}, (\mathsf{Samekey}(\ell', \ell) \implies \ell'.\mathsf{id} = \ell.\mathsf{pid}),$$

where $\ell.\mathsf{accept}$ *holds true if and only if* $\ell.\mathsf{accept} = \mathsf{true}$. *We then say that the AKE protocol* π *with identity sets* $\mathcal{I}, \mathcal{S}$ *provides* implicit key authentication *if, for all PPT* $\mathcal{A}$*,*

$$Adv_{\mathcal{A},\pi,\mathcal{I},\mathcal{S}}^{G_{\mathsf{iKeyAuth}}}(1^\lambda) = \Pr\left[Exp_{\mathcal{A},\pi,\mathcal{I},\mathcal{S}}^{G_{\mathsf{iKeyAuth}}}(1^\lambda) = 0\right] = \mathsf{negl}(\lambda),$$

where G_{iKeyAuth} *is the same as* G_{Match} *with* $\mathsf{P} = \mathsf{iKeyAuth}$.

Note that the predicate only applies to sessions that expect authentication, which is captured by the condition that $\ell.\mathsf{pid} \in \mathcal{S}$. This models the fact that one can only provide authentication to keys if one possesses authenticating information. An artefact of this is that protocols without authenticating parties (i.e. with $\mathcal{S} = \emptyset$) strictly speaking provide implicit key authentication as there is no authenticating party which the adversary can attack. Mathematically, this corresponds to a quantification over the empty set.

On the other end of the spectrum, we see that the case where $\mathcal{S} = \mathcal{I}$ rejoins mutual authentication where, upon completion, session ℓ has authenticated to session ℓ' and vice-versa. Indeed, if $\ell.\mathsf{id} \in \mathcal{S}$ and also $\ell.\mathsf{pid} \in \mathcal{S}$, then it is expected to authenticate itself to its intended peer in the same way that it expects to receive authentication. Upon both sessions completing, the predicate therefore induces a symmetry in the authentication guarantees.

As explained in the introduction, our basic definition considers the strongest adversarial model. We remark that we could relax the above requirement and only consider sessions ℓ with an honest peer, i.e. with $\ell.\delta_{\mathsf{peer}} = \mathsf{honest}$. Indeed, this notion sometimes appears in the literature: it still provides guarantees for parties who engage in sessions of the protocol with an honest peer as intended partner. Clearly, the notion neglects executions in which the intended peer is dishonest in which case one could be vulnerable to certain UKS attacks.

3.2 Key confirmation

Intuitively, this second notion is "the guarantee that another session possesses the same key." While this does not provide authentication in the sense of binding an identity to a key, we define it here because its existential guarantee is a link between implicit and explicit authentication.

Here, we note that key confirmation only makes sense for honest peers because an adversary impersonating an honest party can always compute the key and provide confirmation to the target session. To prevent this trivial attack, we introduce a freshness condition on the peer.

Definition 3.2 (Authentication freshness). *For any* $\ell \in \mathsf{LSID}$*,* $\mathsf{aFresh}(\ell)$ *evaluates to* true *if and only if*

$$\ell.\delta_{\mathsf{peer}} = \mathsf{honest}.$$

We note that this freshness notion does not prohibit Reveal queries; this is because key authentication properties are expected to hold upon *derivation* of the key, and knowledge of the key should not benefit the adversary in breaking these. Furthermore, we show that the Reveal query is not useful for the adversary to wrongfully provide confirmation. Either another session derives the same key and the adversary reveals it, but then another session with the same key does exist, and therefore confirmation holds even though it does with the adversary's intervention; or the adversary reveals the target session itself. However, the second option is not possible, since, in our model, setting the key to a non-trivial value is synonymous with accepting and terminating. Therefore the adversary cannot submit a Reveal query before the session has already accepted, at which point the adversary has already won if the session does not share a key with any other. Hence our freshness predicate does not need to eliminate trivial attacks using the Reveal query.

Fischlin et al. [FGSW16] introduced the distinction between *full* and *almost-full* key confirmation which captures the differences in guarantees that the last sender and last receiver in an AKE session can expect. We present here their predicate-based definitions and refer to [FGSW16] for a discussion. The former one says if an "authentication fresh" session with full key confirmation accepts, then there must be at least one other session holding the same key.

Definition 3.3 (Full key confirmation). *The* $\mathsf{fKeyConf}$ *predicate evaluates to* 1 *if and only if*

$$\forall \ell \in \mathsf{LSID}, (\mathsf{aFresh}(\ell) \land \ell.\mathsf{kconf} = \mathsf{full} \land \ell.\mathsf{pid} \in \mathcal{S} \land \ell.\mathsf{accept}) \implies \exists \ell' \in \mathsf{LSID} :: \mathsf{Samekey}(\ell', \ell).$$

We then say that the AKE protocol π *with identity sets* $\mathcal{I}, \mathcal{S}$ *provides* full key confirmation *if, for all PPT adversaries* $\mathcal{A}$*,*

$$Adv_{\mathcal{A},\pi,\mathcal{I},\mathcal{S}}^{G_{\mathsf{fKeyConf}}}(1^\lambda) = \Pr\left[Exp_{\mathcal{A},\pi,\mathcal{I},\mathcal{S}}^{G_{\mathsf{fKeyConf}}}(1^\lambda) = 0\right] = \mathsf{negl}(\lambda),$$

where G_{fKeyConf} *is defined similarly to* G_{iKeyAuth}*.*

We see that the session's expected level of key confirmation, which is set when the session is first activated, is captured with the condition that $\ell.\mathsf{kconf} = \mathsf{full}$ — for a given protocol π, a session can decide which key confirmation to expect if it is activated as an initiator or a responder. Also, the session ℓ in question is excluded from the existence condition by the $\mathsf{Samekey}$ predicate and therefore a session cannot confirm its own key. We note that $\mathsf{fKeyConf}$ is only tested against sessions that expect authentication, with $\ell.\mathsf{pid} \in \mathcal{S}$, as is discussed in [FGSW16, Section III.D]. Whilst this condition is not strictly required in the predicate for our result in Section 3.4, we adopt it here to align ourselves on the stand-alone definition of key confirmation. This condition also creates the artefact that protocols which do not specify any authenticating identies, by setting $\mathcal{S} = \emptyset$, trivially provide key confirmation, similarly to implicit key authentication.

As pointed out by Fischlin et al. [FGSW16], almost-full key confirmation is delicate to define. We adopt their notion saying that if such a fresh session accepts, then there must be a another session holding the same key-confirmation identifier and, moreover, if that other session has already derived a key, then it is the same one as the original session.

Definition 3.4 (Almost-full key confirmation). *The* $\mathsf{afKeyConf}$ *predicate is defined as*

$$\forall \ell \in \mathsf{LSID}, (\mathsf{aFresh}(\ell) \wedge \ell.\mathsf{kconf} = \mathsf{almost} \wedge \ell.\mathsf{pid} \in \mathcal{S} \wedge \ell.\mathsf{accept}) \implies \\ \exists \ell' \in \mathsf{LSID} :: \Big(\mathsf{Samekcid}(\ell', \ell) \wedge \big[\ell'.\mathsf{key} \neq \bot \implies \mathsf{Samekey}(\ell', \ell)\big]\Big).$$

We then say that the AKE protocol π with identity sets $\mathcal{I}, \mathcal{S}$ provides almost-full key confirmation *if, for all PPT adversaries $\mathcal{A}$,*

$$Adv_{\mathcal{A},\pi,\mathcal{I},\mathcal{S}}^{G_{\mathsf{afKeyConf}}}(1^{\lambda}) = \Pr\left[Exp_{\mathcal{A},\pi,\mathcal{I},\mathcal{S}}^{G_{\mathsf{afKeyConf}}}(1^{\lambda}) = 0\right] = \mathsf{negl}(\lambda)$$

where $G_{\mathsf{afKeyConf}}$ is defined similarly to G_{iKeyAuth}.

3.3 Explicit authentication

This third notion is much stronger than the first. Indeed, we take *explicit* to mean that authentication is obtained *at termination*, and therefore it does not rely on the potential use of the key at a later time. In other words, session ℓ, upon accepting, knows that there is another session ℓ' which has the same key and whose identity is bound to it. Intuitively, it is a combination of implicit authentication and key confirmation, and indeed it was informally defined as such in Menezes, van Oorschot and Vanstone's *Handbook of Applied Cryptography* [MvOV97]; this appears in the predicates below.

Similarly to key confirmation, the existence of a session which has already derived the same key cannot always be guaranteed due to the asymmetry of the final message. We therefore define the two analogous notions of *full* and *almost-full explicit key authentication.*

As before, we study the requirements of a freshness predicate. As in the case of implicit authentication we do not stipulate that the peer is honest for the target session when it comes to the condition that any partner holding the same key is correctly identified ($\forall \ell' \in \mathsf{LSID},\ \mathsf{Samekey}(\ell', \ell) \implies \ell'.\mathsf{id} = \ell.\mathsf{pid}$). This again provides safety against all possible UKS attacks. Only for the "liveness" condition (i.e. that there exists a party with the same key) do we require that the intended peer is honest ($\mathsf{aFresh}(\ell) \implies \exists \ell' \in \mathsf{LSID} :: \mathsf{Samekey}(\ell', \ell)$), otherwise the session may have communicated with an impersonating adversary which could trivially compute the key. Similarly to key confirmation, the Reveal query would not enable the adversary to conduct trivial attacks; this implies that the aFresh predicate is also the correct one here.

In summary, full explicit key authentication demands that for any fresh accepting session, any other session deriving the same key has the correct identity and there exists at least one other session holding the same key, if the peer is honest.

Definition 3.5 (Full explicit key authentication). *The* $\mathsf{fexKeyAuth}$ *predicate evaluates to* 1 *if and only if*

$$\forall \ell \in \mathsf{LSID}, (\ell.\mathsf{pid} \in \mathcal{S} \wedge \ell.\mathsf{kconf} = \mathsf{full} \wedge \ell.\mathsf{accept}) \implies \\ \big(\forall \ell' \in \mathsf{LSID},\ \mathsf{Samekey}(\ell', \ell) \implies \ell'.\mathsf{id} = \ell.\mathsf{pid}\big) \\ \wedge \big(\mathsf{aFresh}(\ell) \implies \exists \ell' \in \mathsf{LSID} :: \mathsf{Samekey}(\ell', \ell)\big).$$

We then say that the AKE protocol π with identity sets $\mathcal{I}, \mathcal{S}$ provides full explicit key authentication *if, for all PPT adversaries $\mathcal{A}$,*

$$Adv_{\mathcal{A},\pi,\mathcal{I},\mathcal{S}}^{G_{\mathsf{fexKeyAuth}}}(1^{\lambda}) = \Pr\left[Exp_{\mathcal{A},\pi,\mathcal{I},\mathcal{S}}^{G_{\mathsf{fexKeyAuth}}}(1^{\lambda}) = 0\right] = \mathsf{negl}(\lambda)$$

where $G_{\mathsf{fexKeyAuth}}$ is defined similarly to G_{iKeyAuth}.

We see that a session's expectation of both authentication and confirmation appears as $\ell.\mathsf{pid} \in \mathcal{S}$ and $\ell.\mathsf{kconf} = \mathsf{full}$ in the predicate.

Definition 3.6 (Almost-full explicit key authentication). *The predicate* $\mathsf{afexKeyAuth}$ *evaluates to* 1 *if and only if*

$$\forall \ell \in \mathsf{LSID}, (\ell.\mathsf{pid} \in \mathcal{S} \wedge \ell.\mathsf{kconf} = \mathsf{almost} \wedge \ell.\mathsf{accept}) \implies \big(\forall \ell' \in \mathsf{LSID}, (\mathsf{Samekey}(\ell', \ell) \implies \ell'.\mathsf{id} = \ell.\mathsf{pid})\big) \wedge \Big(\mathsf{aFresh}(\ell) \implies \exists \ell' \in \mathsf{LSID} :: \big[\mathsf{Samekcid}(\ell', \ell) \wedge (\ell'.\mathsf{key} \neq \bot \implies \mathsf{Samekey}(\ell', \ell))\big]\Big).$$

We then say that the AKE protocol π with identity sets $\mathcal{I}, \mathcal{S}$ provides almost-full explicit key authentication *if, for all PPT adversaries $\mathcal{A}$,*

$$Adv_{\mathcal{A},\pi,\mathcal{I},\mathcal{S}}^{G_{\mathsf{afexKeyAuth}}}(1^\lambda) = \Pr\left[Exp_{\mathcal{A},\pi,\mathcal{I},\mathcal{S}}^{G_{\mathsf{afexKeyAuth}}}(1^\lambda) = 0\right] = \mathsf{negl}(\lambda)$$

where $G_{\mathsf{afexKeyAuth}}$ is defined similarly to G_{iKeyAuth}.

3.4 Equivalence results

We formally prove the coherence of the authentication definitions presented above. Namely, we show that a protocol which satisfies both implicit key authentication and key confirmation also satisfies explicit key authentication, and we show that the converse holds.

Theorem 3.1. *Let π be an AKE protocol; it holds for π that*

$$\mathsf{iKeyAuth} \wedge \mathsf{fKeyConf} \iff \mathsf{fexKeyAuth}, \tag{5}$$

$$\mathsf{iKeyAuth} \wedge \mathsf{afKeyConf} \iff \mathsf{afexKeyAuth}. \tag{6}$$

proof We first focus on equation (5) and show that $\mathsf{iKeyAuth} \wedge \mathsf{fKeyConf} \implies \mathsf{fexKeyAuth}$; we proceed by proving the contrapositive. Let $\mathcal{A}$ be a successful adversary against the $\mathsf{fexKeyAuth}$ predicate; i.e. $\mathcal{A}$ reaches an execution state where $\neg\mathsf{fexKeyAuth}$ holds true. This is equivalent to

$$\exists \ell^* \in \mathsf{LSID} :: \ell^*.\mathsf{pid} \in \mathcal{S} \wedge \ell^*.\mathsf{kconf} = \mathsf{full} \wedge \ell^*.\mathsf{accept} \wedge \Big(\big(\exists \ell' :: \mathsf{Samekey}(\ell', \ell^*) \wedge \ell'.\mathsf{id} \neq \ell^*.\mathsf{pid}\big) \vee \big(\mathsf{aFresh}(\ell^*) \wedge \forall \ell'', \neg\mathsf{Samekey}(\ell'', \ell^*)\big)\Big) \tag{7}$$

Thus if $\neg\mathsf{fexKeyAuth}$ holds true, either the first expression of the OR clause holds, which implies $\neg\mathsf{iKeyAuth}$, or the second one holds and implies $\neg\mathsf{fKeyConf}$. We therefore obtain that

$$\neg\mathsf{fexKeyAuth} \implies \neg\mathsf{iKeyAuth} \vee \neg\mathsf{fKeyConf} \tag{8}$$

which completes the first part of the proof.

We now show that $\mathsf{fexKeyAuth} \implies \mathsf{iKeyAuth} \wedge \mathsf{fKeyConf}$. We first show that $\mathsf{fexKeyAuth} \implies \mathsf{iKeyAuth}$. Let $\mathcal{A}$ be a successful adversary against the $\mathsf{iKeyAuth}$ predicate; i.e. $\mathcal{A}$ reaches an execution state where $\neg\mathsf{iKeyAuth}$ holds true which is equivalent to

$$\exists \ell^* :: \ell^*.\mathsf{pid} \in \mathcal{S} \wedge (\ell^*.\mathsf{kconf} = \mathsf{full}) \wedge \ell^*.\mathsf{accept} \wedge \exists \ell' :: \mathsf{Samekey}(\ell^*, \ell') \wedge (\ell'.\mathsf{id} \neq \ell^*.\mathsf{pid}). \tag{9}$$

Note that we include $\ell^*.\mathsf{kconf} = \mathsf{full}$ in $\neg\mathsf{iKeyAuth}$ as we only aim to prove that $\mathsf{fexKeyAuth}$ implies $\mathsf{iKeyAuth}$ for sessions that expect full explicit key authentication. We now assume, for contradiction, that $\mathsf{fexKeyAuth}$ holds; this implies

$$\forall \ell, (\mathsf{Samekey}(\ell, \ell^*) \implies \ell.\mathsf{id} = \ell.\mathsf{pid}) \wedge (\mathsf{aFresh}(\ell^*) \implies \exists \ell'' :: \mathsf{Samekey}(\ell^*, \ell'')) \tag{10}$$

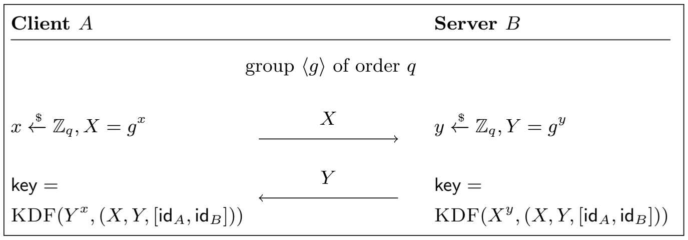

Client A
Server B
group $\langle g \rangle$ of order q
$x \xleftarrow{\$} \mathbb{Z}_q, X = g^x$
X
$y \xleftarrow{\$} \mathbb{Z}_q, Y = g^y$
$\mathsf{key} =$
$\mathrm{KDF}(Y^x, (X, Y, [\mathsf{id}_A, \mathsf{id}_B]))$
Y
$\mathsf{key} =$
$\mathrm{KDF}(X^y, (X, Y, [\mathsf{id}_A, \mathsf{id}_B]))$

Fig. 2: The plain Diffie-Hellman protocol with identifiers $\mathsf{sid} = \mathsf{kcid} = (\mathsf{id}_A, X, \mathsf{id}_B, Y)$. Identities id_A and id_B are known in advance; $[\mathsf{id}_A, \mathsf{id}_B]$ means that the identities are optional.

for ℓ^* as in (9). We see that the existence of ℓ' that holds from (9) contradicts the first condition of (10) which shows that $\mathsf{fexKeyAuth} \implies \mathsf{iKeyAuth}$ as expected from the formulation of the predicates.

We now show that $\mathsf{fexKeyAuth} \implies \mathsf{fKeyConf}$. Let $\mathcal{A}$ be a successful adversary against the $\mathsf{fKeyConf}$ predicate; i.e. $\mathcal{A}$ reaches an execution state where $\neg\mathsf{fKeyConf}$ holds true. This is equivalent to

$$\exists \ell^* :: \mathsf{aFresh}(\ell^*) \wedge \ell^*.\mathsf{pid} \in \mathcal{S} \wedge (\ell^*.\mathsf{kconf} = \mathsf{full}) \wedge \ell^*.\mathsf{accept} \wedge \forall \ell, \neg\mathsf{Samekey}(\ell^*, \ell).$$

This ℓ^* is now exactly one that satisfies $\neg\mathsf{fexKeyAuth}$ and hence we immediately have that $\mathsf{fexKeyAuth} \implies \mathsf{fKeyConf}$. Since we have that $\mathsf{fexKeyAuth}$ implies both $\mathsf{iKeyAuth}$ and $\mathsf{fKeyConf}$, combined with (8) this concludes the proof that

$$\mathsf{iKeyAuth} \wedge \mathsf{fKeyConf} \iff \mathsf{fexKeyAuth}.$$

The proof of the same equivalence for almost-full confirmation notions, equation (6), follows from a similar argument. □

4 Protocol examples

In this section we present established protocols and study which of our authentication notions they achieve.

Our results confirm that a “rule of thumb” for protocol design to achieve implicit key authentication is to include the parties' identities in the key derivation step, $\mathsf{key} = \mathrm{KDF}(K, (\mathsf{id}_A, \mathsf{id}_B, \dots))$. If the key derivation function is collision-resistant then different identities immediately imply different session keys. Indeed this strategy has already been applied in very early protocols proposals, such as [BPR00], and has even been sometimes used to fix insecure protocols, e.g., [CBH05b]. We note that this method is also used in the TLS 1.3 protocol.

From the analysis of key confirmation in TLS 1.3 of [FGSW16], we see that a good strategy to obtain full or almost-full key confirmation is to send a MAC computed over a known value (such as the transcript) with a key derived from the same material as the final session key.

4.1 Plain Diffie–Hellman

We begin with the plain Diffie-Hellmann (DH) protocol, presented in Figure 2, in which the parties exchange g^x and g^y to derive a key from g^{xy} and the communication transcript. One may also use the identities in the key derivation. The exchanged elements live in a cyclic group $\mathbb{G}$ of prime order $|\mathbb{G}| = q$ with generator g such that $\langle g \rangle = \mathbb{G}$. We assume that this group is known to all parties. Since this is an unauthenticated protocol, we have $\mathcal{S} = \emptyset$.

Match security. The plain DH protocol provides Match security. Indeed both the session and key-confirmation identifiers fully determine the key (Properties 1 and 2). Furthermore, since the key-confirmation and session identifiers are identical, equal key-confirmation identifiers imply identical keys (Property 4). Finally, an honest party will contribute a random Diffie-Hellman share, such that the probability of matching any other share, is at most $n_s^2 \cdot \frac{1}{q}$ for a total number of n_s sessions, and thus negligible (Property 3).

Implicit key authentication. The plain DH protocol trivially provides implicit key authentication, as $\mathcal{S} = \emptyset$, but this is somewhat meaningless since this guarantee does not apply for any identity (again, as $\mathcal{S} = \emptyset$). We note that setting $\mathcal{S} \neq \emptyset$ would allow an adversary to break implicit key authentication as it could deroute messages to create a mismatch in the expected peer identities for any two sessions.

However, by including the identities in the key derivation function (as shown in Figure 2) this protocol can provide implicit key authentication even in the setting where $\mathcal{S} \neq \emptyset$. Indeed, the adversary has no control over a session's owner identity $\ell.\mathsf{id}$ which implies that if $\ell.\mathsf{pid} \neq \ell'.\mathsf{id}$ then $\ell.\mathsf{key} = \ell'.\mathsf{key}$ only if there is a collision in the KDF. In the random oracle model, or assuming a collision-resistant KDF, this happens only with negligible propability.

Key confirmation and explicit key authentication. As for implicit key authentication, this protocol also trivially provides key confirmation in a meaningless way since $\mathcal{S} = \emptyset$. We can show formally that setting $\mathcal{S} \neq \emptyset$ breaks key confirmation by taking an adversary which initiates a session with an honest session ℓ (either client or server), such that $\ell.\mathsf{pid} \in \mathcal{S}$, without initiating a matching partner session. It then creates the message g^x or g^y to complete the exchange with the honest party. Obviously, there is then no other session which holds the same key nor the same key-confirmation identifier thus contradicting the requirement for key confirmation.

Interestingly, Theorem 3.1 therefore implies that the DH protocol provides explicit key authentication *as long as no identity is expected to authenticate itself* (i.e. $\mathcal{S} = \emptyset$). The two attacks discussed above show that this no longer holds when $\mathcal{S} \neq \emptyset$. This example demonstrates that the definition of $\mathcal{S}$ within $\mathcal{I}$ is crucial in giving meaning to the various security guarantees.

4.2 HMQV

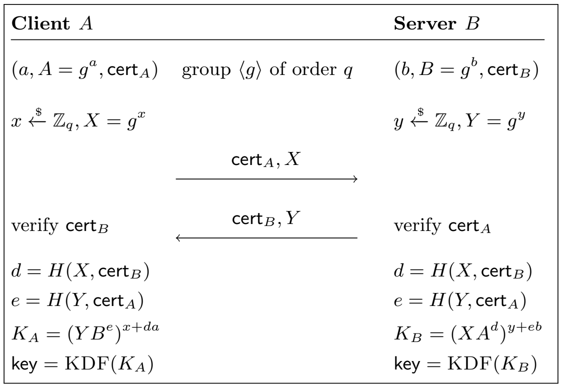
We next come to one of the most prominent candidates for implicitly authenticated key exchange, the HMQV protocol [Kra05]. The idea here is to run a DH key exchange and to mix Schnorr-type signatures under the parties' public keys in the key derivation. These signatures are not sent but only used locally, thus "implicitly" authenticating the key.

The protocol works over a group $\langle g \rangle = \mathbb{G}$ and uses a hash function H to compute the Schnorr signature. It is mutually authenticating, i.e. $\mathcal{S} = \mathcal{I}$, for which both parties use a long-term key. We assume that each party holds a certificate cert_i for its public key pk_i, and that the certificate is verified upon receiving it. We also assume that the public key and the owner's identity can be recovered from the certificate. We set $\mathsf{sid} = \mathsf{kcid} = (\mathsf{cert}_A, X, \mathsf{cert}_B, Y)$. Since key derivation in HMQV is also determined by the transcript and the hash function, Match security follows as in the plain DH case.

Implicit key authentication. We provide a proof that the HMQV protocol achieves our strong notion of implicit key authentication and is therefore secure against all possible UKS and KCI attacks. Recall that we need to show that

$$\forall \ell \in \mathsf{LSID}, (\ell.\mathsf{pid} \in \mathcal{S} \wedge \ell.\mathsf{accept}) \implies \forall \ell' \neq \ell \in \mathsf{LSID}, (\mathsf{Samekey}(\ell', \ell) \implies \ell'.\mathsf{id} = \ell.\mathsf{pid}) .$$

Due to the differences in the corruption queries that the adversary is allowed to make, the proof of secrecy for HMQV in [Kra05] is not immediately sufficient to imply our strong notion. Indeed, this proof holds only when the test-session in the secrecy experiment is fresh *in the sense of secrecy freshness* (see Section 5). An attack on implicit key authentication which would require the corruption of the owner before the session took place would therefore not be considered as valid against key secrecy and would not be ruled out by the proof of [Kra05].



Client A
Server B
$(a, A = g^a, \mathsf{cert}_A)$
group $\langle g \rangle$ of order q
$(b, B = g^b, \mathsf{cert}_B)$
$x \xleftarrow{\$} \mathbb{Z}_q, X = g^x$
$y \xleftarrow{\$} \mathbb{Z}_q, Y = g^y$
$\xrightarrow{\mathsf{cert}_A, X}$
verify cert_B
$\xleftarrow{\mathsf{cert}_B, Y}$
verify cert_A
$d = H(X, \mathsf{cert}_B)$
$e = H(Y, \mathsf{cert}_A)$
$K_A = (YB^e)^{x+da}$
$\mathsf{key} = \mathrm{KDF}(K_A)$
$d = H(X, \mathsf{cert}_B)$
$e = H(Y, \mathsf{cert}_A)$
$K_B = (XA^d)^{y+eb}$
$\mathsf{key} = \mathrm{KDF}(K_B)$

Fig. 3: HMQV protocol with session and key-confirmation identifiers $\mathsf{sid} = \mathsf{kcid} = (\mathsf{cert}_A, X, \mathsf{cert}_B, Y)$.

Proposition 4.1. *In the random oracle model, HMQV provides unconditional mutual implicit key authentication, with*

$$Adv^{G_{\mathsf{iKeyAuth}}}_{\mathcal{A},\mathrm{HMQV},\mathcal{I},\mathcal{I}}(1^\lambda) \leq \frac{n_i^2 \cdot n_s \cdot h}{2q} + \mathsf{negl}(\lambda),$$

where h is the number of queries made to H.

proof To break the $\mathsf{iKeyAuth}$ predicate, it must be that a session $\ell_A = (A, B, *)$ shares a key with a session $\ell_C = (C, D, *)$ where $C \neq B$. This can happen either if $K_A = K_C$, or if $K_A \neq K_C$ but $\mathrm{KDF}(K_A) = \mathrm{KDF}(K_C)$. The later implies a collision in the KDF and we assume that this happens only with negligible probability. The only freedom that $\mathcal{A}$ then has is to modify the Y value sent to ℓ_A as a response to its first message. Since the value of K_C, x, d and a are already fixed, $\mathcal{A}$ must choose a value of Y such that YB^e, where $e = H(Y, \mathsf{cert}_A)$ is exactly the right value such that $K_A = K_C$. Modelling H as a random oracle ensures that each value of Y yields a new random value of e and therefore that there is a probability of $1/q$ that a given value of Y will yield the correct value of YB^e. Given that there are at $n_i^2 \cdot n_s/2$ pairs of sessions, it holds that the adversary has at most a $n_i^2 \cdot n_s \cdot h/2q$ probability of finding a suitable Y for which the equality holds. □

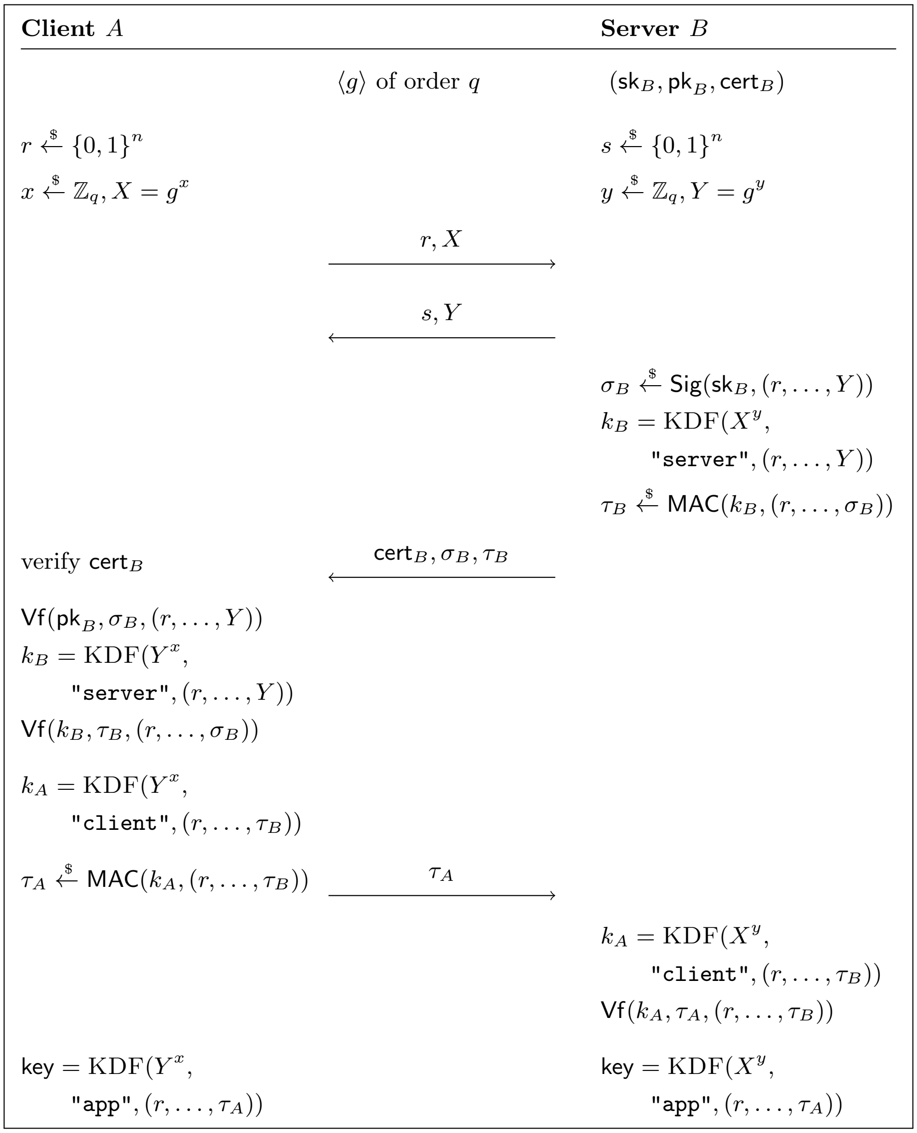
Similarly to the plain DH protocol, setting $\mathsf{key} = \mathrm{KDF}(K, (\mathsf{id}_A, \mathsf{id}_B))$ immediately provides implicit key authentication if the KDF is collision resistant.

Key confirmation and explicit key authentication. HMQV does not provide key confirmation in the same way that the plain DH does not. It immediately follows that the protocol does not provide explicit key authentication either.

4.3 TLS 1.3

We give a simplified version of the DH mode of the TLS 1.3 protocol suite in Figure 4 on page 16 which omits intermediate keys (e.g. handshake key and encryption of the handshake protocol). We also only look at server-only authentication.

Match security and implicit key authentication. TLS 1.3 is Match-secure; the argument is identical to the plain DH case and appears in [DFGS15]. Implicit key authentication follows as for the HMQV variant with identifiers in the KDF, if we assume that it is collision-resistant; as the server authenticates and cert_B appears in the key derivation, equal keys imply a correct authentication. Similarly to the HMQV protocol, key secrecy of TLS is not enough to imply implicit key authentication.



Client A		**Server** B
	$\langle g \rangle$ of order q	$(\mathsf{sk}_B, \mathsf{pk}_B, \mathsf{cert}_B)$
$r \xleftarrow{\$} \{0,1\}^n$		$s \xleftarrow{\$} \{0,1\}^n$
$x \xleftarrow{\$} \mathbb{Z}_q, X = g^x$		$y \xleftarrow{\$} \mathbb{Z}_q, Y = g^y$
	$\xrightarrow{r, X}$	
	$\xleftarrow{s, Y}$	
		$\sigma_B \xleftarrow{\$} \mathsf{Sig}(\mathsf{sk}_B, (r, \dots, Y))$ $k_B = \mathrm{KDF}(X^y,$ $\texttt{"server"}, (r, \dots, Y))$ $\tau_B \xleftarrow{\$} \mathsf{MAC}(k_B, (r, \dots, \sigma_B))$
verify cert_B	$\xleftarrow{\mathsf{cert}_B, \sigma_B, \tau_B}$	
$\mathsf{Vf}(\mathsf{pk}_B, \sigma_B, (r, \dots, Y))$ $k_B = \mathrm{KDF}(Y^x,$ $\texttt{"server"}, (r, \dots, Y))$ $\mathsf{Vf}(k_B, \tau_B, (r, \dots, \sigma_B))$		
$k_A = \mathrm{KDF}(Y^x,$ $\texttt{"client"}, (r, \dots, \tau_B))$		
$\tau_A \xleftarrow{\$} \mathsf{MAC}(k_A, (r, \dots, \tau_B))$	$\xrightarrow{\tau_A}$	
		$k_A = \mathrm{KDF}(X^y,$ $\texttt{"client"}, (r, \dots, \tau_B))$ $\mathsf{Vf}(k_A, \tau_A, (r, \dots, \tau_B))$
$\mathsf{key} = \mathrm{KDF}(Y^x,$ $\texttt{"app"}, (r, \dots, \tau_A))$		$\mathsf{key} = \mathrm{KDF}(X^y,$ $\texttt{"app"}, (r, \dots, \tau_A))$

Fig. 4: (Simplified) TLS 1.3 in mode (EC)DH, without handshake encryption and with server-only authentication. The session identifier and key-confirmation identifier are given by $\mathsf{sid} = \mathsf{kcid} = (r, X, S, Y, \mathsf{cert}_B)$. Notation $x, \dots, y$ means all transmitted communication data, ranging from x to y.

Key confirmation and explicit key authentication. Key confirmation for TLS 1.3 (`draft-10`) was shown in [FGSW16] and our version corresponds to this variant. The idea of the proof is that the parties use the key in the handshake protocol within the MAC and that his provides the required confirmation. Together with the argument that TLS 1.3 `draft-10` provides implicit key authentication, this shows that it also provides explicit key authentication according to our new definition, independently of any key secrecy guarantees; this was not the case in previous work [DFGS15].

5 Key Secrecy

We now define BR-style key secrecy with a security game and define the notion of Key-Match soundness.

5.1 BR-Style Key Secrecy

We recall the definition of secrecy for AKE protocols from [BFWW11]. The adversary is challenged, for a session of its choice, to distinguish between an honest key or a random one. To define the BR-secrecy game $G_{\mathsf{BRSec},\mathcal{D}}$, where $\mathcal{D}$ denotes the key distribution, we use the same execution, session and local session states as in Section 2. The model state contains two bits, b_{test} (initialised at random) and b_{guess} (intialised to $\perp$) along with a session identifier $\ell_{\mathsf{test}} \in \mathsf{LSID}$ (initialised to $\perp$). The identifier ℓ_{test} stores the object of the Test query (see below). The bit b_{test} determines whether the adversary receives the real key from ℓ_{test}, or a random value, in response to the Test query. The bit b_{guess} stores the adversary's guess.

There are two additional queries, Test and Guess. The $\mathsf{Test}(\ell)$ query sets $\ell_{\mathsf{test}} \leftarrow \ell$ and returns $\mathsf{key} = \ell.\mathsf{key}$ if $b_{\mathsf{test}} = 1$ or $\mathsf{key} \overset{\$}{\leftarrow} \mathcal{D}$ otherwise. The query $\mathsf{Guess}(b)$ sets $b_{\mathsf{guess}} \leftarrow b$. The **Valid** predicate requires that only one Test is made and to a session which has derived a key, and that only one Guess is made. The adversary may therefore chose to submit queries that will trivially allow him to win the game.

To catch this, if $\ell_{\mathsf{test}}.\delta_{\mathsf{ownr}} = \mathsf{corrupt}$ or $\ell_{\mathsf{test}}.\delta_{\mathsf{peer}} = \mathsf{corrupt}$, the sFresh predicate returns false. It also does so if ℓ_{test}, or any of its partners, has been revealed. We also take care of sessions which do not expect authentication as $\mathcal{A}$ may impersonate the unauthenticated party and learn the session key. Hence, sFresh also returns false if the intended partner $\ell_{\mathsf{test}}.\mathsf{pid}$ belongs to the set $\mathcal{I} \setminus \mathcal{S}$ of unauthenticated parties. However, there is one exception: if there exists an honest session which is partnered to ℓ_{test}, even if it does not belong to the intended partner, then the session took place between two honest session and the key is still expected to remain secret.

Definition 5.1 (Secrecy freshness). *For any $\ell \in \mathsf{LSID}$, the $\mathsf{sFresh}(\ell)$ predicate evaluates to true if and only if,*

$$\begin{aligned} &(\ell.\delta_{\mathsf{ownr}} = \ell.\delta_{\mathsf{peer}} = \mathsf{honest}) \ \wedge \ (\ell.\delta_{\mathsf{sess}} = \mathsf{fresh}) \\ &\wedge (\forall \ell' \in \mathsf{LSID}, \mathsf{Partner}(\ell', \ell) \implies \ell'.\delta_{\mathsf{sess}} = \mathsf{fresh}) \\ &\wedge \left[\ell.\mathsf{pid} \in \mathcal{I} \setminus \mathcal{S} \implies (\exists \ell' \in \mathsf{LSID} :: \mathsf{Partner}(\ell, \ell'))\right] \end{aligned}$$

The least strict requirement for sessions with unauthenticated parties would be to define almost-partnered sessions and allow these to be tested. We however refrain from introducing another identifier and keep the definition with partnering.

Definition 5.2 (BR-secrecy). *The BRSec predicate is defined differently from the authentication ones due to its distinguishing nature. Instead of returning 0 or 1 to signify whether a certain condition holds, the BRSec predicates evaluates to $\mathsf{MST}.b_{\mathsf{guess}}$ if and only if*

$$\mathsf{MST}.\ell_{\mathsf{test}} \neq \perp \ \wedge \ \mathsf{sFresh}(\mathsf{MST}.\ell_{\mathsf{test}})$$

and evaluates to $\perp$ otherwise. We also denote by $G^{b_{\mathsf{test}}}_{\mathsf{BRSec},\mathcal{D}}$ the secrecy game with a specific value for $\mathsf{b}_{\mathsf{test}}$. We then say that the AKE protocol π with identity sets $\mathcal{I}, \mathcal{S}$ is BR-secret w.r.t. output key distribution $\mathcal{D}$ if, for all PPT adversaries $\mathcal{A}$,

$$Adv^{G_{\mathsf{BRSec},\mathcal{D}}}_{\mathcal{A},\pi,\mathcal{I},\mathcal{S}}(1^\lambda) = \left| \Pr\left[Exp^{G^0_{\mathsf{BRSec},\mathcal{D}}}_{\mathcal{A},\pi,\mathcal{I},\mathcal{S}}(1^\lambda) = 1 \right] - \Pr\left[Exp^{G^1_{\mathsf{BRSec},\mathcal{D}}}_{\mathcal{A},\pi,\mathcal{I},\mathcal{S}}(1^\lambda) = 1 \right] \right|$$

is a negligible function in λ.

5.2 Key-Match Soundness

We now define the $\mathsf{KMSoundness}$ property which captures the essence of secrecy as a predicate without Test and Guess queries. It says that for any *authentication* fresh and accepting session ℓ, there does not exist another session ℓ' which holds the same key but is not partnered with ℓ.

Definition 5.3 (Key-Match Soundness). *The* $\mathsf{KMSoundness}$ *predicate evaluates to* 1 *if and only if*

$$\forall \ell \in \mathsf{LSID}, (\mathsf{aFresh}(\ell) \wedge \ell.\mathsf{pid} \in \mathcal{S} \wedge \ell.\mathsf{accept}) \implies \forall \ell' \in \mathsf{LSID} :: (\mathsf{Samekey}(\ell', \ell) \implies \mathsf{Partner}(\ell', \ell)).$$

We then say that the AKE protocol π with identity sets $\mathcal{I}, \mathcal{S}$ provides key-match soundness *if, for all PPT adversaries $\mathcal{A}$,*

$$Adv_{\mathcal{A},\pi,\mathcal{I},\mathcal{S}}^{G_{\mathsf{KMSoundness}}}(1^\lambda) = \Pr\left[Exp_{\mathcal{A},\pi,\mathcal{I},\mathcal{S}}^{G_{\mathsf{KMSoundness}}}(1^\lambda) = 0\right] = \mathsf{negl}(\lambda)$$

where $G_{\mathsf{KMSoundness}}$ is defined similarly to G_{iKeyAuth}.

The next theorem states that BR-secrecy and Match-security imply Key-Match soundness.

Theorem 5.1. *Let π be an AKE protocol with* Match *security and BR secrecy w.r.t. $\mathcal{D}$. Then it also provides key-match soundness. More precisely, for any PPT algorithm $\mathcal{A}$ attacking* $\mathsf{KMSoundness}$ *in at most n sessions, it holds that for some PPT algorithms $\mathcal{B}_1$, $\mathcal{B}_2$ and the output length $|\mathsf{key}|$ of keys,*

$$Adv_{\mathcal{A},\pi,\mathcal{I},\mathcal{S}}^{G_{\mathsf{KMSoundness}}}(1^\lambda) \le n^2 \cdot Adv_{\mathcal{B}_2,\pi,\mathcal{I},\mathcal{S}}^{G_{\mathsf{BRSec},\mathcal{D}}}(1^\lambda) + Adv_{\mathcal{B}_1,\pi,\mathcal{I},\mathcal{S}}^{G_{\mathsf{Match}}}(1^\lambda) + 2^{-|\mathsf{key}|}.$$

proof The first observation is that Match security implies that any partnered session to either ℓ or to ℓ' must hold the same key as the corresponding session. If this would not hold with overwhelming probability, we could build an algorithm $\mathcal{B}_1$ to refute Match security in a straightforward way. This enables us to assume that there are two sessions ℓ_0 and ℓ_0' with the above property and which accept first. That is, ℓ_0 and ℓ_0'

- hold identical keys, $\mathsf{Samekey}(\ell_0, \ell_0')$,
- are not partnered, $\neg\mathsf{Partner}(\ell_0, \ell_0')$, and
- for neither of the two sessions, in the moment when the session accepts, there is another session which is yet partnered with the session.

Note that ℓ_0 must have accepted by assumption, such that $\ell_0.\mathsf{key} \neq \perp$, and therefore $\mathsf{Samekey}(\ell_0, \ell_0')$ implies that session ℓ_0', too, must have a valid key. In particular it must have accepted (and can both be tested and revealed in an attack on secrecy).

Assume now that there was a successful adversary $\mathcal{A}$ against key-match soundness (with two sessions ℓ_0, ℓ_0' as above). We show how to break BRSec through an adversary $\mathcal{B}_2$ with non-negligible probability in this case.

Our adversary $\mathcal{B}_2$ will try to predict the sessions ℓ_0, ℓ_0' by picking two session numbers i, j at random from $\{1, 2, \dots, n\}$, where we count sessions according to their initialisation in $\mathcal{A}$'s simulated attack. Next, $\mathcal{B}_2$ runs $\mathcal{A}$'s attack, relaying all inputs and oracle queries and answers between $\mathcal{B}_2$'s game and $\mathcal{A}$. Note that $\mathcal{B}_2$ has the same oracle interfaces as $\mathcal{A}$, but in addition may call the Test and the Guess oracle.

Adversary $\mathcal{B}_2$ diverges from $\mathcal{A}$ with respect to two points: If the i-th session in the attack accepts, then $\mathcal{B}_2$ immediately asks to Reveal the session key key_i. If the j-the session accepts, then $\mathcal{B}_2$ immediately calls Test to get a key value key_j. Adversary $\mathcal{B}_2$ makes the $\mathsf{Guess}(b_{\mathsf{guess}})$ query and stops, where the bit b_{guess} is set to 1 if $\mathsf{key}_i = \mathsf{key}_j$, and to 0 otherwise.

Note that up to the point when $\mathcal{B}_2$ makes the Test query, the simulation to $\mathcal{A}$ is perfect. Assume that $\mathcal{B}_2$ predicts ℓ_0, ℓ_0' correctly for the Test query resp. the Reveal query, which happens with probability at least $1/n^2$. Then the Test session ℓ_0 does not have a partner yet, is authentication fresh and the authenticating partner is still honest, such that the session is still secrecy fresh. In this case, if $b_{\mathsf{test}} = 1$ the Test oracle returns the actual session key, such that the keys match, and we have $\mathcal{B}_2$ output 1, too. In summary, the probability of this happening is at least $\frac{1}{n^2}$ times the probability that $\mathcal{A}$ succeeds, minus a negligible term for refuting Match security. This is non-negligible.

Next consider the case $b_{\mathsf{test}} = 0$ such that the test session ℓ_0 returns a random key. Then the probability that this independent random key matches the other key is $2^{-|\mathsf{key}|}$ and thus negligible, such that $\mathcal{B}_2$ only returns 1 with this negligible probability. □

6 Relation with CK-style Security

In all CK-style models (CK, $\mathrm{CK}_{\mathrm{HMQV}}$ and eCK) the definition of "matching sessions" includes the requirement that parties agree on each other's identities. As there is no other mention of matching expected identities, this seems to be the only capture of authentication in such models.

Capturing authentication in CK-style models. In [Cre11], Cremers states that since "the test session-key must be indistinguishable from keys computed by non-matching sessions", then sessions with the same key must be matching sessions. This is analogous to our Theorem 5.1 concerning $\mathsf{KMSoundness}$ except that the CK-style definition of matching includes expected identities, and therefore this implies that CK-style security also guarantees (at least) implicit authentication.

Crucially, this argument however only applies to sessions for which CK-style security holds, that is sessions that remain $\mathsf{sFresh}_{\mathrm{CK}}$, for a suitable definition of this that captures the restriction of the CK-style models as presented in [Cre11]. In contrast, our standalone Definition 3.1 of implicit authentication captures security against a wider range of attacks due to the absence of freshness requirements on the target session.

Extending authentication freshness. To establish our authentication definitions of Section 3 in a CK-style model, we consider whether the freshness conditions change with the addition of a $\mathsf{StateReveal}$ query. This reveals to the adversary either the entire state or only the ephemeral key, as defined by the protocol, for the CK and eCK models respectively [Cre11].

Following from Section 3.1, we allow the adversary to make $\mathsf{StateReveal}$ queries against the $\mathsf{iKeyAuth}_{\mathrm{CK}}$ predicate to capture the widest possible range of attacks. This implies that $\mathcal{A}$ can both $\mathsf{StateReveal}$ a session and $\mathsf{Corrupt}$ its owner which is not allowed by CK-style models. Following similarly from Sections 3.2 and 3.3, we restrict the adversary from trivially obtaining the key when attacking key confirmation and explicit key authentication. Therefore we state that it cannot both corrupt the intended partner and also $\mathsf{StateReveal}$ a partner session against CK variants of these predicates.

Separating authentication from secrecy. Let $\mathcal{E}$ denote the event, in a CK-style secrecy experiment, where $\mathcal{A}$ succeeds in causing two game-controlled sessions to share a key without matching in the CK-style sense, i.e. without agreeing on each other's identities. Let G_{CK} denote the usual CK-style secrecy game and G^{-}_{CK} denote the game which instead uses our definition of partnering of Section 2.2 without expected identities (i.e. an extension of the secrecy game of Section 5.1). Using a rather informal terminology, we then have

$$\Pr\left[\mathcal{A} \text{ wins } G_{\mathsf{CK}}\right] = \Pr\left[\mathcal{A} \text{ wins } G_{\mathsf{CK}} | \mathcal{E}\right] \cdot \Pr\left[\mathcal{E}\right] + \Pr\left[\mathcal{A} \text{ wins } G_{\mathsf{CK}} | \neg\mathcal{E}\right] \cdot \Pr\left[\neg\mathcal{E}\right].$$

First, $\mathcal{E}$ corresponds to a break of $\mathsf{iKeyAuth}_{\mathrm{CK}}$ and so we have

$$\Pr\left[\mathcal{A} \text{ wins } G_{\mathsf{CK}} | \mathcal{E}\right] \cdot \Pr\left[\mathcal{E}\right] \leq \Pr\left[\neg\mathsf{iKeyAuth}_{\mathrm{CK}}\right].$$

Second, if $\mathcal{A}$ wins G_{CK} without triggering $\mathcal{E}$, then his attack can be reproduced in G^{-}_{CK} and so we have

$$\Pr\left[\mathcal{A} \text{ wins } G_{\mathsf{CK}} | \neg\mathcal{E}\right] \cdot \Pr\left[\neg\mathcal{E}\right] \leq \Pr\left[\mathcal{A} \text{ wins } G^{-}_{\mathsf{CK}}\right].$$

In conclusion,

$$\Pr\left[\mathcal{A} \text{ wins } G_{\mathsf{CK}}\right] \leq \Pr\left[\neg\mathsf{iKeyAuth}_{\mathrm{CK}}\right] + \Pr\left[\mathcal{A} \text{ wins } G^{-}_{\mathsf{CK}}\right],$$

which shows that we can represent CK-style key secrecy through our separate notions of authentication and key secrecy when adapted to these models.

7 Key-confirming protocols

We now define symmetric *key-confirming protocols.* These provide guarantees on the existence of a session with the same key. This guarantee may be secondary to the main purpose of these protocols. For example, we expect that protocols for authenticated message transmission would belong to this class.

Based on [BFWW11, Section 4], we describe the syntax and the security game for such protocols. We denote these as $\pi = (\mathsf{kg}, \zeta)$ and write $\mathcal{D}_{\mathsf{kg}}$ for the output distribution of the randomized algorithm kg; we use the mechanism of local session identifiers. Here, EST is not defined as there are no long-term keys.

Session state. For key-confirming protocols, it consists of:

- $\mathsf{crypt} \in \{0,1\}^*$: protocol-specific private session state.
- $\mathsf{key} \in \{0,1\}^* \cup \{\bot\}$: the symmetric key used.
- $\mathsf{kcind} \in \{\mathsf{true}, \mathsf{false}, \bot\}$: indicates if key confirmation is achieved. Initially set to $\bot$, it must be changed to true or false before termination. Its value is always public.

The difference with [BFWW11] is the addition of the *key confirmation indicator* kcind. We stress that setting kcind is done independently of termination. For example, a secure channel protocol could achieve key confirmation after the first messages but continue running for much longer as the channel is used for communication. We focus on guarantees on the setting of kcind and do not make assumptions or requirements on termination.

Local session state. As in [BFWW11], it consists of:

- $\delta_{\mathsf{key}} \in \{\mathsf{fresh}, \mathsf{revealed}\}$ denotes whether the key is known to the adversary.
- $\mathsf{lst} \in \{0,1\}^*$ is any other local session state required to model the protocol's other security requirements.

Setup. The setupE algorithm only initialises $\mathsf{crypt}, \mathsf{key}$ and kcind to $\bot$ for each $\ell \in \mathsf{LSID}$. The setupG algorithm also only initialises $\delta_{\mathsf{key}} \leftarrow \mathsf{fresh}$ for every session as our security game for key confirmation does not require any model-wide state.

Queries. As in [BFWW11], our model allows $\mathcal{A}$ to initialise sessions with three different queries. The first, $\mathsf{InitS}(\ell)$, initialises a session with an honestly generated key, $\ell.\mathsf{key} \leftarrow \mathsf{kg}(1^\lambda)$, which remains hidden from $\mathcal{A}$. The second, $\mathsf{InitP}(\ell_1, \ell_2)$, initialises a session with the same key as another. The game sets $\ell_2.\mathsf{key} \leftarrow \ell_1.\mathsf{key}$ and $\ell_2.\delta_{\mathsf{key}} \leftarrow \ell_1.\delta_{\mathsf{key}}$. The third, $\mathsf{InitK}(\ell, \kappa)$, allows $\mathcal{A}$ to set his own key. It sets $\ell.\mathsf{key} \leftarrow \kappa$ and immediately sets $\ell.\delta_{\mathsf{key}} \leftarrow \mathsf{revealed}$. As before, $\mathsf{Send}(\ell, m)$ and $\mathsf{Reveal}(\ell)$ allow $\mathcal{A}$ to control the network and view honestly generated keys.

The **Valid** predicate verifies that Send and Reveal queries are made to initialised sessions and that initialisation queries are made to sessions without keys. For the InitP query, it also verifies that ℓ_1 is initialised.

Key confirmation guarantee. Here there no longer is a distinction between full and almost-full key confirmation since keys are set upon initialisation. This notion says that for any session which has set the key confirmation identifier to true, there is another session which uses the same key.

Definition 7.1 (Key confirmation guarantee). *The* $\mathsf{symKeyConf}$ *predicate evaluates to* 1 *if and only if*

$$\forall \ell \in \mathsf{LSID}, (\ell.\delta_{\mathsf{key}} = \mathsf{fresh} \wedge \ell.\mathsf{kcind} = \mathsf{true}) \implies \exists \ell' \in \mathsf{LSID} :: \mathsf{Samekey}(\ell', \ell),$$

where $\mathsf{Samekey}$ *is defined as before. The game* $G_{\mathsf{symKeyConf}}$ *is then defined with state,* setupE*,* setupG *and behaviour as above, with query set* $Q = \{\mathsf{Send}, \mathsf{InitS}, \mathsf{InitP}, \mathsf{InitK}, \mathsf{Reveal}\}$ *and winning predicate* $P = \mathsf{symKeyConf}$.

The protocol π *provides* (secure) key confirmation*, or is a* key-confirming protocol*, if, for all PPT adversaries* $\mathcal{A}$*,*

$$Adv_{\mathcal{A},\pi,\mathcal{I}}^{G_{\mathsf{symKeyConf}}}(1^\lambda) = \Pr\left[Exp_{\mathcal{A},\pi,\mathcal{I}}^{G_{\mathsf{symKeyConf}}}(1^\lambda) = 0\right] = \mathsf{negl}(\lambda).$$

We note that a symmetric protocol π which always sets $\mathsf{kcind} = \mathsf{false}$ trivially achieves secure key confirmation. This is similar to an AKE protocol formally achieving implicit key authentication by setting $\mathcal{S} = \emptyset$.

```
$\pi_{\mathsf{kconf}}$ (with secret $\mathsf{key}$)
Initialise $\mathsf{kcind} \leftarrow \perp$ and role $\rho \leftarrow \perp$
while $\mathsf{kcind} = \perp$ do
  Receive $m^*$
  if $m^* = \mathsf{init}$ then
    Set $\rho \leftarrow \mathsf{init}$
    Send $t = \mathsf{MAC}(\mathsf{key}, 1)$
  elseif $\rho = \mathsf{init}$ then
    if $\mathsf{Vf}(\mathsf{key}, 2, m^*) = 1$ then
      Set $\mathsf{kcind} \leftarrow \mathsf{true}$
    else set $\mathsf{kcind} \leftarrow \mathsf{false}$
  else
    if $\mathsf{Vf}(\mathsf{key}, 1, m^*) = 1$ then
      Set $\mathsf{kcind} = \mathsf{true}$
      Send $t = \mathsf{MAC}(\mathsf{key}, 2)$
    else set $\mathsf{kcind} \leftarrow \mathsf{false}$
```

Fig. 5: A simple key confirmation protocol that expects a tag on the message "1" or "2" depending on the role (initiator or responder) played by the session.

Protocol example. We present an example of key-confirming protocols. Let $\mathcal{M} = (\mathsf{kg}, \mathsf{MAC}, \mathsf{Vf})$ be an unforgeable message authentication code (MAC) which requires that, for any PPT adversary $\mathcal{A}$,

$$\Pr\left[\mathsf{Vf}(\mathsf{key}, m, t) = 1; \begin{array}{c} \mathsf{key} \leftarrow \mathsf{kg}(\lambda), \\ (m, t) \leftarrow \mathcal{A}^{\mathsf{MAC}(\mathsf{key},\cdot)}(\lambda), \\ m \notin \mathcal{Q} \end{array}\right] \leq \mathsf{negl}(\lambda),$$

where $\mathsf{MAC}(\mathsf{key}, \cdot)$ denotes access to a tagging oracle and $\mathcal{Q}$ denotes the messages queried by $\mathcal{A}$ for tagging. From such a MAC, we construct the protocol π_{kconf} as follows. If a session is activated with $m = \mathsf{init}$, it sends $t = \mathsf{MAC}(\mathsf{key}, 1)$. When it receives a second message m^*, it verifies that it is a tag for the message "2" by checking if $\mathsf{Vf}(\mathsf{key}, 2, m^*) = 1$. If this holds, then it sets $\mathsf{kcind} \leftarrow \mathsf{true}$, otherwise it sets $\mathsf{kcind} \leftarrow \mathsf{false}$. If a session is instead activated as a receiver, then it plays the counterpart and checks that it correctly receives a tag for the message "1" and, if so, sets $\mathsf{kcind} \leftarrow \mathsf{true}$ and replies with a tag for "2". We present the formal description of π_{kconf} in Figure 5.

To show that the protocol π_{kconf} is key-confirming, we use the concept of single-session reducible games presented in [BFWW11]. Without specifying the formal details, we see that the independence or sessions in the game $G^{\pi_{\mathsf{kconf}}}_{\mathsf{symKeyConf}}$, apart from their potential partner, implies that this game is session restricted. Theorem 2 of [BFWW11, Appendix B] then gives us that $G^{\pi_{\mathsf{kconf}}}_{\mathsf{symKeyConf}}$ is single session reducible and therefore that the key-confirmation property of π_{kconf} depends only on the security of a single session.

We then reduce the security of one session to the unforgeability of the MAC $\mathcal{M}$. The reduction sets up a session and waits for the adversary to submit a message. If it submits $m^* = \mathsf{init}$, then the reduction uses the oracle $\mathsf{MAC}(\mathsf{key}, \cdot)$ to respond with a correct tag. If it submits any other message, then the reduction submits $(1, m^*)$ or $(2, m^*)$ as its forgery, depending on its role. If the reduction has queried the MAC oracle on "1" in response to an init message, then the adversary must create a tag for the message "2" to make π_{kconf} accept and therefore there is no risk of the reduction outputting a message which it has already queried. This shows that the reduction creates a forgery exactly when the adversary is capable of winning is $G^{\pi_{\mathsf{kconf}}}_{\mathsf{symKeyConf}}$ and thus π_{kconf} provides secure key confirmation if $\mathcal{M}$ is an unforgeable MAC.

While this protocol is a simple example that has no objective beyond providing secure key confirmation, it nonetheless provides justification for expecting useful constructions, such as authenticated encryption schemes of secure channel protocols, to provide the same guarantees.

8 Explicit authentication from key-confirming protocols

We now define the composition of AKE protocols with key-confirming protocols, similarly to [BFWW11, Section 5]. As a significant difference, we consider the composition *as an AKE protocol*, not as a symmetric protocol, and we prove that composing an AKE protocol with implicit key authentication and key secrecy together with a secure key-confirming protocol yields an *AKE protocol* with *explicit* key authentication.

Syntax of composed protocols. The composition first runs the AKE protocol and then, once this accepts, the symmetric protocol, initialised with the key from the first step, until key confirmation.

Recall that a key-confirming protocol $\pi = (\mathsf{kg}_\pi, \zeta_\pi)$ sets kcind before terminating and that it can then continue its execution; this is not the case when forming an AKE protocol since it must terminate upon acceptance of the key, as in Section 2. We therefore define $\bar{\pi}$ to be π with the algorithm $\zeta_{\bar{\pi}}$ the same as ζ_π but halted after kcind is set. Thus the key derived by ζ_{ke} is only accepted as the final key for the composition once $\zeta_{\bar{\pi}}$ has set kcind to true. Given an AKE protocol $\mathsf{ke} = (\mathsf{kg}_{\mathsf{ke}}, \zeta_{\mathsf{ke}})$, we write $\mathsf{ke};\bar{\pi} = (\mathsf{kg}_{\mathsf{ke};\bar{\pi}}, \zeta_{\mathsf{ke};\bar{\pi}})$ for the composition.

As we consider $\mathsf{ke};\bar{\pi}$ as an AKE protocol, it uses the same long-term key generation as ke and therefore $\mathsf{kg}_{\mathsf{ke};\bar{\pi}} = \mathsf{kg}_{\mathsf{ke}}$. The algorithm $\zeta_{\mathsf{ke};\bar{\pi}}$ first runs ζ_{ke}. If this rejects, then $\zeta_{\mathsf{ke};\bar{\pi}}$ rejects the session; otherwise it runs $\zeta_{\bar{\pi}}$ with the derived key and accepts or rejects depending on kcind.

Syntax of composed games. The game $G^{\mathsf{ke};\bar{\pi}}_{\mathsf{Pred}}$ enables $\mathcal{A}$ to interact with simultaneous sessions of the composed protocol. Here, the adversary attacks the authentication property Pred of $\mathsf{ke};\bar{\pi}$ seen as an AKE protocol. We build the game from the elements of the games for the protocols ke and π and use indices to distinguish them.

Game state. The execution state $\mathsf{EST}_{\mathsf{ke};\bar{\pi}}$ is $\mathsf{EST}_{\mathsf{ke}}$ as key-confirming protocols do not have one. The session state $\mathsf{SST}_{\mathsf{ke};\bar{\pi}}$ is made up of the same elements as for key exchange protocols but constructed from the composing session states as follows:

- The long-term keying information is as in $\mathsf{SST}_{\mathsf{ke}}$.
- The protocol private session state is the concatenation of both states: $\mathsf{crypt}_{\mathsf{ke};\bar{\pi}} = \mathsf{crypt}_{\mathsf{ke}} \| \mathsf{crypt}_{\bar{\pi}}$.
- The $\mathsf{accept}_{\mathsf{ke};\bar{\pi}}$ indicator is set to true once $\mathsf{kcind}_{\bar{\pi}}$ is set to true.
- The session identifier $\mathsf{sid}_{\mathsf{ke};\bar{\pi}}$ is set to $\mathsf{sid}_{\mathsf{ke}}$ *only* when $\mathsf{accept}_{\mathsf{ke};\bar{\pi}}$ is set to true.
- The $\mathsf{key}_{\mathsf{ke};\bar{\pi}}$ is set to $\mathsf{key}_{\mathsf{ke}}$ *only* when $\mathsf{accept}_{\mathsf{ke};\bar{\pi}}$ is set to true. Before then, $\mathsf{key}_{\mathsf{ke}}$ is kept internally and passed on to $\bar{\pi}$, so that $\mathsf{key}_{\bar{\pi}} \leftarrow \mathsf{key}_{\mathsf{ke}}$, when $\mathsf{accept}_{\mathsf{ke}} \leftarrow \mathsf{true}$.
- As $\bar{\pi}$ always provides full key confirmation, we have that $\mathsf{kconf}_{\mathsf{ke};\bar{\pi}} = \mathsf{true}$ for all sessions.

The local session state $\mathsf{LST}_{\mathsf{ke};\bar{\pi}}$ is the same as for AKE protocols and the model state remains undefined as not required for authentication.

Setup, queries and **Valid** *predicate.* These are the same as in Section 2 with the addition that the **Valid** predicate uses $\mathbf{Valid}_{\bar{\pi}}$ for Send and Reveal queries to sessions executing $\bar{\pi}$.

Winning predicates. Any from Sections 2 and 3.

Composition result. We show that a implicitly authenticated key exchange protocol composed with a key-confirming protocol produces an explicitly authenticated key exchange protocol. Our choice of public session identifiers means we do not require a session matching algorithm as in [BFWW11, Section 3].

Theorem 8.1. *Let* ke *be a* Match*-secure key exchange protocol which provides implicit key authentication and BR-secrecy w.r.t. key distribution* $\mathcal{D}$*. Let* π *be a symmetric-key protocol with key generation distribution* $\mathcal{D}$ *which provides secure key confirmation. Then* $\mathsf{ke};\bar{\pi}$ *is a key exchange protocol with provides explicit key authentication.*

proof We make use of Theorem 3.1 to separate our work into two steps. First we prove in Lemma 8.1 that $\mathsf{ke};\bar{\pi}$ provides implicit key authentication under the assumption that ke does. Then we prove in Lemma 8.2 that $\mathsf{ke};\bar{\pi}$ provides full key confirmation under the assumption that ke is BR-secret and that $\bar{\pi}$ provides secure key confirmation. As the two property hold separately, Theorem 3.1 immediately gives us that $\mathsf{ke};\bar{\pi}$ provides explicit key authentication. □

Lemma 8.1. *Let* ke *be a key exchange protocol and let* π *be a symmetric-key protocol. For any PPT adversary* $\mathcal{A}$*, it holds that, for some PPT algorithm* $\mathcal{B}$*,*

$$Adv^{G^{\mathsf{ke};\bar{\pi}}_{\mathsf{iKeyAuth}}}_{\mathcal{A},\mathsf{ke};\bar{\pi},\mathcal{I},\mathcal{S}}(1^{\lambda}) = Adv^{G^{\mathsf{ke}}_{\mathsf{iKeyAuth}}}_{\mathcal{B},\mathsf{ke},\mathcal{I},\mathcal{S}}(1^{\lambda}).$$

proof Let $\mathcal{A}$ be an adversary against $\mathsf{ke};\bar{\pi}$ in the $G^{\mathsf{ke};\bar{\pi}}_{\mathsf{iKeyAuth}}$ game, which we refer to $G^{\mathsf{ke};\bar{\pi}}$ when the context is clear. We build an adversary $\mathcal{B}$ against ke in the $G^{\mathsf{ke}}_{\mathsf{iKeyAuth}}$ game, which we similarly refer to G^{ke}.

$\mathcal{B}$ sets up $G^{\mathsf{ke};\bar{\pi}}$ for $\mathcal{A}$ as described above using elements from G^{ke}. It then responds to $\mathcal{A}$'s queries in the following way. (We use the notation $\ell_{\mathsf{ke};\bar{\pi}}$ to denote session identifiers used by $\mathcal{A}$ in $G^{\mathsf{ke};\bar{\pi}}$ and the notation ℓ_{ke} to denote the corresponding identifiers used by $\mathcal{B}$ in G^{ke}.)

- When $\mathcal{A}$ submits $\mathsf{Send}(\ell_{\mathsf{ke};\bar{\pi}}, m)$, $\mathcal{B}$ checks the value of $\ell_{\mathsf{ke};\bar{\pi}}.\mathsf{accept}$. If it is either true or false, $\mathcal{B}$ responds $\perp$ to $\mathcal{A}$ as the sessions has either already accepted or rejected. If it is still $\perp$, $\mathcal{B}$ examines the value of $\ell_{\mathsf{ke}}.\mathsf{accept}$.
  - If $\ell_{\mathsf{ke}}.\mathsf{accept} = \perp$, $\mathcal{B}$ submits $\mathsf{Send}(\ell_{\mathsf{ke}}, m)$ to G^{ke} and responds to $\mathcal{A}$ with m' returned by G^{ke}. If $\ell_{\mathsf{ke}}.\mathsf{sid}$ is set at that step, $\mathcal{B}$ sets $\ell_{\mathsf{ke};\bar{\pi}}.\mathsf{sid} \leftarrow \ell_{\mathsf{ke}}.\mathsf{sid}$. If $\ell_{\mathsf{ke}}.\mathsf{accept} \leftarrow \mathsf{false}$, $\mathcal{B}$ sets $\ell_{\mathsf{ke};\bar{\pi}}.\mathsf{accept} \leftarrow \mathsf{false}$ and makes this known to $\mathcal{A}$. If $\ell_{\mathsf{ke}}.\mathsf{accept} \leftarrow \mathsf{true}$, $\mathcal{B}$ submits $\mathsf{Reveal}(\ell_{\mathsf{ke}})$ to G^{ke} to obtain $\ell_{\mathsf{ke}}.\mathsf{key}$.
  - If $\ell_{\mathsf{ke}}.\mathsf{accept} = \mathsf{true}$ and $\ell_{\bar{\pi}}.\mathsf{kcind} = \perp$, $\mathcal{B}$ has obtained $\ell_{\mathsf{ke}}.\mathsf{key}$ so it can respond to $\mathcal{A}$ according to $\bar{\pi}$ by computing the response internally. If $\ell_{\bar{\pi}}.\mathsf{kcind} \leftarrow \mathsf{false}$, $\mathcal{B}$ sets $\ell_{\mathsf{ke};\bar{\pi}}.\mathsf{accept} \leftarrow \mathsf{false}$. If $\ell_{\bar{\pi}}.\mathsf{kcind} \leftarrow \mathsf{true}$, $\mathcal{B}$ sets $\ell_{\mathsf{ke};\bar{\pi}}.\mathsf{accept} \leftarrow \mathsf{true}$ and $\ell_{\mathsf{ke};\bar{\pi}}.\mathsf{key} \leftarrow \ell_{\mathsf{key}}.\mathsf{key}$.
  - The case of $\ell_{\mathsf{ke}}.\mathsf{accept} = \mathsf{false}$ is never examined as it would already hold that $\ell_{\mathsf{ke};\bar{\pi}}.\mathsf{accept} = \mathsf{false}$.
- When $\mathcal{A}$ submits $\mathsf{Reveal}(\ell_{\mathsf{ke};\bar{\pi}})$, $\mathcal{B}$ checks the value of $\ell_{\mathsf{ke};\bar{\pi}}.\mathsf{accept}$. If it is either $\perp$ or false, $\mathcal{B}$ responds $\perp$ to $\mathcal{A}$ as his Reveal query is invalid. If it is true, then $\ell_{\mathsf{ke};\bar{\pi}}.\mathsf{key}$ was set when $\ell_{\mathsf{ke};\bar{\pi}}.\mathsf{accept} \leftarrow \mathsf{true}$ so $\mathcal{B}$ responds with $\ell_{\mathsf{ke};\bar{\pi}}.\mathsf{key}$ to $\mathcal{A}$ and sets $\ell_{\mathsf{ke};\bar{\pi}}.\delta_{\mathsf{sess}} \leftarrow \mathsf{revealed}$.
- When $\mathcal{A}$ submits $\mathsf{Corrupt}(i)$, $\mathcal{B}$ submits $\mathsf{Corrupt}(i)$ to G^{ke} and receives sk_i which it returns to $\mathcal{A}$. At that moment, G^{ke} will mark the values of $\ell_{\mathsf{ke}}.\delta_{\mathsf{ownr}}$ and $\ell_{\mathsf{ke}}.\delta_{\mathsf{peer}}$ as $\mathsf{corrupt}$ for relevant ℓ_{ke} as decribed in Section 2. However, $\mathcal{B}$ will not update the corresponding sessions in $G^{\mathsf{ke};\bar{\pi}}$ in the same way as this would leak information to $\mathcal{A}$ about the internal stage of the sessions. Instead, $\mathcal{B}$ marks the values as corrupt for the sessions $\ell_{\mathsf{ke};\bar{\pi}}$ which have not completed the entire composed protocol, even if they have already completed the key exchange protocol and would not be marked as corrupt in G^{ke}.

We now argue that if $\mathcal{A}$ is able to reach an execution state in $G^{\mathsf{ke};\bar{\pi}}$ for which the $\mathsf{iKeyAuth}$ predicate evaluates to 0, then $\mathcal{B}$, by behaving as described above, reaches a state in G^{ke} for which the $\mathsf{iKeyAuth}$ predicate also evaluates to 0. This means that the composed protocol preserves implicit key authentication. If $\mathcal{A}$ reaches such a state, then we have that

$$\begin{aligned}\exists \ell_{\mathsf{ke};\bar{\pi}} \in \mathsf{LSID}_{\mathsf{ke};\bar{\pi}} :: (&\ell_{\mathsf{ke};\bar{\pi}}.\mathsf{pid} \in \mathcal{S} \wedge \ell_{\mathsf{ke};\bar{\pi}}.\mathsf{accept}) \\ &\wedge \big(\exists \ell'_{\mathsf{ke};\bar{\pi}} \in \mathsf{LSID}_{\mathsf{ke};\bar{\pi}} :: \mathsf{Samekey}(\ell'_{\mathsf{ke};\bar{\pi}}, \ell_{\mathsf{ke};\bar{\pi}}) \\ &\wedge \ell'_{\mathsf{ke};\bar{\pi}}.\mathsf{id} \neq \ell_{\mathsf{ke};\bar{\pi}}.\mathsf{pid}\big).\end{aligned}$$

We show that this also holds for the corresponding sessions ℓ_{ke} and ℓ'_{ke} in G^{ke}. We first have that $\ell_{\mathsf{ke}}.\mathsf{pid} \in \mathcal{S}$ as all the sessions and the set $\mathcal{S}$ match one-to-one between the two games. We then have that $\ell_{\mathsf{ke}}.\mathsf{accept} = \mathsf{true}$

as $\ell_{\mathsf{ke};\bar{\pi}}.\mathsf{accept}$ is set to true only if the ke session accepts, and $\mathcal{B}$ relays $\mathcal{A}$'s Send queries exactly which causes ℓ_{ke} to accept in G^{ke}.

As per the definition of the composed protocol $\mathsf{ke};\bar{\pi}$, the final key is fixed as soon as the ke part completes, therefore it holds that if two sessions accept with the same key in $G^{\mathsf{ke};\bar{\pi}}$, then they have derived that same key in the first part. As $\mathcal{B}$ relays $\mathcal{A}$'s queries exactly, we have that $\mathsf{Samekey}(\ell'_{\mathsf{ke}}, \ell_{\mathsf{ke}})$ holds in G^{ke} for the sessions corresponding to $\ell'_{\mathsf{ke};\bar{\pi}}$ and $\ell_{\mathsf{ke};\bar{\pi}}$. Furthermore, $\ell'_{\mathsf{ke}}.\mathsf{id} \neq \ell_{\mathsf{ke}}.\mathsf{pid}$ also holds as these values are the same as the ones for the sessions in $G^{\mathsf{ke};\bar{\pi}}$. This shows that the following holds for the corresponding sessions:

$$\exists \ell_{\mathsf{ke}} \in \mathsf{LSID}_{\mathsf{ke}} :: (\ell_{\mathsf{ke}}.\mathsf{pid} \in \mathcal{S} \wedge \ell_{\mathsf{ke}}.\mathsf{accept}) \\ \wedge (\exists \ell'_{\mathsf{ke}} \in \mathsf{LSID}_{\mathsf{ke}} :: \mathsf{Samekey}(\ell'_{\mathsf{ke}}, \ell_{\mathsf{ke}}) \wedge \ell'_{\mathsf{ke}}.\mathsf{id} \neq \ell_{\mathsf{ke}}.\mathsf{pid}),$$

which implies that $\mathcal{B}$ is successful for the game $G^{\mathsf{ke}}_{\mathsf{iKeyAuth}}$ exactly when $\mathcal{A}$ is succesful for the game $G^{\mathsf{ke};\bar{\pi}}_{\mathsf{iKeyAuth}}$. □

Lemma 8.2. *Let* ke *be a* Match*-secure key exchange protocol with output key distribution* $\mathcal{D}$*. Let* π *be a symmetric-key protocol with key generation distribution* $\mathcal{D}$*. Let* $n = n_i^2 \cdot n_s$*. For any PPT adversary* $\mathcal{A}$*, it holds that*

$$Adv_{\mathcal{A},\mathsf{ke};\bar{\pi},\mathcal{I},\mathcal{S}}^{G^{\mathsf{ke};\bar{\pi}}_{\mathsf{fKeyConf}}}(1^\lambda) \leq n \cdot Adv_{\mathcal{B}_1,\mathsf{ke},\mathcal{I},\mathcal{S}}^{G_{\mathsf{BRSec},\mathcal{D}}}(1^\lambda) + Adv_{\mathcal{B}_2,\pi,\mathcal{I}}^{G_{\mathsf{symKeyConf}}}(1^\lambda),$$

for some PPT algorithms $\mathcal{B}_1$ *and* $\mathcal{B}_2$*.*

proof We use a strategy similar to the proof of Theorem 1 in [BFWW11], namely we first replace all the keys derived by the ke part of the composed protocol by randomly sampled keys from the correct distribution, using BR-secrecy to show that the final game is indistinguishable from the first. Then we show, similarly to Lemma 8.1, that if an adversary manages to break the key confirmation property of the composed protocol, then a reduction can break the key confirmation property of the symmetric protocol π.

To replace all the keys used, we proceed with a hybrid argument. Let the game $G^{\mathsf{ke};\bar{\pi},\Sigma,\mathcal{D}}_{\mathsf{fKeyConf}}$ be the game G_{fKeyConf} played against protocol $\mathsf{ke};\bar{\pi}$, where the first Σ sessions to accept a new key, i.e. where a partner session has not already accepted a key, have their keys from ke replaced by a random value from $\mathcal{D}$ for the π part, where $\mathcal{D} = \mathcal{D}_{\mathsf{kg}}$ is the output distribution of the key generation algorithm for π. We remove the mention of $\mathsf{fKeyConf}$ when the context is clear. The original game G_{fKeyConf} for $\mathcal{A}$ is therefore $G^{\mathsf{ke};\bar{\pi},0,\mathcal{D}}$ where only honestly computed keys are used for π.

The game $G^{\mathsf{ke};\bar{\pi},\Sigma,\mathcal{D}}$ runs just as $G^{\mathsf{ke};\bar{\pi}}_{\mathsf{fKeyConf}}$ does with the following modifications. It maintains a counter σ to keep track of the number of new keys that are accepted (not counting those which the adversary might already know by corrupting one of the parties); this is set to 0 initially. The behaviour of $G^{\mathsf{ke};\bar{\pi},\Sigma,\mathcal{D}}$ is then the same as $G^{\mathsf{ke};\bar{\pi}}$ with the following differences to the $\mathsf{Send}(\ell_{\mathsf{ke};\bar{\pi}}, m)$ query:

- If $\sigma \geq \Sigma$, behave as in $G^{\mathsf{ke};\bar{\pi}}$, otherwise:
- If ℓ_{ke} has accepted already, simulate the π part honestly with $\ell_{\bar{\pi}}.\mathsf{key}$;
- Compute the response and the state update according to the ke algorithm;
- If $\ell_{\mathsf{ke}}.\mathsf{accept} \leftarrow \mathsf{true}$:
    - If there exists an $\ell'_{\mathsf{ke};\bar{\pi}} \in \mathsf{LSID}$ such that $\mathsf{Partner}(\ell_{\mathsf{ke};\bar{\pi}}, \ell'_{\mathsf{ke};\bar{\pi}}) = \mathsf{true}$ and $\ell'_{\mathsf{ke}}.\mathsf{accept} = \mathsf{true}$, then set $\ell_{\pi}.\mathsf{key} \leftarrow \ell'_{\pi}.\mathsf{key}$;
    - If there does not exist such an $\ell'_{\mathsf{ke};\bar{\pi}}$ that is partnered and whose ke part has already accepted, but either $\ell_{\mathsf{ke};\bar{\pi}}.\delta_{\mathsf{ownr}} = \mathsf{corrupt}$ or $\ell_{\mathsf{ke};\bar{\pi}}.\delta_{\mathsf{peer}} = \mathsf{corrupt}$ then set $\ell_{\pi}.\mathsf{key} \leftarrow \ell_{\mathsf{ke}}.\mathsf{key}$;
    - If both identities are still honest, and no partner session exists or has already accepted a ke key, then set $\ell_{\pi}.\mathsf{key} \xleftarrow{\$} \mathcal{D}$ and update $\sigma \leftarrow \sigma + 1$.

With this new behaviour, we have that the first Σ new keys that are unknown to the adversary at the time of their acceptance are replaced with keys sampled from $\mathcal{D}$ for the π part of the protocol.

Lemma 8.3 now allows us to change the game $G^{\mathsf{ke};\bar{\pi},0,\mathcal{D}}$ into the game $G^{\mathsf{ke};\bar{\pi},n,\mathcal{D}}$ for $n = n_i^2 \cdot n_s$ where the indinstinguishability of the two games is guaranteed by the BR-secrecy of the ke protocol. This yields

$$\left| \mathrm{Adv}_{\mathcal{A},\mathsf{ke};\bar{\pi},\mathcal{I},\mathcal{S}}^{G^{\mathsf{ke};\bar{\pi},0,\mathcal{D}}_{\mathsf{fKeyConf}}}(1^\lambda) - \mathrm{Adv}_{\mathcal{A},\mathsf{ke};\bar{\pi},\mathcal{I},\mathcal{S}}^{G^{\mathsf{ke};\bar{\pi},n,\mathcal{D}}_{\mathsf{fKeyConf}}}(1^\lambda) \right| \leq n \cdot \mathrm{Adv}_{\mathcal{B}_1,\mathsf{ke},\mathcal{I},\mathcal{S}}^{G^{\mathcal{D}}_{\mathsf{BRSec}}}(1^\lambda),$$

for a first reduction $\mathcal{B}_1$. In Lemma 8.4, we then show that key confirmation of the composed protocol follows from key confirmation of the symmetric-key protocol:

$$\mathrm{Adv}_{\mathcal{A},\mathsf{ke};\bar{\pi},\mathcal{I},\mathcal{S}}^{G_{\mathsf{fKeyConf}}^{\mathsf{ke};\bar{\pi},n,\mathcal{D}}}(1^\lambda) = \mathrm{Adv}_{\mathcal{B}_2,\pi,\mathcal{I}}^{G_{\mathsf{symKeyConf}}}(1^\lambda)$$

for a second reduction $\mathcal{B}_2$. This allows us to conclude that

$$\mathrm{Adv}_{\mathcal{A},\mathsf{ke};\bar{\pi},\mathcal{I},\mathcal{S}}^{G_{\mathsf{fKeyConf}}^{\mathsf{ke};\bar{\pi}}}(1^\lambda) \leq n \cdot \mathrm{Adv}_{\mathcal{B}_1,\mathsf{ke},\mathcal{I},\mathcal{S}}^{G_{\mathsf{BRSec},\mathcal{D}}}(1^\lambda) + \mathrm{Adv}_{\mathcal{B}_2,\pi,\mathcal{I}}^{G_{\mathsf{symKeyConf}}}(1^\lambda).$$

□

Lemma 8.3. *Let* ke *be a* Match-*secure key exchange protocol with output key distribution* $\mathcal{D}$. *Let* π *be a symmetric-key protocol with key generation distribution* $\mathcal{D}$. *For* $\Sigma = 1, \ldots, n_i^2 \cdot n_s$ *and for any PPT adversary* $\mathcal{A}$, *we have*

$$Adv_{\mathcal{A},\mathsf{ke};\bar{\pi},\mathcal{I},\mathcal{S}}^{G_{\mathsf{fKeyConf}}^{\mathsf{ke};\bar{\pi},\Sigma-1,\mathcal{D}}}(1^\lambda) \leq Adv_{\mathcal{A},\mathsf{ke};\bar{\pi},\mathcal{I},\mathcal{S}}^{G_{\mathsf{fKeyConf}}^{\mathsf{ke};\bar{\pi},\Sigma,\mathcal{D}}}(1^\lambda) + Adv_{\mathcal{B},\mathsf{ke},\mathcal{I},\mathcal{S}}^{G_{\mathsf{BRSec},\mathcal{D}}}(1^\lambda),$$

for some PPT algorithm $\mathcal{B} = \mathcal{B}(\Sigma)$.

proof Given an adversary $\mathcal{A}$ against the game $G_{\mathsf{fKeyConf}}^{\mathsf{ke};\bar{\pi},\Sigma-1,\mathcal{D}}$, we construct an algorithm $\mathcal{B}$ against the game $G_{\mathsf{BRSec},\mathcal{D}}$. The reduction $\mathcal{B}$ sets up the game for $\mathcal{A}$ as described at the beginning of this section and keeps track of the internal variable of each of the stages of the protocol. It also initialises $\sigma \leftarrow 0$.

As $\mathcal{A}$ runs, $\mathcal{B}$ responds to a $\mathsf{Send}(\ell_{\mathsf{ke};\bar{\pi}}, m)$ query as follows. (We recall that $\ell_{\mathsf{ke};\bar{\pi}}$ refers here to the variables of $G_{\mathsf{fKeyConf}}^{\mathsf{ke};\bar{\pi},\Sigma,\mathcal{D}}$ simulated by $\mathcal{B}$ to $\mathcal{A}$, ℓ_{ke} refers here to the variables of $G_{\mathsf{BRSec},\mathcal{D}}$ played by $\mathcal{B}$ and that $\ell_{\bar{\pi}}$ refers to the variables for the execution of $\bar{\pi}$ simulated by $\mathcal{B}$.)

- If $\ell_{\mathsf{ke};\bar{\pi}}.\mathsf{accept} \in \{\mathsf{true}, \mathsf{false}\}$, $\mathcal{B}$ returns $\bot$ to $\mathcal{A}$; otherwise:
- If $\ell_{\mathsf{ke}}.\mathsf{accept} = \bot$, $\mathcal{B}$ submits $\mathsf{Send}(\ell_{\mathsf{ke}}, m)$ to $G_{\mathsf{BRSec},\mathcal{D}}$ and receives an updated state for ℓ_{ke} and a response m'. If $\ell_{\mathsf{ke}}.\mathsf{sid}$ is set, $\mathcal{B}$ sets $\ell_{\mathsf{ke};\bar{\pi}}.\mathsf{sid} \leftarrow \ell_{\mathsf{ke}}.\mathsf{sid}$ for $\mathcal{A}$. If $\ell_{\mathsf{ke}}.\mathsf{accept} \leftarrow \mathsf{false}$, $\mathcal{B}$ sets $\ell_{\mathsf{ke};\bar{\pi}}.\mathsf{accept} \leftarrow \mathsf{false}$ and informs $\mathcal{A}$. If $\ell_{\mathsf{ke}}.\mathsf{accept} \leftarrow \mathsf{true}$, the following takes place:
  - If $\sigma = \Sigma$, $\nexists \ell'_{\mathsf{ke}} \in \mathsf{LSID}_{\mathsf{ke}} :: \mathsf{Partner}(\ell_{\mathsf{ke}}, \ell'_{\mathsf{ke}})$ and $\ell_{\mathsf{ke};\bar{\pi}}.\delta_{\mathsf{ownr}} \neq \mathsf{corrupt}$ and $\ell_{\mathsf{ke};\bar{\pi}}.\delta_{\mathsf{peer}} \neq \mathsf{corrupt}$, then
    - Submit $\mathsf{Test}(\ell)$ to $G_{\mathsf{BRSec},\mathcal{D}}$ and receive $\mathsf{key}_{\mathsf{ke}}$.
    - Set $\ell_{\bar{\pi}}.\mathsf{key} \leftarrow \mathsf{key}_{\mathsf{ke}}$.
    - Update $\sigma \leftarrow \sigma + 1$.
  - Else, if $\sigma \leq \Sigma$ then
    - If there does not exist $\ell'_{\mathsf{ke}} \in \mathsf{LSID}_{\mathsf{ke}}$ such that $\mathsf{Partner}(\ell_{\mathsf{ke}}, \ell'_{\mathsf{ke}})$ and $\ell_{\mathsf{ke};\bar{\pi}}.\delta_{\mathsf{ownr}} \neq \mathsf{corrupt} \neq \ell_{\mathsf{ke};\bar{\pi}}.\delta_{\mathsf{peer}}$, then sample a random key $\mathsf{key}_\pi \xleftarrow{\$} \mathcal{D}$ and set $\ell_{\bar{\pi}}.\mathsf{key} \leftarrow \mathsf{key}_\pi$. Update $\sigma \leftarrow \sigma + 1$.
    - Else, if $\nexists \ell'_{\mathsf{ke}} \in \mathsf{LSID}_{\mathsf{ke}} :: \mathsf{Partner}(\ell_{\mathsf{ke}}, \ell'_{\mathsf{ke}})$ and either $\ell_{\mathsf{ke};\bar{\pi}}.\delta_{\mathsf{ownr}} = \mathsf{corrupt}$ or $\ell_{\mathsf{ke};\bar{\pi}}.\delta_{\mathsf{peer}} = \mathsf{corrupt}$, then submit the query $\mathsf{Reveal}(\ell_{\mathsf{ke}})$ to $G_{\mathsf{BRSec},\mathcal{D}}$ and receive $\mathsf{key}_{\mathsf{ke}}$. Then set $\ell_{\bar{\pi}}.\mathsf{key} \leftarrow \mathsf{key}_{\mathsf{ke}}$.
    - Else, there exists an $\ell'_{\mathsf{ke}} \in \mathsf{LSID}_{\mathsf{ke}} :: \mathsf{Partner}(\ell_{\mathsf{ke}}, \ell'_{\mathsf{ke}})$ for which $\ell'_{\bar{\pi}}.\mathsf{key}$ has already been set. Then set $\ell_{\bar{\pi}}.\mathsf{key} \leftarrow \ell'_{\bar{\pi}}.\mathsf{key}$.
  - Else $\sigma > \Sigma$ so perform the following:
    - If $\exists \ell'_{\mathsf{ke}} \in \mathsf{LSID}_{\mathsf{ke}} :: \mathsf{Partner}(\ell_{\mathsf{ke}}, \ell'_{\mathsf{ke}})$ then set $\ell_{\bar{\pi}}.\mathsf{key} \leftarrow \ell'_{\bar{\pi}}.\mathsf{key}$.
    - Else submit the query $\mathsf{Reveal}(\ell_{\mathsf{ke}})$ to $G_{\mathsf{BRSec},\mathcal{D}}$, receive $\mathsf{key}_{\mathsf{ke}}$ and set $\ell_{\bar{\pi}}.\mathsf{key} \leftarrow \mathsf{key}_{\mathsf{ke}}$. Update $\sigma \leftarrow \sigma + 1$.

If $\mathcal{A}$ submits a $\mathsf{Reveal}(\ell_{\mathsf{ke};\bar{\pi}})$ query, $\ell_{\mathsf{ke};\bar{\pi}}$ must have accepted for it to be valid. Therefore $\mathcal{B}$ has already manually set the internal key $\ell_{\bar{\pi}}.\mathsf{key}$ and it can return it to $\mathcal{A}$ consistently.

If $\mathcal{A}$ submits a $\mathsf{Corrupt}(i)$ query, $\mathcal{B}$ marks all relevant sessions $\ell_{\mathsf{ke};\bar{\pi}} \in \mathsf{LSID}_{\mathsf{ke};\bar{\pi}}$ as $\mathsf{corrupt}$ if they are still running and then submits $\mathsf{Corrupt}(i)$ to $G_{\mathsf{BRSec},\mathcal{D}}$ to receive sk_i and return it to $\mathcal{A}$.

In the processing of a Send query, when $\sigma > \Sigma$ and there is an existing partner session, we initialise the key directly from the partner session's. As we assume that ke is Match-secure, these two partner sessions will derive the same key with overwhelming probability.

We note that if the Test query returns the real key, then $\mathcal{B}$ will perfectly simulate $G_{\mathsf{fKeyConf}}^{\mathsf{ke};\bar{\pi},\Sigma-1,\mathcal{D}}$ to $\mathcal{A}$, but if it returns a random key from $\mathcal{D}$, then $\mathcal{B}$ will perfectly simulate $G_{\mathsf{fKeyConf}}^{\mathsf{ke};\bar{\pi},\Sigma,\mathcal{D}}$. When $\mathcal{A}$ terminates and was successful, $\mathcal{B}$ submits $\mathsf{Guess}(1)$ to $G_{\mathsf{BRSec},\mathcal{D}}$; it submits $\mathsf{Guess}(0)$ otherwise. The advantage of $\mathcal{B}$ in $G_{\mathsf{BRSec},\mathcal{D}}$ therefore corresponds to the difference in the success probability of $\mathcal{A}$ as we have

$$\Pr\left[\mathrm{Exp}_{\mathcal{B},\mathsf{ke},\mathcal{I},\mathcal{S}}^{G^0_{\mathsf{BRSec},\mathcal{D}}}(1^\lambda)=1\right]=\mathrm{Adv}_{\mathcal{A},\mathsf{ke};\bar{\pi},\mathcal{I},\mathcal{S}}^{G_{\mathsf{fKeyConf}}^{\mathsf{ke};\bar{\pi},\Sigma,\mathcal{D}}}(1^\lambda)$$

and

$$\Pr\left[\mathrm{Exp}_{\mathcal{B},\mathsf{ke},\mathcal{I},\mathcal{S}}^{G^1_{\mathsf{BRSec},\mathcal{D}}}(1^\lambda)=1\right]=\mathrm{Adv}_{\mathcal{A},\mathsf{ke};\bar{\pi},\mathcal{I},\mathcal{S}}^{G_{\mathsf{fKeyConf}}^{\mathsf{ke};\bar{\pi},\Sigma-1,\mathcal{D}}}(1^\lambda)$$

which gives

$$\mathrm{Adv}_{\mathcal{B},\mathsf{ke},\mathcal{I},\mathcal{S}}^{G_{\mathsf{BRSec},\mathcal{D}}}(1^\lambda)=\left|\mathrm{Adv}_{\mathcal{A},\mathsf{ke};\bar{\pi},\mathcal{I},\mathcal{S}}^{G_{\mathsf{fKeyConf}}^{\mathsf{ke};\bar{\pi},\Sigma-1,\mathcal{D}}}(1^\lambda)-\mathrm{Adv}_{\mathcal{A},\mathsf{ke};\bar{\pi},\mathcal{I},\mathcal{S}}^{G_{\mathsf{fKeyConf}}^{\mathsf{ke};\bar{\pi},\Sigma,\mathcal{D}}}(1^\lambda)\right|$$

and yields the desired result. □

Lemma 8.4. *Let* ke *be a* Match*-secure key exchange protocol with output key distribution* $\mathcal{D}$ *and* π *be a symmetric-key protocol with key generation distribution* $\mathcal{D}$*. Let* $n=n_i^2\cdot n_s$*. For any PPT adversary* $\mathcal{A}$*, it holds that*

$$Adv_{\mathcal{A},\mathsf{ke};\bar{\pi},\mathcal{I},\mathcal{S}}^{G_{\mathsf{fKeyConf}}^{\mathsf{ke};\bar{\pi},n,\mathcal{D}}}(1^\lambda)=Adv_{\mathcal{B},\pi,\mathcal{I}}^{G_{\mathsf{symKeyConf}}}(1^\lambda),$$

for some PPT algorithm $\mathcal{B}$*.*

proof Similarly to the proof of Lemma 8.1, we build a reduction $\mathcal{B}$ against π in $G_{\mathsf{symKeyConf}}$ which uses an adversary against $\mathsf{ke};\bar{\pi}$ in $G_{\mathsf{fKeyConf}}^{\mathsf{ke};\bar{\pi},n,\mathcal{D}}$, which we refer to as $\mathsf{G}^{\mathsf{ke};\bar{\pi},n}$ in this proof for simplicity.

The algorithm $\mathcal{B}$ sets up $G^{\mathsf{ke};\bar{\pi},n}$ for $\mathcal{A}$ by simulating all the elements relevant to the ke stage of the composed protocol. It then responds to $\mathcal{A}$'s queries as follows (we once again use $\ell_{\mathsf{ke};\bar{\pi}}$ to refer to identifiers used by $\mathcal{A}$, ℓ_{ke} for corresponding identifiers simulated internally by $\mathcal{B}$ and ℓ_π for those used by $\mathcal{B}$ in $G_{\mathsf{symKeyConf}}$.

- When $\mathcal{A}$ submits $\mathsf{Send}(\ell_{\mathsf{ke};\bar{\pi}},m)$: if $\ell_{\mathsf{ke};\bar{\pi}}.\mathsf{accept}\in\{\mathsf{true},\mathsf{false}\}$, $\mathcal{B}$ returns $\perp$ to $\mathcal{A}$. Otherwise:
  - If $\ell_{\mathsf{ke}}.\mathsf{accept}=\perp$, $\mathcal{B}$ simulates the execution of ke. If $\ell_{\mathsf{ke}}.\mathsf{accept}\leftarrow\mathsf{false}$, $\mathcal{B}$ sets $\ell_{\mathsf{ke};\bar{\pi}}.\mathsf{accept}\leftarrow\mathsf{false}$. If $\ell_{\mathsf{ke}}.\mathsf{accept}\leftarrow\mathsf{true}$, $\mathcal{B}$ leaves $\ell_{\mathsf{ke}}.\mathsf{key}=\perp$ and then:
    - ∗ If $\ell_{\mathsf{ke}}.\delta_{\mathsf{peer}}=\mathsf{honest}\wedge\nexists\ell'_{\mathsf{ke}}\in\mathsf{LSID}_{\mathsf{ke}}::(\mathsf{Partner}(\ell_{\mathsf{ke}},\ell'_{\mathsf{ke}})\wedge\ \ell'_{\mathsf{ke}}.\mathsf{accept}=\mathsf{true})$, then $\mathcal{B}$ submits $\mathsf{InitS}(\ell_\pi)$ to $G_{\mathsf{symKeyConf}}$.
    - ∗ If $\ell_{\mathsf{ke}}.\delta_{\mathsf{peer}}=\mathsf{corrupt}\wedge\nexists\ell'_{\mathsf{ke}}\in\mathsf{LSID}_{\mathsf{ke}}::(\mathsf{Partner}(\ell_{\mathsf{ke}},\ell'_{\mathsf{ke}})\wedge\ \ell'_{\mathsf{ke}}.\mathsf{accept}=\mathsf{true})$, then $\mathcal{B}$ submits $\mathsf{InitS}(\ell_\pi)$ and then $\mathsf{Reveal}(\ell_\pi)$ to $G_{\mathsf{symKeyConf}}$ to generate and obtain key_π which it saves by setting $\ell_{\mathsf{ke}}.\mathsf{key}\leftarrow\mathsf{key}_\pi$.
    - ∗ If $\exists\ell'_{\mathsf{ke}}\in\mathsf{LSID}_{\mathsf{ke}}::(\mathsf{Partner}(\ell_{\mathsf{ke}},\ell'_{\mathsf{ke}})\wedge\ell'_{\mathsf{ke}}.\mathsf{accept}=\mathsf{true})$, then $\mathcal{B}$ submits $\mathsf{InitP}(\ell'_\pi,\ell_\pi)$ to $G_{\mathsf{symKeyConf}}$ and sets $\ell_{\mathsf{ke}}.\mathsf{key}\leftarrow\ell'_{\mathsf{ke}}.\mathsf{key}$.
  - If $\ell_{\mathsf{ke}}.\mathsf{accept}=\mathsf{true}$, $\mathcal{B}$ submits $\mathsf{Send}(\ell_\pi,m)$ to $G_{\mathsf{symKeyConf}}$ and returns the reply to $\mathcal{A}$. If $\ell_\pi.\mathsf{kcind}\leftarrow\mathsf{false}$, $\mathcal{B}$ sets $\ell_{\mathsf{ke};\bar{\pi}}.\mathsf{accept}\leftarrow\mathsf{false}$. If $\ell_\pi.\mathsf{kcind}\leftarrow\mathsf{true}$, $\mathcal{B}$ sets $\ell_{\mathsf{ke};\bar{\pi}}.\mathsf{accept}\leftarrow\mathsf{true}$ and sets $\ell_{\mathsf{ke};\bar{\pi}}.\mathsf{key}\leftarrow\ell_{\mathsf{ke}}.\mathsf{key}$.
- When $\mathcal{A}$ submits $\mathsf{Reveal}(\ell_{\mathsf{ke};\bar{\pi}})$: if $\ell_{\mathsf{ke};\bar{\pi}}.\mathsf{accept}\in\{\perp,\mathsf{false}\}$, $\mathcal{B}$ returns $\perp$ to $\mathcal{A}$. Otherwise:
  - If $\ell_{\mathsf{ke};\bar{\pi}}.\mathsf{key}\neq\perp$, $\mathcal{B}$ returns $\ell_{\mathsf{ke};\bar{\pi}}.\mathsf{key}$ to $\mathcal{A}$.
  - If $\ell_{\mathsf{ke};\bar{\pi}}.\mathsf{key}=\perp$, $\mathcal{B}$ submits $\mathsf{Reveal}(\ell_\pi)$ to $G_{\mathsf{symKeyConf}}$ to obtain key_π, sets $\ell_{\mathsf{ke};\bar{\pi}}.\mathsf{key}\leftarrow\mathsf{key}_\pi$ and sets $\ell'_{\mathsf{ke};\bar{\pi}}.\mathsf{key}\leftarrow\mathsf{key}_\pi$ for any $\ell'_{\mathsf{ke}}\in\mathsf{LSID}_{\mathsf{ke}}$ such that $\mathsf{Partner}(\ell_{\mathsf{ke}},\ell'_{\mathsf{ke}})\wedge\ell'_{\mathsf{ke}}.\mathsf{accept}=\mathsf{true}$.
- When $\mathcal{A}$ submits $\mathsf{Corrupt}(i)$: $\mathcal{B}$ marks all relevant sessions $\ell_{\mathsf{ke};\bar{\pi}}\in\mathsf{LSID}_{\mathsf{ke};\bar{\pi}}$ as $\mathsf{corrupt}$ (either δ_{ownr} or δ_{peer}) and returns sk_i to $\mathcal{A}$.

By processing each query as above, the algorithm $\mathcal{B}$ ensures that the first session that accepts the ke stage within a potential partnership pair is mapped to a new session in $G_{\mathsf{symKeyConf}}$ by an InitS query. If the peer of that session was already corrupt, then $\mathcal{B}$ submits a Reveal query so that this session is flagged as $\mathsf{revealed}$ in the game for π. If a session is the second to accept within a partnership pair at the ke stage, then $\mathcal{B}$

uses an InitP query to initialise it with the same key as it's partner and to give it the same value for δ_{key} within $G_{\mathsf{symKeyConf}}$. As we assume that ke is Match-secure, these two partner sessions will derive the same key with overwhelming probability. This, together with $\mathcal{B}$'s handling of $\mathcal{A}$'s Reveal and $\mathsf{Corrupt}$ queries ensures that every session for which $\mathcal{A}$ could trivially obtain the session key is immediately marked as revealed in $G_{\mathsf{symKeyConf}}$.

Furthermore, since the key derived by $\mathcal{B}$'s internal simulation of the ke stage is never used by the π stage, but instead replaced with a randomly generated key using an InitS query, $\mathcal{B}$ provides a perfect simulation of $G^{\mathsf{ke};\bar{\pi},n}$ to $\mathcal{A}$. Therefore, as $\mathcal{B}$ relays $\mathcal{A}$'s Send queries exactly, we see that if $\mathcal{A}$ wins against the $\mathsf{fKeyConf}$ predicate in $G^{\mathsf{ke};\bar{\pi},n}$ then $\mathcal{B}$ will also reach a state that wins against $G_{\mathsf{symKeyConf}}$. We therefore have

$$\mathrm{Adv}_{\mathcal{A},\mathsf{ke};\bar{\pi},\mathcal{I},\mathcal{S}}^{G_{\mathsf{fKeyConf}}^{\mathsf{ke};\bar{\pi},n,\mathcal{D}}}(1^{\lambda}) = \mathrm{Adv}_{\mathcal{B},\pi,\mathcal{I}}^{G_{\mathsf{symKeyConf}}}(1^{\lambda}).$$

□

9 Entity authentication

This second form of authentication does not involve the key in its security guarantees. Our *entity authentication* definitions are instead based on the $\mathsf{Partner}$ predicate, instead of $\mathsf{Samekey}$, and say that if two sessions terminate with the same sid, then they should agree on each other's identities. This corresponds to the intuitive notion of entity authentication where sessions obtain guarantees upon terminating and deriving an identifier.

This equivalence also shows that, similarly to including the identities in the key to ensure implicit key authentication, including the identities in the session identifiers ensures implicit entity authentication. It also suggests that involving the sids in a MAC is a good method for ensuring entity confirmation.

9.1 Implicit and full explicit entity authentication and confirmation

To adapt our definitions to entity authentication, we replace $\mathsf{Samekey}$ by $\mathsf{Partner}$. Below we present only the predicates for implicit entity authentication, entity confirmation and full explicit entity authentication; the full case requires the definition of the session state variable $\mathsf{econf} \in \{\mathsf{full}, \mathsf{almost}, \mathsf{no}, \bot\}$, indicating, analogously to kconf, which form of authentication a session expects. The almost-full case is more involved because it also requires entity confirmation identifiers, similarly to the kcid; for the sake of brevity, we do not include it here. The full definitions can be derived from the following predicates.

Implicit entity authentication: The $\mathsf{iEntAuth}$ predicate is defined as

$$\forall \ell \in \mathsf{LSID}, (\ell.\mathsf{pid} \in \mathcal{S} \wedge \ell.\mathsf{accept}) \implies \forall \ell' \in \mathsf{LSID}, (\mathsf{Partner}(\ell', \ell) \implies \ell'.\mathsf{id} = \ell.\mathsf{pid})\,.$$

Full entity confirmation: The $\mathsf{fexEntConf}$ predicate is defined as

$$\forall \ell \in \mathsf{LSID}, (\mathsf{aFresh}(\ell) \wedge \ell.\mathsf{econf} = \mathsf{full} \wedge \ell.\mathsf{pid} \in \mathcal{S}) \wedge \ell.\mathsf{accept}) \implies \exists \ell' \in \mathsf{LSID} :: \mathsf{Partner}(\ell', \ell).$$

Full explicit entity authentication: The $\mathsf{fexEntAuth}$ predicate is defined as

$$\forall \ell \in \mathsf{LSID}, \begin{pmatrix} \ell.\mathsf{pid} \in \mathcal{S} \wedge \ell.\mathsf{accept} \\ \wedge\, \ell.\mathsf{econf} = \mathsf{full} \end{pmatrix} \implies$$

$$(\forall \ell' \in \mathsf{LSID}, \mathsf{Partner}(\ell', \ell) \implies \ell'.\mathsf{id} = \ell.\mathsf{pid})$$
$$\wedge\, (\mathsf{aFresh}(\ell) \implies \exists \ell' \in \mathsf{LSID} :: \mathsf{Partner}(\ell', \ell)).$$

We note that implicit entity authentication is a very weak notion as the session terminates neither with an explicit guarantee nor with a secret element, such as a key, that it may use later to obtain a stronger guarantee. Separating the two different properties that constitute explicit authentication may be helpful to understand and guide protocol design. For example, with this separation in mind, a signature over the transcript sent at the end of an execution can be seen as providing entity confirmation, and therefore boosting implicit entity authentication to explicit authentication. A similar argument could show that password authentication over a secure channel can serve a similar purpose.

9.2 Key and entity authentication relationships

We now present the necessary conditions for key and entity authentication notions to be equivalent to one another. We use the "secrecy and match-security predicate" $\mathsf{KMSoundness}$ from Definition 5.3 to state the relationships in terms of predicates. Recall that this predicate holds (with overwhelming probability) for a Match-secure and BR-secret protocol. The results are intuitively compelling. Under the assumption that partnered sessions derive equal keys and equal keys can only be derived in partnered sessions, authentication guarantees obtained via keys are equivalent to those obtained directly via session identifiers.

Proposition 9.1. *Let π be an AKE protocol; it holds that*

$$\mathsf{iKeyAuth} \wedge \mathsf{Match} \Longrightarrow \mathsf{iEntAuth}, \tag{11}$$

$$\mathsf{fexKeyAuth} \wedge \mathsf{Match} \wedge \mathsf{KMSoundness} \Longrightarrow \mathsf{fexEntAuth}, \tag{12}$$

proof The first implication (11) can be seen as follows. Both predicates are identical, except that entity authentication uses the $\mathsf{Partner}$ predicate instead of $\mathsf{Samekey}$. Hence, a mismatch—in the sense that $\mathsf{iKeyAuth}$ holds but $\mathsf{iEntAuth}$ does not—can only occur if:

$$\exists \ell \in \mathsf{LSID} :: [((\ell.\mathsf{pid} \in \mathcal{S}) \wedge \ell.\mathsf{accept}) \wedge \exists \ell' \in \mathsf{LSID} :: (\neg\mathsf{Samekey}(\ell', \ell) \wedge \mathsf{Partner}(\ell', \ell))].$$

If the sessions ℓ, ℓ' would have the same key, and then they would also satisfy the identity requirement $\ell'.\mathsf{id} = \ell.\mathsf{pid}$, because of the $\mathsf{iKeyAuth}$ property.

Note that the partnering predicate stipulates that the session identifiers of ℓ and ℓ' are equal (and different from $\perp$). According to our specification of key exchange protocols this, in turn, implies that both sessions must have accepted and, if so, that they have set the keys to values different from $\perp$. But now we would get an immediate contradiction to Property (1) of Match security:

$$\exists \ell, \ell' \in \mathsf{LSID} :: \mathsf{Partner}(\ell, \ell') \wedge (\ell.\mathsf{key} \neq \perp \neq \ell'.\mathsf{key}) \wedge \neg\mathsf{Samekey}(\ell, \ell').$$

The second implication (12) follows similarly, there are two possibilities for a mismatch ($\mathsf{fexKeyAuth}$ holds, but $\mathsf{fexEntAuth}$ does not). Either the first property in the implication in $\mathsf{fexEntAuth}$, which also appears in the implicit definition, is false, in which case we get the same contradiction as before. Or the second property in the implication ($\mathsf{aFresh} \implies \exists \ell' :: \mathsf{Partner}(\ell', \ell)$) is false, although it holds in $\mathsf{fexKeyAuth}$ for the $\mathsf{Samekey}$ case. This means that

$$\exists \ell \in \mathsf{LSID} :: \Big(\mathsf{aFresh}(\ell) \wedge \ell.\mathsf{pid} \in \mathcal{S} \wedge \ell.\mathsf{accept} \\ \wedge \big[\exists \ell' \in \mathsf{LSID} :: \mathsf{Samekey}(\ell', \ell)\big] \wedge \big[\forall \ell'' \in \mathsf{LSID} :: \neg\mathsf{Partner}(\ell'', \ell)\big]\Big).$$

Note that this means that there will be a session ℓ' which has the same key as ℓ, and since ℓ has accepted it must be a valid key $\ell.\mathsf{key} \neq \perp$, but such that no other session is partnered with ℓ. This, however, contradicts the $\mathsf{KMSoundness}$ predicate. □

Proposition 9.2. *Let π be an AKE protocol; it holds that*

$$\mathsf{iKeyAuth} \Longleftarrow \mathsf{iEntAuth} \wedge \mathsf{Match} \wedge \mathsf{KMSoundness}, \tag{13}$$

$$\mathsf{fexKeyAuth} \Longleftarrow \mathsf{fexEntAuth} \wedge \mathsf{Match} \wedge \mathsf{KMSoundness} \tag{14}$$

proof We start with the first implication (13). Similar to the previous proposition one can show that any mismatch in the predicates implies

$$\exists \ell \in \mathsf{LSID} :: \big[\ell.\mathsf{pid} \in \mathcal{S} \wedge \ell.\mathsf{accept} \wedge \exists \ell' \in \mathsf{LSID} :: (\mathsf{Samekey}(\ell', \ell) \wedge \neg\mathsf{Partner}(\ell', \ell))\big].$$

This would also contradict the $\mathsf{KMSoundness}$ predicate.

The second implication (14) derives a contradiction as in the implicit case, or we can analogously to the other direction conclude that

$$\exists \ell \in \mathsf{LSID} :: \Big(\mathsf{aFresh}(\ell) \wedge \ell.\mathsf{pid} \in \mathcal{S} \wedge \ell.\mathsf{accept} \\ \wedge \big[\exists \ell' \in \mathsf{LSID} :: \mathsf{Partner}(\ell', \ell)\big] \wedge \big[\forall \ell'' \in \mathsf{LSID} :: \neg\mathsf{Samekey}(\ell'', \ell)\big]\Big).$$

This, of course, would contradict Property (1) of the Match predicate, since we would have partnered sessions ℓ, ℓ' which do not hold the same (valid) key $\ell.\mathsf{key} \neq \perp$. Here we use the fact that partnering implies non-trivial session identifiers, and if session ℓ' has set the identifier, it has accepted and set a key $\ell'.\mathsf{key} \neq \perp$, too. □

Finally, we show that implicit entity authentication together with (full) explicit entity confirmation is equivalent to full explicit entity authentication.

Proposition 9.3. *Let π be an AKE protocol; it holds that*

$$\mathsf{iEntAuth} \wedge \mathsf{fexEntConf} \Longleftrightarrow \mathsf{fexEntAuth}$$

The proof follows as in the case of key authentication.

Acknowledgments

This work was supported in part by CyberSecurity Research Flanders with reference number VR20192203, by the ERC Advanced Grant ERC-2015-AdG-IMPaCT and by the Defense Advanced Research Projects Agency (DARPA) and Space and Naval Warfare Systems Center, Pacific (SSC Pacific) under contract No. N66001-15-C-4070. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the ERC, DARPA or the US Government. Marc Fischlin's research work has been funded by the German Federal Ministry of Education and Research and the Hessen State Ministry for Higher Education, Research and the Arts within their joint support of the National Research Center for Applied Cybersecurity ATHENE.

References

[BFWW11] Christina Brzuska, Marc Fischlin, Bogdan Warinschi, and Stephen C. Williams. Composability of Bellare-Rogaway key exchange protocols. In Yan Chen, George Danezis, and Vitaly Shmatikov, editors, *ACM CCS 2011*, pages 51–62. ACM Press, October 2011.

[BJM97] Simon Blake-Wilson, Don Johnson, and Alfred Menezes. Key agreement protocols and their security analysis. In *Cryptography and Coding, 6th IMA International Conference, 1997, Proceedings*, volume 1355 of *LNCS*, pages 30–45. Springer, 1997.

[BPR00] Mihir Bellare, David Pointcheval, and Phillip Rogaway. Authenticated key exchange secure against dictionary attacks. In Bart Preneel, editor, *EUROCRYPT 2000*, volume 1807 of *LNCS*, pages 139–155. Springer, Heidelberg, May 2000.

[BR94] Mihir Bellare and Phillip Rogaway. Entity authentication and key distribution. In Douglas R. Stinson, editor, *CRYPTO'93*, volume 773 of *LNCS*, pages 232–249. Springer, Heidelberg, August 1994.

[BR95] Mihir Bellare and Phillip Rogaway. Provably secure session key distribution: The three party case. In *27th ACM STOC*, pages 57–66. ACM Press, May / June 1995.

[BSWW13] Christina Brzuska, Nigel P. Smart, Bogdan Warinschi, and Gaven J. Watson. An analysis of the EMV channel establishment protocol. In Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung, editors, *ACM CCS 2013*, pages 373–386. ACM Press, November 2013.

[BWM99] Simon Blake-Wilson and Alfred Menezes. Unknown key-share attacks on the station-to-station (STS) protocol. In Hideki Imai and Yuliang Zheng, editors, *PKC'99*, volume 1560 of *LNCS*, pages 154–170. Springer, Heidelberg, March 1999.

[CBH05a] Kim-Kwang Raymond Choo, Colin Boyd, and Yvonne Hitchcock. Examining indistinguishability-based proof models for key establishment protocols. In Bimal K. Roy, editor, *ASIACRYPT 2005*, volume 3788 of *LNCS*, pages 585–604. Springer, Heidelberg, December 2005.

[CBH05b] Kim-Kwang Raymond Choo, Colin Boyd, and Yvonne Hitchcock. On session key construction in provably-secure key establishment protocols. In *Progress in Cryptology - Mycrypt 2005, First International Conference on Cryptology in Malaysia, Kuala Lumpur, Malaysia, September 28-30, 2005, Proceedings*, volume 3715 of *Lecture Notes in Computer Science*, pages 116–131. Springer, 2005.

[CF12] Cas J. F. Cremers and Michele Feltz. Beyond eCK: Perfect forward secrecy under actor compromise and ephemeral-key reveal. In Sara Foresti, Moti Yung, and Fabio Martinelli, editors, *ESORICS 2012*, volume 7459 of *LNCS*, pages 734–751. Springer, Heidelberg, September 2012.

[CK01] Ran Canetti and Hugo Krawczyk. Analysis of key-exchange protocols and their use for building secure channels. In Birgit Pfitzmann, editor, *EUROCRYPT 2001*, volume 2045 of *LNCS*, pages 453–474. Springer, Heidelberg, May 2001.

[Cre11] Cas Cremers. Examining indistinguishability-based security models for key exchange protocols: The case of CK, CK-HMQV, and eCK. ASIACCS '11, pages 80–91, New York, NY, USA, 2011. ACM.

[DFGS15] Benjamin Dowling, Marc Fischlin, Felix Günther, and Douglas Stebila. A cryptographic analysis of the TLS 1.3 handshake protocol candidates. In Indrajit Ray, Ninghui Li, and Christopher Kruegel, editors, *ACM CCS 2015*, pages 1197–1210. ACM Press, October 2015.

[DvOW92] Whitfield Diffie, Paul C. van Oorschot, and Michael J. Wiener. Authentication and authenticated key exchanges. *Des. Codes Cryptography*, 2(2):107–125, 1992.

[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. IEEE Computer Society Press, May 2016.

[Kra05] Hugo Krawczyk. HMQV: A high-performance secure Diffie-Hellman protocol. In Victor Shoup, editor, *CRYPTO 2005*, volume 3621 of *LNCS*, pages 546–566. Springer, Heidelberg, August 2005.

[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*, volume 4784 of *LNCS*, pages 1–16. Springer, Heidelberg, November 2007.

[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*, pages 1343–1360. ACM Press, October / November 2017.

[MTI86] Tsutomu Matsumoto, Youichi Takashima, and Hideki Imai. On seeking smart public-key-distribution systems. *Transactions of the Institute of Electronics and Communication Engineers of Japan. Section E*, E69(2):99–106, 2 1986.

[MU08] Alfred Menezes and Berkant Ustaoglu. Comparing the pre- and post-specified peer models for key agreement. In Yi Mu, Willy Susilo, and Jennifer Seberry, editors, *ACISP 08*, volume 5107 of *LNCS*, pages 53–68. Springer, Heidelberg, July 2008.

[MvOV97] Alfred Menezes, Paul van Oorschot, and Scott Vanstone. *Handbook of Applied Cryptography.* CRC Press, 1997.

[RS09] Phillip Rogaway and Till Stegers. Authentication without elision: Partially specified protocols, associated data, and cryptographic models described by code. In John C Mitchell, editor, *CSF 2009Computer Security Foundations Symposium*, pages 26–39. IEEE Computer Society Press, 2009.

[Ust09] Berkant Ustaoglu. Comparing sessionstatereveal and ephemeralkeyreveal for Diffie-Hellman protocols. In Josef Pieprzyk and Fangguo Zhang, editors, *ProvSec 2009*, volume 5848 of *LNCS*, pages 183–197. Springer, Heidelberg, November 2009.

[Yan13] Zheng Yang. Modelling simultaneous mutual authentication for authenticated key exchange. In *Foundations and Practice of Security - 6th International Symposium, FPS 2013, La Rochelle, France, October 21-22, 2013, Revised Selected Papers*, volume 8352 of *Lecture Notes in Computer Science*, pages 46–62. Springer, 2013.