Generic Compilers for Authenticated Key Exchange*

Tibor Jager, Florian Kohlar, Sven Schäge, and Jörg Schwenk

Ruhr-University Bochum

Abstract. So far, all solutions proposed for *authenticated key agreement* combine key agreement and authentication into a single cryptographic protocol. However, in many important application scenarios, key agreement and entity authentication are clearly separated protocols. This fact enables efficient attacks on the naïve combination of these protocols. In this paper, we propose new compilers for two-party key agreement and authentication, which are provably secure in the standard Bellare-Rogaway model. The constructions are generic: key agreement is executed first and results (without intervention of the adversary) in a secret session key on both sides. This key (or a derived key) is handed over, together with a transcript of all key exchange messages, to the authentication protocol, where it is combined with the random challenge(s) exchanged during authentication.

Keywords: authenticated key agreement, protocol compiler, TLS

1 Introduction

Authenticated key agreement (AKE) is a basic building block in modern cryptography. Many secure protocols for two-party and group key agreement have been proposed, including generic compilers that transform simple key agreement protocols into authenticated key agreement protocols, with many additional security properties.

However, all known constructions (including e.g. the modular approach of [1], and the Katz-Yung compiler [22]) result in a single cryptographic protocol, whereas many security-critical *real-world* applications combine two or more clearly separated protocols:

- **(Client) Authentication and SSL/TLS.** The most prominent example is SSL/TLS. Although server and browser can be authenticated in a provably secure way [20, 25] within a single cryptographic protocol (the TLS handshake protocol), nearly all known web applications authenticate the client

* The research leading to these results has received funding from the European Community (FP7/2007-2013) under grant agreement number ICT-2007-216646 - European Network of Excellence in Cryptology II (ECRYPT II) and the Ministry of Economic Affairs and Energy of the State of North Rhine-Westphalia, grant 315-43-02/2-005-WFBO-009.

through a different protocol on top of the TLS channel. The security of these protocols is based on the sole assumption that the (human) user is able to authenticate the server on the basis of security indicators of the browser, which was shown to be false in [17]. We do not rely on this assumption. Instead, we regard SSL/TLS simply as a key agreement protocol, which cannot be changed due to the large number of implementations that are running worldwide. We may however change the authentication protocol, since the authentication protocol is often implemented in HTML/Javascript. [1]

– **Browser based Single Sign-On (SSO).** This scenario is perhaps the most complex one and a formalization is out of scope of this paper. However, it may serve as an illustration of how cryptographic protocols are combined today to implement key exchange (KE) and authentication functionalities. In SSO protocols, two key agreement protocols, and two different authentication protocols are combined to achieve the desired goal. Cryptographically secure SSO protocols have e.g. been described in [19].

In this work, we present a new compiler that handles these scenarios. Moreover, we can use our compiler to combine existing authentication protocols in a novel way with key exchange protocols. This includes:

– **Zero-Knowledge Authentication.** Zero-knowledge protocols have been developed with the goal to authenticate entities. However, in all known compilers, they cannot be combined with key agreement, except if they are transformed into digital signature schemes using the Fiat-Shamir heuristic. With our second compiler, ZK protocols can be used directly, which enables many interesting new protocols.
– **Privacy-preserving authentication.** With our compiler, we can easily combine privacy-preserving authentication protocols like Direct Anonymous Attestation with different key agreement protocols.

Man-in-the-middle Attack. Our real world attack scenario is as follows (cf. Figure 1): the adversary E ("Eve") acts as an active (wo)man-in-the-middle (MITM) between A and B during key exchange, and then acts as a passive "wire" during authentication. As a result, E has successfully authenticated as "A" towards B, and as "B" towards A, and shares (different) keys with A and B.

To counter this attack, one could of course apply standard cryptographic primitives to turn the key exchange protocol into an authenticated key exchange protocol (AKE) [1], but this is not possible in the cases cited above, *because the implementation of the KE protocol cannot be changed*, or the desired security

[1] At first glance, it seems that the security of TLS as a key agreement protocol could easily be proven in the Bellare-Rogaway model, since we only have to consider passive adversaries, and the TLS ciphersuites includes e.g. ephemeral Diffie-Hellman key exchange. However, there are some subtle problems with the `Reveal` query and the fact that the final `Finished` message of the TLS handshake is already encrypted. Therefore it is still unclear if TLS fits in our theoretical framework.

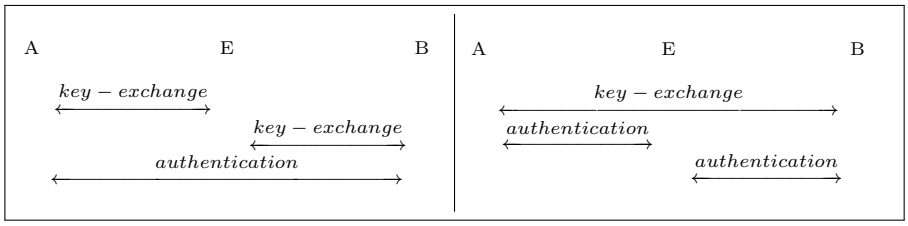

Fig. 1. Attack Scenario: Real world man-in-the-middle attack (left), and unknown key share attack (right)

goals (e.g. privacy) cannot be reached with standard compilers. Our compiler turns the combination of the two protocols into a provably secure AKE protocol. During compilation, only the authentication protocol is changed slightly.

Unknown Key Share (UKS) Attacks. To be able to prove the security in the standard Bellare-Rogaway (BR) model, the resulting AKE protocol must also be secure against unknown key share (UKS) attacks [14, 13] that do not directly lead to an attack in the real world, but invalidate security proofs in the model. Interestingly, in our scenario this is a kind of orthogonal attack to MITM attacks (cf. Figure 1): The adversary acts as a man-in-the-middle on the authentication protocol. To achieve security against both (MITM and UKS) attacks, one usually needs two compilers: One compiler who adds authenticators to each message [1], and one compiler who includes the complete state of the session into the computation of the session key [15]. Our compilers achieve this in one step, because we force the adversary to prove knowledge of the session key k through the derived key dk during authentication. Thus the adversary cannot authenticate to A or B without knowing k, and neither A nor B will accept.

Practical AKE protocols. If the two parties accept, they share a common state. This state consists of the secret key k, and the transcript of all messages sent and received. This transcript plays an important role in the BR model, since it defines the attack possibilities of the adversary. In practically relevant AKE protocols, a hash of this transcript is included in a final message secured with a MAC, to protect against MITM attacks.

The A&KE Compilers. To protect against MITM attacks in our generic scenario, it is sufficient to simply include the transcript of the KE protocol into the authentication protocol. (Many authentication protocols offer the possibility to authenticate arbitrary strings chosen by A od B, e.g. authentication protocols based on digital signatures, or the MAP2 protocol from [2].) Such a compiler protects against MITM attacks because (a) any modification of messages in the KE protocol automatically results in a modification of messages in the authentication protocol (since the transcript is included), which results in an abort of the authentication protocol if this protocol is secure in the BR model. Thus (b) the adversary is restricted to a passive role when attacking the KE protocol, but this protocol is by definition secure against passive adversaries.

Unfortunately, this simple compiler cannot be proven secure in the BR model, because the adversary also has access to the transcript of the protocol, and can use this in both instances of the authentication protocol (cf. right side of Fig. 1.) To avoid this attack, a secret value only known to A and B (i.e. the session key k) must be used in the authentication protocol in a generic way. There are at least two different methods (besides [15]) how to achieve this:

- An additional pair of messages can be sent after the KE and the authentication protocol. These messages contain a cryptographic checksum over the transcripts of both protocols. This checksum is basically a MAC, computed over the transcript of both the KE and the authentication protocol, using a key $K_{\mathsf{mac}} = \mathsf{PRF}(k, \text{“MAC”})$ derived from the key k returned by the KE protocol and some *pseudo-random function* PRF. The actual session key K returned by the compiled protocol (i.e., the value returned by a Reveal or Test query in the BR model) is also derived from k as $K = \mathsf{PRF}(k, \text{“KE”})$. In Section 3, we describe the compiler for this in detail, and prove its security in the standard model.
- Alternatively, we can modify a value that is present in *all* secure authentication protocols, in such a way that it does not change the security properties of the protocol:
  In a generic authentication protocol, a random challenge r_A guaranteeing the freshness of the message(s) must be sent from the challenger A to the prover B, which is answered with a response s_B from B. Ideally, this challenge is chosen from a large message space with uniform distribution. We assume that r_A is chosen uniformly from $\{0,1\}^t$, for some security parameter t. The answer $s_B := f(sk_B, r_A)$ is computed using the secret long-lived key sk_B of B, and the challenge r_A.
  Our compiler changes the computation of s_B slightly. Instead of using the challenge r_A directly, we use a derived value r'_A from the same distribution:

$$r'_A := \mathcal{H}(K_{\mathsf{mac}}, r_A, transcript_{KE}),\ s'_B := f(sk_B, r'_A),$$

  where $\mathcal{H}$ is some *hash function* modeled as a random oracle. Please note that r'_A is never sent (cf. Figure 3), but has to be computed by A and B. Thus the adversary E does not learn r'_A. This construction does not alter the security properties of the authentication protocol. In Section 4, we give a security proof for this compiler in the random oracle model.

1.1 Related Work

In their seminal papers [2, 1] on two-party authenticated key agreement, Bellare et al. started a line of research that has expanded in two directions: group key agreement [9], [8, 22, 10], and refined models to cover different types of attacks [11, 23, 24]. All these models cover concurrent execution of the protocol, and at least corruption of non-related session keys.

All models can roughly be classified in two groups: models that require a unique session ID before the start of the protocol, and models that construct

this session ID. [11] is the prototype of the former case: proofs and definitions are easier, but it is unclear how a session ID can be defined for practical applications. (E.g. in case of an SSL man-in-the-middle, browser and server do not share any common state.) Newer models like [24] or [23] thus avoid this assumption, and construct the session identifiers from the messages sent and received by the intended communication partners.

Unknown key share [5] attacks do not threaten the real world security of cryptographic protocols, but invalidate security proofs in the formal models that follow [2]: If the adversary is able to force two protocol participants into accepting the same session key, but without a matching conversation, a `Reveal` query to one of the participants will help to win the `Test` game against the other participant. Choo, Boyd and Hitchcock have shown how to invalidate security proofs of various protocols in the different models [14, 13], and how to fix the problem by including the whole session information in the computation of the session key [15]. They were able to compare the relative strengths of the different models assuming that session identifiers are constructed as a concatenation of the exchanged messages.

Canetti and Krawczyk in [12] consider a practically important protocol (IPSec IKE), which has a structure that places authentication after key exchange. Still, this is a single AKE protocol, and thus not comparable to our construction. In 2008 Morissey et al. studied the security of the TLS key agreement protocol [25] and provided a modular and generic proof of security for the established application keys.

Katz and Yung presented in [22] a first scalable compiler that transforms any passively secure group key-exchange protocol to an actively secure AKE. Their compiler adds one round and constant size (per user) to the original scheme, by appending an additional signature to each message of the protocol.

1.2 Contribution

In this paper, we describe two new compilers that allow us to combine key agreement protocols (which, in the BR model, need only be secure against passive adversaries) with arbitrary authentication protocols to form an authenticated key agreement (AKE) protocol in the sense of [2].

These compilers enable us to formally prove the security of real world protocols in the BR model, which was not possible before. The most important case here is TLS with an authentication protocol on top of the TLS channel, which can be proven secure if the authentication protocol is secure in the BR model. This is possible since we consider TLS only as a key agreement protocol, and not as an AKE protocol, and it seems likely that the security of (some ciphersuites of) TLS against passive adversaries can be proven.

Additionally, the compilers allow for a modular design of new AKE protocols, using existing protocols (e.g. TLS, IPSec IKE) or new ones (e.g. zero-knowledge authentication, group signatures). The formal security proof is simplified considerably, since the security of key agreement and authentication protocols can be proven separately, and our theorems yield the security of the combined protocol.

2 Preliminaries and Definitions

In this section, we recall the syntax and security definitions of the building blocks for our protocol compilers.

2.1 Digital Signature Schemes

A digital signature scheme is a triple $\Sigma = (\mathsf{SIG.Gen}, \mathsf{SIG.Sign}, \mathsf{SIG.Vfy})$, consisting of a key generation algorithm $(sk, pk) \xleftarrow{\$} \mathsf{SIG.Gen}(1^\kappa)$ generating a (public) verification key pk and a secret signing key sk on input of security parameter κ, signing algorithm $\sigma \xleftarrow{\$} \mathsf{SIG.Sign}(sk, m)$ generating a signature for message m, and verification algorithm $\mathsf{SIG.Vfy}(pk, m, \sigma)$ returning 1, if σ is a valid signature for m under key pk, and 0 otherwise.

Consider the following security experiment played between a challenger $\mathcal{C}$ and an adversary $\mathcal{A}$.

1. The challenger generates a public/secret key pair $(sk, pk) \xleftarrow{\$} \mathsf{SIG.Gen}(1^\kappa)$, the adversary receives pk as input.
2. The adversary may query arbitrary messages m_i to the challenger. The challenger replies each query with a signature $\sigma_i = \mathsf{SIG.Sign}(sk, m_i)$. Here i is an index, ranging between $1 \leq i \leq q$ for some polynomial $q = q(\kappa)$. Queries can be made adaptively.
3. Eventually, the adversary outputs a message/signature pair (m, σ).

Definition 1. *We say that Σ is secure against* existential forgeries under adaptive chosen-message attacks *(EUF-CMA), if*

$$\Pr\left[(m, \sigma) \xleftarrow{\$} \mathcal{A}^{\mathcal{C}}(1^\kappa, pk) : \mathsf{SIG.Vfy}(pk, m, \sigma) = 1 \wedge m \notin \{m_1, \ldots, m_q\}\right] \leq \epsilon.$$

for all probabilistic polynomial-time (in κ) adversaries $\mathcal{A}$, where $\epsilon = \epsilon(\kappa)$ is some negligible function in the security parameter.

2.2 Message Authentication Codes

A *message authentication code* is an algorithm MAC. This algorithm implements a deterministic function $w = \mathsf{MAC}(K_{\mathsf{mac}}, m)$, taking as input a (symmetric) key $K_{\mathsf{mac}} \in \{0,1\}^\kappa$ and a message m, and returning a string w.

Consider the following security experiment played between a challenger $\mathcal{C}$ and an adversary $\mathcal{A}$.

1. The challenger samples $K_{\mathsf{mac}} \xleftarrow{\$} \{0,1\}^\kappa$ uniformly random.
2. The adversary may query arbitrary messages m_i to the challenger. The challenger replies each query with $w_i = \mathsf{MAC}(K_{\mathsf{mac}}, m_i)$. Here i is an index, ranging between $1 \leq i \leq q$ for some polynomial $q = q(\kappa)$. Queries can be made adaptively.
3. Eventually, the adversary outputs a pair (m, w).

Definition 2. *We say that* MAC *is a* secure *message authentication code, if*

$$\Pr\left[(m,w) \xleftarrow{\$} \mathcal{A}^{\mathcal{C}}(1^\kappa) : w = \mathsf{MAC}(K_{\mathsf{mac}}, m) \wedge m \notin \{m_1, \ldots, m_q\}\right] \leq \epsilon$$

for all probabilistic polynomial-time (in κ*) adversaries* $\mathcal{A}$*, where* $\epsilon = \epsilon(\kappa)$ *is some negligible function in the security parameter.*

2.3 Pseudo-Random Functions

A *pseudo-random function* is an algorithm PRF. This algorithm implements a deterministic function $z = \mathsf{PRF}(k, x)$, taking as input a key $k \in \{0,1\}^\kappa$ and some bit string x, and returning a string $z \in \{0,1\}^\kappa$.

Consider the following security experiment played between a challenger $\mathcal{C}$ and an adversary $\mathcal{A}$.

1. The challenger samples $k \xleftarrow{\$} \{0,1\}^\kappa$ uniformly random.
2. The adversary may query arbitrary values x_i to the challenger. The challenger replies each query with $z_i = \mathsf{PRF}(k, x_i)$. Here i is an index, ranging between $1 \leq i \leq q$ for some polynomial $q = q(\kappa)$. Queries can be made adaptively.
3. Eventually, the adversary outputs value x and a special symbol $\top$. The challenger sets $z_0 = \mathsf{PRF}(k, x)$ and samples $z_1 \xleftarrow{\$} \{0,1\}^\kappa$ uniformly random. Then it tosses a coin $b \xleftarrow{\$} \{0,1\}$, and returns z_b to the adversary.
4. Finally, the adversary outputs a guess $b' \in \{0,1\}$.

Definition 3. *We say that* PRF *is a* secure *pseudo-random function, if*

$$|\Pr[b = b'] - 1/2| \leq \epsilon$$

for all probabilistic polynomial-time (in κ*) adversaries* $\mathcal{A}$*, where* $\epsilon = \epsilon(\kappa)$ *is some negligible function in the security parameter.*

2.4 Key Exchange Protocols

A (two-party) key-exchange protocol is a protocol executed among two parties A and B. At the end of the protocol, both A and B obtain the same key K_0 as the output of the protocol.

Definition 4. *We say that a key-exchange protocol is* passively-secure *if for all polynomial-time adversary holds that* $|\Pr[b = b'] - 1/2| \leq \epsilon$ *for some negligible function* ϵ *in the following experiment.*

1. *A challenger generates the public parameters* Λ *of the protocol (e.g. a generator describing a group etc.).*
2. *The adversary receives* Λ *as input, and may query the challenger. To this end, it submits a symbol* $\perp$*. Then, the challenger runs a protocol instance, and obtains the transcript* T *of all messages exchanged during the protocol and a key* K_0*. The challenger returns* (T, K_0)*.*

3. *Eventually, the adversary outputs a special symbol* $\top$*. Given* $\top$*, the challenger runs a protocol instance, obtaining the transcript* T *and key* K_0*, samples* K_1 *uniformly at random from the key space of the protocol, and tosses a fair coin* $b \in \{0,1\}$*. Then it returns* (T, K_b) *to the adversary.*
4. *The adversary may continue making* $\perp$*-queries to the challenger.*
5. *Finally, adversary* E *outputs a bit* b'*.*

We say that the adversary wins the game, if $b = b'$*.*

Simple protocols satisfying the above definition are the Diffie-Hellman protocol (under the DDH assumption), or key-transport using an IND-CPA secure encryption scheme (i.e., party A samples a random key k, encrypts k under B's public key, and sends the ciphertext to B).

2.5 Secure Authenticated Key Exchange

While the security model for passively-secure key-exchange protocols is very simple, a more complex model is required to model the capabilities of active adversaries to define secure authenticated key-exchange. We must describe the subtleties of executions that we expect from the implementations of the protocol, the attacks against which our protocol should be secure, and which outcome we expect if we run the protocol with the defined adversary. In accordance with the line of research [5, 11, 24, 16] initiated by Bellare and Rogaway [2], we model our adversary by providing an "execution environment", which emulates the real-world capabilities of an active adversary. That is, the adversary has full control over the communication network, thus may forward, alter, or drop any message sent by the participants, or insert new messages.

Execution Model. Let $I = I(\kappa)$ and $S = S(\kappa)$ be polynomials in the security parameter κ. Our model is characterized by a collection of oracles

$$\{\pi_{i,j}^{s} : i, j \in [I], s \in [S]\}$$

An oracle $\pi_{i,j}^{s}$ represents an entity i running the protocol with entity j for the s-th time. Each oracle maintains its own internal state (e.g. nonces), all oracles representing some entity i share the same long-term secrets of entity i. Moreover, each oracle $\pi_{i,j}^{s}$ maintains a variable T storing an ordered list of all messages sent and received by $\pi_{i,j}^{s}$ so far.

An oracle aborts, if it receives a message which is not valid according to the protocol specification, or terminates after it has sent or received the last protocol message according to the protocol specification. When a process terminates, it outputs "accept" or "reject" and (possibly) a key k.

An adversary may interact with these oracles by issuing different types of queries. Before the first query is asked, long-term secret/public key pairs (pk_i, sk_i) for each entity i are generated. An adversary $\mathcal{A}$ receives as input the long-term public keys $(pk_1, \ldots, pk_l)$ of all parties, and may then ask the following query:

– $\mathsf{Send}(\pi^s_{i,j}, m)$: The adversary can use this query to send any message m of his own choice to oracle $\pi^s_{i,j}$. The oracle will respond according to the protocol specification. If $m = \emptyset$, where $\emptyset$ denotes the empty string, then $\pi^s_{i,j}$ will respond with the first protocol message.

Secure Authentication Protocols. An authentication protocol is a protocol run between two processes $\pi^s_{i,j}$ and $\pi^t_{j,i}$ of two parties P_i and P_j, where both processes output either "accept" or "reject" at the end of the protocol. We define correctness and security of an authentication protocol following the idea of *matching conversations*, as introduced by Bellare and Rogaway [2].

In the following let $T_{i,s}$ denote the transcript of all messages sent and received by process $\pi^s_{i,j}$. Intuitively, we would like to say that a protocol is *correct*, if a process $\pi^s_{i,j}$ outputs "accept" *if* there exists a process $\pi^t_{j,i}$ with $T_{i,s} = T_{j,t}$. Likewise, we would like to say that a protocol is *secure*, if a process accepts *only if* there exists a process $\pi^t_{j,i}$ with $T_{i,s} = T_{j,t}$.

As in [2], we face a minor technical obstacle here, which is inherent to authentication protocols. Suppose that P_j sends the last message of the protocol (thus, P_i has initiated the protocol run if the number of protocol rounds is even, or P_j has initiated the protocol if the number of rounds is odd). Party P_j does not get any response to its last message, thus has to accept without knowing whether P_i received the last message.[2] To overcome this obstacle, we let $T'_{i,s}$ be the transcript $T_{i,s}$ truncated by the last message, and we have to define correctness and security in a slightly more complicated way.

Definition 5. *We say that two processes $\pi^s_{i,j}$ and $\pi^t_{j,i}$ have* matching conversations*, if either*

– *P_i sends the last message of the protocol according to the protocol specification and it holds that $T'_{j,t} = T'_{i,s}$, or*
– *P_j sends the last message of the protocol according to the protocol specification and it holds that $T_{j,t} = T_{i,s}$.*

Definition 6. *We say that an authentication protocol is* correct*, if for all processes $\pi^s_{i,j}$ holds that $\pi^s_{i,j}$ "accepts" if there exists a process $\pi^t_{j,i}$ such that $\pi^s_{i,j}$ and $\pi^t_{j,i}$ have matching conversations.*

Definition 7. *We say that an authentication protocol is* secure *in the Bellare-Rogaway model, if for all probabilistic polynomial-time (PPT) adversaries $\mathcal{A}$, interacting with the black-box $\mathcal{O}(\Pi)$ as described above in the execution model, holds that:*

Each process $\pi^s_{i,j}$ of $\mathcal{O}(\Pi)$ "accepts" only if *there exists a process $\pi^t_{j,i}$ such that $\pi^s_{i,j}$ and $\pi^t_{j,i}$ have matching conversations, except for some negligible probability $\epsilon = \epsilon(\kappa)$ in the security parameter.*

[2] In contrast, a protocol can be designed such that the party receiving the last message accepts only if it has received this message correctly according to the protocol specification.

Secure Authenticated Key-Exchange Protocols. An authenticated key-exchange protocol is an authentication protocol, where additionally both parties obtain a key k after accepting. Intuitively, we would like to say that a authenticated key-exchange protocol is secure, if

- the protocol is a secure authentication protocol, and
- an adversary can not distinguish a key k computed in a protocol run from a uniformly random value from the key space. This should hold even if the adversary is able to learn the key computed in other protocol instances.

We formalize this by extending the execution model by two more type of queries, which may be asked by the adversary.

- $\mathsf{Test}(\pi^s_{i,j})$: This query may only be asked once throughout the game. If process $\pi^s_{i,j}$ has not (yet) "accepted", the black-box returns some failure symbol $\perp$. Otherwise the black-box flips a fair coin b. If $b = 0$, a random element from the keyspace is returned. If $b = 1$ then the session key k computed in process $\pi^s_{i,j}$ is returned.
- $\mathsf{Reveal}(\pi^s_{i,j})$: The adversary may learn the encryption key K computed in process $\pi^s_{i,j}$ by asking this type of query. The adversary submits $\pi^s_{i,j}$ to the black-box. If process $\pi^s_{i,j}$ has "accepted", the black-box responds with the key k in process $\pi^s_{i,j}$. Otherwise some failure symbol $\perp$ is returned.

Definition 8. *Let $\mathcal{A}$ be a PPT adversary, interacting with the black-box $\mathcal{O}(\Pi)$ described in the above execution model (denoted with $\mathcal{A}^{\mathcal{O}(\Pi)}$).*

We say that an authenticated key-exchange protocol Π is secure *in the Bellare-Rogaway model, if 1.) Π is a secure authentication protocol according to Definition 7, and 2.)*

$$\left| \Pr[\mathcal{A}^{\mathcal{O}(\Pi)}(1^\kappa) = b] - \frac{1}{2} \right| \leq \epsilon$$

for all $\mathcal{A}$.

As Shoup pointed out in [27, §15], we do not have to explicitly model a $\mathsf{Corrupt}$-query, as one can efficiently reduce the standard BR-Model to a model without $\mathsf{Corrupt}$-queries (see also [6, p. 70 ff.]).

3 Authenticated Key Exchange Compiler in the Standard Model

Let us now describe our generic AKE compiler. The compiler takes as input the following building blocks (which have been defined in Section 2).

- A key-exchange protocol KE,
- a digital signature scheme $\Sigma = (\mathsf{SIG.Gen}, \mathsf{SIG.Sign}, \mathsf{SIG.Vfy})$,
- a message authentication code MAC,
- and a pseudorandom function PRF.

The compiled protocol between two parties A and B proceeds as follows (see also Figure 2).

1. A and B run the key exchange protocol. For instance, both parties may run the well-known Diffie-Hellman protocol [18]. Throughout this protocol run, both parties compute key k and record a transcript T^A_{KE} and T^B_{KE}, where T^C_{KE} consists of the list of all messages sent and received by party $C \in \{A, B\}$.
2. The key k computed by KE is used to derive two keys $K = \mathsf{PRF}(k, \text{“KE”})$ and $K_{\mathsf{mac}} = \mathsf{PRF}(k, \text{“MAC”})$, where “KE” and “MAC” are some arbitrary fixed constants such that “KE” $\neq$ “MAC”.[3]
3. Then A samples a random nonce $r_A \xleftarrow{\$} \{0,1\}^\lambda$ and sends it to B, B samples $r_B \xleftarrow{\$} \{0,1\}^\lambda$ and sends it to A.
4. Party A computes a signature $\sigma_A \xleftarrow{\$} \mathsf{SIG.Sign}(sk_A, T^A_1)$ under A's secret key sk_A, where $T^A_1 = (T^A_{\mathsf{KE}}||r_A||r^A_B)$ is the transcript of all messages sent and received by A so far. Then B computes a signature over the transcript $T^B_1 = (T^B_{\mathsf{KE}}||r^B_A||r_B)$ of all messages sent and received by B. Let $T^A_2 = (\sigma_A||\sigma^A_B)$ denote the signatures sent and received by A, and $T^B_2 = (\sigma^B_A||\sigma_B)$ be the signatures sent and received by B.
5. A sends a MAC $t_A = \mathsf{MAC}(K_{\mathsf{mac}}, T^A_2||0)$ over transcript T^A_2 using the key K_{mac} computed in 2. B replies with $t_B = \mathsf{MAC}(K_{\mathsf{mac}}, T^B_2||1)$.
6. Party A accepts, if $\mathsf{SIG.Vfy}(pk_B, T^A_1, \sigma^A_B) = 1$ and $t_B = \mathsf{MAC}(K_{\mathsf{mac}}, T^A_2||1)$, that is, if σ^A_B is a valid signature for T^A_2 under B's verification key pk_B and if w_B is a valid MAC under key K_{mac} for $T^A_2||1$. B accepts if it holds that $\mathsf{SIG.Vfy}(pk_A, T^B_1, \sigma^B_A) = 1$ and $w_A = \mathsf{MAC}(K_{\mathsf{mac}}, T^B_2||0)$. Finally, if both parties accept then the key K is returned.

Observe that the signatures and MACs are verified using the *internal* transcripts of party A and B. The intention behind the idea of embedding the transcripts in the protocol is to detect any changes that an active adversary makes to the messages sent by A and B. Informally, in the two-layer authentication consisting of the signature scheme and MAC, the signature is used to authenticate users and thwart *man-in-the-middle* attacks on the key-exchange protocol, while the MAC is used as an implicit “key confirmation” step to avoid *unknown key-share* attacks [14, 13].

This allows us to prove security requiring only pretty weak security properties from the utilized building blocks, namely we require that KE is secure against *passive* adversaries only, that the digital signatures are existential unforgeable under (non-adaptive) chosen-message attacks, and that the MAC and PRF meet their standard security notions.

Remark 1. The digital signatures sent in the first round after running KE are merely a concrete instantiation of a *tag-based authentication scheme* as introduced in [21]. It is possible to generalize the above protocol by replacing the

[3] Note that we assume here implicitly, that the output key space of KE matches the input key space of PRF. This fact is not only important for correctness, but also for the security proof.

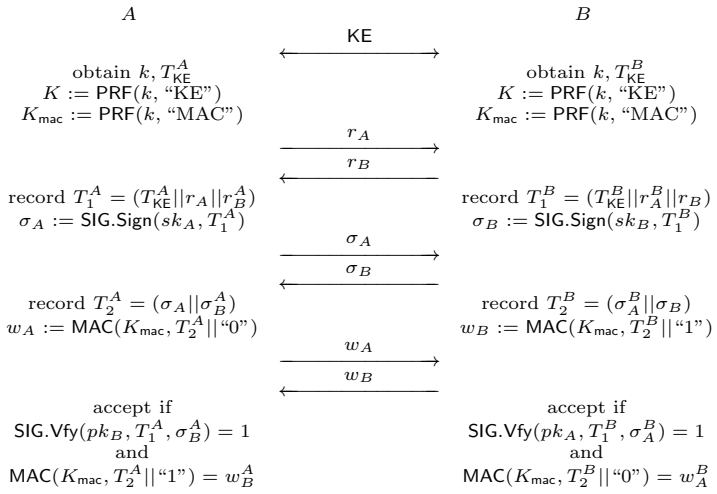

Fig. 2. AKE Protocol

digital signatures with a tag-based authentication scheme, without making substantial changes to the protocol or the security proof given below.

3.1 Security Analysis

Theorem 1. *If the* KE *protocol, the signature scheme, the message authentication code and the pseudo-random function are secure with respect to the definitions in Section 2, then the above protocol is a secure* AKE *protocol in the sense of Definition 8.*

We prove the above theorem by two lemmas. Lemma 1 states that the AKE protocol meets property 1) of Definition 8, Lemma 3 states that it meets property 2) of Definition 8.

Lemma 1. *If the key exchange protocol (*KE*), the signature scheme (*SIG*), the message authentication code (*MAC*) and the pseudo-random function (*PRF*) are secure with respect to the definitions in Section 2, then the above protocol holds property 1) of Definition 8.*

Proof. (Sketch) The proof proceeds in a *sequence of games*, following [3, 28]. We assume there exists an adversary $\mathcal{A}$ that breaks the security of the above protocol. We then describe several intermediate games that step-wisely modify the original game. Next we show that in the final security game the adversary has only negligible advantage in breaking the security of the protocol. Finally we prove that (under the stated security assumptions) no adversary can distinguish any of these games G_{i+1} from its predecessor G_i. Let Win_i be the event that $\mathcal{A}$ wins in Game i. In the following let $\mathsf{negl}(\kappa)$ be some (unspecified) negligible

function in the security parameter κ and let Win_i be the event, that $\mathcal{A}$ wins in game G_i (meaning he can answer the Test-query correctly).

In the following proof we introduce certain events, which we use to describe a difference between two games. We introduce a Difference Lemma as follows:

Lemma 2. *Let two Games G_i and G_{i+1} proceed identical until an event **abort** occurs.*

$$\Pr[Win_i \wedge \neg abort] = \Pr[Win_{i+1} \wedge \neg abort].$$

Then we have

$$\begin{aligned} |\Pr[Win_i] - \Pr[Win_{i+1}]| &= |\Pr[Win_i \wedge abort] + \Pr[Win_i \wedge \neg abort] \\ &\quad - \Pr[Win_{i+1} \wedge abort] - \Pr[Win_{i+1} \wedge \neg abort]| \\ &= |\Pr[Win_i \wedge abort] - \Pr[Win_{i+1} \wedge abort]| \\ &\leq Pr[abort] \end{aligned}$$

The simulator starts by internally choosing an oracle $\pi^s_{i,j}$, which will hold the challenge information in the following games. In the following for reasons of simplicity we call this oracle π_{ch}. The simulator aborts, when the adversary asks a Test-query to a different oracle than π_{ch}. (**Note**: The probability for choosing the correct oracle is $1/q^2$, q being the number of all parties of $\mathcal{S}$.)

Game 0. This is the original security game.

Game 1. In this game, the challenger proceeds exactly like the challenger in Game 0, except that we add an abort rule. The challenger raises event $Koll$ and aborts, if during the simulation any of the nonces (r_A or r_B) appears twice.

Since both games proceed identical until $Koll$ is raised, we can state by Lemma 2 that

$$|\Pr[Win_0] - \Pr[Win_1]| \leq \Pr[Koll].$$

All oracles sample r_A or r_B uniformly random from $\{0,1\}^{\lambda_1}$. Thus, by applying the birthday bound and the fact that the adversary has access to at most $d\ell$ oracles, we have

$$\Pr[Koll] \leq \frac{d^2\ell^2}{2^{\lambda_1}}.$$

The right-hand term is a negligible function in the security parameter, since d, ℓ, and λ_1 are polynomials.

Game 2. This game proceeds like the previous game, except that in this game we let the challenger raise event $NoMatch_1$ and abort, if A or B accept and $T_1^A \neq T_1^B$.

Claim 2. We claim by lemma 2 that

$$|\Pr[Win_1] - \Pr[Win_2]| \leq \Pr[NoMatch_1].$$

by the EUF-CMA security of the digital signature scheme.
Proof. We show by contradiction that $Pr[NoMatch_1] = \mathsf{negl}(\kappa)$ by designing a simulator $\mathcal{S}$ that uses an adversary $\mathcal{A}$ in a black-box manner to break the EUF-CMA security of the digital signature scheme for a given challenge public key pk^* when event $NoMatch_1$ occurs with non-negligible probability. We say, that games G_1 and G_2 are also *identical-until-bad*, meaning the games proceed *identical* until event $NoMatch_1$ occurs in game G_2. Thus, we have

$$|\Pr[Win_2] - \Pr[Win_1]| \leq \Pr[NoMatch_1].$$

The simulator proceeds as follows: It first chooses a party P_a belonging to π_{ch}. The simulator generates key pairs $(pk_i, sk_i)\ \forall\ P_i$ with $i \neq a$ with $\mathsf{SIG.Gen}(1^\kappa)$ and inputs the challenge public key pk^* to P_a. $\mathcal{A}$ now receives all public keys and may ask queries to any processes, receiving the appropriate response according to the protocol specification. If needed, the simulator may compute signatures for all parties $P_i \neq P_a$ using sk_i and may ask the signing oracle $\mathcal{O}_{SIG}$ for $P_i = P_a$. If queried, $\mathcal{O}_{SIG}$ outputs a signature σ for any input message m, such that σ is verifiable under the challenge public key pk^*.

It follows from the protocol specification that a party P_i accepts only if it holds that $\mathsf{SIG.Vfy}(pk_j, T_1^j, \sigma_1^j) = 1$. If $NoMatch_1$ is triggered at one time during the protocol run of a process $\pi^s_{a,j}$ (or $\pi^s_{j,a}$) $\mathcal{A}$ must have sent a signature σ^* of a message m^* (here: $m^* = T_1^a$), which he has not received from $\mathcal{O}_{SIG}$ before, so that $\mathsf{SIG.Vfy}(pk^*, m^*, \sigma^*) = 1$. If the simulator initially guessed the correct party P_a, it follows that (m^*, σ^*) is a valid forgery for the challenge public key pk^*, contradicting definition 1. It follows from game G_1, that an identical Transcript has not been sent before. This concludes the proof. □

Game 3. This game proceeds exactly like the previous game, except that the simulator now chooses a uniformly random key $\hat{k}$ to derive K_{mac} and K as $K_{\mathsf{mac}} = \mathsf{PRF}(\hat{k},\ \text{“MAC”})$ and $K = \mathsf{PRF}(\hat{k},\ \text{“KE”})$.

Claim 3. We claim that

$$|\Pr[Win_3] - \Pr[Win_2]| \leq \mathsf{negl}(\kappa)$$

by the passive security of the KE.

Recall that our modifications in Game 2 guarantee that $T_1^A = T_1^B$ if A or B accept, so we only have to consider a passive adversary in this game. *Proof.* We show that we can use an attacker $\mathcal{A}$ able to distinguish between this game and the previous game to break the passive security of our KE as follows:

Our simulator proceeds as in the previous game. $\mathcal{A}$ may ask queries to any processes, receiving the appropriate response according to the protocol specification (except for π_{ch}). If needed, the simulator computes the transcripts and the corresponding key. When $\mathcal{A}$ queries the challenge oracle π_{ch} it receives the challenge transcript T^{ch}_{KE} and the challenge key k_{ch} as output by the KE. The adversary may continue to ask queries and at some time point in time $\mathcal{A}$ asks

$\mathsf{Test}(\pi_{ch})$ and outputs a guess b'. Observe that for $b = 0$ ($K = \mathsf{PRF}(k,$ "KE")) we behave identical to Game 2 and for $b = 1$ ($K = \mathsf{PRF}(\hat{k},$ "KE")) we are in Game 3. Thus, if there is an attacker able to distinguish between games 2 and 3 with non-negligible probability $((|\Pr[Win_3] - Pr[Win_2]| > negl(\kappa))$, we can distinguish between k and $\hat{k}$. □

(Note, that an adversary able to distinguish between k and $\hat{k}$ would also be able to distinguish between $K_{\mathsf{mac}} = \mathsf{PRF}(\hat{k},$ "MAC") and $K_{\mathsf{mac}} = \mathsf{PRF}(k,$ "MAC"), hence breaking the security of the PRF.)

Game 4. This game proceeds exactly like the previous game, except that the simulator now chooses a uniformly random key $\tilde{k}$ (instead of K_{mac} output by the PRF) to compute w_A and w_B as $w_A = \mathsf{MAC}(\tilde{k}, T_2||0)$ and $w_B = \mathsf{MAC}(\tilde{k}, T_2||1)$.

Claim 4. We claim that

$$|\Pr[Win_4] - \Pr[Win_3]| \leq \mathsf{negl}(\kappa)$$

by the security of the pseudorandom function PRF. In the proof we exploit that we have exchanged the "real" key k computed in KE with a "random" key $\tilde{k}$ in Game 3.

Observe here that, since the output key space of KE must match the input key space of PRF, and PRF is assumed to be secure, it follows implicitly here that the output key space of KE needs to be super-polynomially large. *Proof.*

Similar to the previous game, the adversary $\mathcal{A}$ may ask queries to any processes, receiving the appropriate response according to the protocol specification. When $\mathcal{A}$ queries the challenge oracle π_{ch} it receives the challenge key k_{ch}. For all other processes, the simulator computes the output of PRF. At some point $\mathcal{A}$ asks $\mathsf{Test}(\pi_{ch})$ and then may continue to ask queries.

Observe that for $b = 0$ we behave identical to Game 3 ($w = \mathsf{MAC}(, T_2||x)$) and for $b = 1$ we are in Game 4 ($w = \mathsf{MAC}(\tilde{k}, T_2||x)$). Thus, if there is an attacker able to distinguish between games 3 and 4 with non-negligible probability $((|\Pr[Win_4] - Pr[Win_3]| > negl(\kappa))$, we can distinguish between K_{mac} and $\tilde{k}$. □

Game 5. This game proceeds exactly like the previous game, except that the simulator aborts if A or B accepts and $T_2^A \neq T_2^B$. We call that event $NoMatch_2$.

Claim 5. We claim by lemma 2 that

$$|\Pr[Win_4] - \Pr[Win_5]| \leq \Pr[NoMatch_2].$$

Recall that in Game 5 we must have $T_1^A = T_1^B$ due to our abort condition from Game 1, and that we have replaced the key k computed in KE with a uniformly random key $\tilde{k}$ in Game 3 to compute the MACs in the considered protocol instance. Thus, if we have $T_2^A \neq T_2^B$, then the adversary must have forged a MAC to make A or B accept. We can therefore use the adversary to break the security of MAC as follows:

Proof. We show by contradiction that $Pr[NoMatch_2] = \mathsf{negl}(\kappa)$ by designing a simulator $\mathcal{S}$ that uses an adversary $\mathcal{A}$ in a black-box manner to break the security of the message authentication code (MAC) when event $NoMatch_2$ occurs with non-negligible probability.

The simulator proceeds as follows: Similar to Game 2, it first guesses some party P_a, which is, participating in a process $\pi^s_{a,j}$ with some other party P_j, triggering the event $NoMatch_2$. The simulator also samples keys $k_i \xleftarrow{\$} \{0,1\}^\kappa \; \forall \; P_i$ with $i \neq a$ (and implicitly lets P_a use the challenge MAC-key). $\mathcal{A}$ now may ask queries to any processes, receiving the appropriate response according to the protocol specification. If needed, the simulator may compute MACs for all parties $P_i \neq P_a$ using k_i and may ask the MAC oracle $\mathcal{O}_{MAC}$ for $P_i = P_a$. If queried, $\mathcal{O}_{MAC}$ outputs a value $w = \mathsf{MAC}(k^*, m)$ for any input message m.

It follows from the protocol specification that a party P_i accepts only if it holds that $\mathsf{MAC}(k^*, T_2^A) = w^* = \mathsf{MAC}(k^*, T_2^B)$. If $NoMatch_2$ is triggered at one time during the protocol run of a process $\pi^s_{a,j}$ (or $\pi^s_{j,a}$) $\mathcal{A}$ must have sent a value w^* of a message m^* (here: $m^* = T_2^A$), which he has not received from $\mathcal{O}_{MAC}$ before, so that $\mathsf{MAC}(k^*, m^*) = \mathsf{MAC}(k^*, T_2^B)$. If the simulator initially guessed the correct party P_a, it follows that (m^*, w^*) is valid under the challenge key k^*, contradicting definition 2. It follows from Game 1, that an identical Transcript has not been sent before. This concludes the proof. □

Game 6. This game proceeds exactly like the previous game except that the simulator aborts if A or B accepts and $T_3^A \neq T_3^B$, where $T_3^A = (w_A, w_B^A)$ consists of the MACs sent and received by A and $T_3^B = (w_A^B, w_B)$ consists of the MACs sent and received by B.

Claim 6. We have

$$\Pr[Win_6] = \Pr[Win_5].$$

This follows from the fact that we have defined MAC as a deterministic function, and we have $T_1^A = T_1^B$ due to Game 2 and $T_2^A = T_2^B$ due to Game 5.

Collecting probabilities from Game 0 to 6, we obtain that both A and B accept *only if* they have *matching conversations*, except for some negligible error probability.

Lemma 3. *If* KE, SIG, MAC *and* PRF *are secure with respect to the definitions in Section 2, then the above protocol holds property 2) of Definition 8.*

Proof. (Sketch). Again we proceed in a sequence of games. The first 6 games of the proof are identical to the sequence in the proof of Lemma 1. We merely add one further game.

Game 7. This game proceeds exactly like the previous game except that the simulator now chooses K uniformly at random from the keyspace.

Claim 7. We claim that

$$|\Pr[Win_7] - \Pr[Win_6]| \leq \mathsf{negl}(\kappa)$$

This again follows from the security of the PRF, where we use that the seed $\hat{k}$ is chosen uniformly random and independent (cf. Game 2).

In Game 7, the adversary receives a uniformly random key K. However, by collecting probabilities from Game 0 to 7 we obtain that Game 7 is (computationally) indistinguishable from Game 0, which proves indistinguishability of "real" from "random" keys. Thus, the protocol is secure in the sense of Definition 8.

4 An Alternative AKE Compiler for Practical Protocols

Our second compiler is designed for practical applications, where we cannot change the session key K resulting from the KE protocol [15], or where we want to avoid an additional round of protocol messages after the authentication protocol. In this compiler, we directly integrate the transcript of the KE protocol, and the secret value K_{mac}, into the authentication protocol. To do so, we first have to define a "generic" scheme for an authentication protocol.

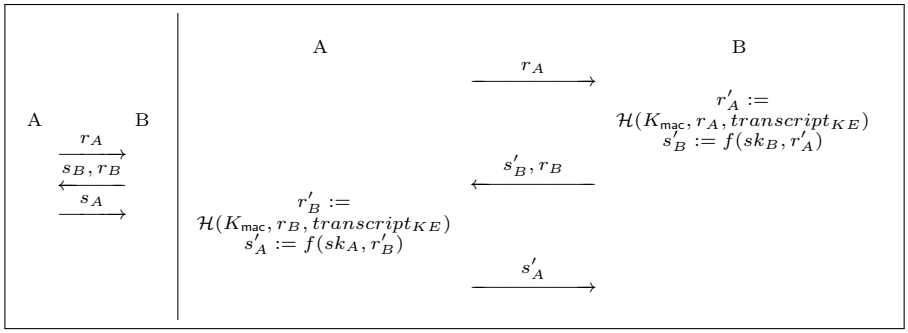

Fig. 3. Scheme of a standard mutual authentication protocol Γ (left), and the version Γ' modified by our compiler (right).

We only have minimal requirements on the authentication protocols. The party ("challenger") who wants to authenticate the other party ("prover") has to include a random value of high entropy into one of its protocol messages. (Otherwise an adversary may just query different instances of the prover for responses for the most probable challenges to increase her advantage.) The prover must answer with a value that was computed using his long-lived key sk, and the challenge itself.

The following protocols fulfill our requirements:

- AKEP1 and AKEP2 as defined in [2]
- Sigma- and Schnorr protocols (see [26])
- Zero-Knowledge Authentication protocols as introduced in [7]
- Zero-Knowledge Password-Proof protocols as introduced in [4].
- Signature based authentication protocols.

In this respect, our compiler may even enhance the security of the authentication protocol. This applies to the authentication of both parties, or of one party only.

Let Γ be an authentication protocol as depicted in Fig. 3. Then we denote by r_A a value (the challenge) that is sent from A to B, and by $s_B = f(sk_B, r_A)$ the value (response) returned to A that allows A to check the authenticity of B. The values r_B and s_A are defined analogously.

The main idea in the construction of a modified authentication protocol Γ' is to transmit r_A and r_B according to the protocol specification of Γ, but to compute the response based on both the received challenge, the transcript $transcript_\Pi$ of the key agreement protocol Π, and secret value K_{mac}. This is done using a random oracle $\mathcal{H}$. Our compiler $Comp$, which takes as input a key agreement protocol Π secure against passive adversaries, and a secure authentication protocol Γ, outputs an authenticated key agreement protocol $Comp(\Pi, \Gamma)$ which works as follows:

A&KE-2 Compiler: Let $(\pi^s_{A,B}, \gamma^s_{A,B})$ and $(\pi^t_{B,A}, \gamma^t_{B,A})$ be two pairs of oracles for Π and Γ.

1. Π is executed by $\pi^s_{A,B}$ and $\pi^t_{B,A}$ without any change. The resulting secret value is $k = (K, K_{\mathsf{mac}})$ for $\pi^s_{A,B}$, and $k' = (K', K'_{\mathsf{mac}})$ for $\pi^t_{B,A}$. (Ideally $k = k'$, but we have to take into account actions by the adversary.) The session key K (K', resp.) is used for encryption and integrity protection, and the secret value K_{mac} (K'_{mac}, resp.) is sent locally to the processes $\gamma^s_{A,B}$ and $\gamma^t_{B,A}$, together with the local transcript of the messages of Π. (The values K and K_{mac} are computed as described in Section 3.)
2. Now Γ is executed by $\gamma^s_{A,B}$ and $\gamma^t_{B,A}$ with the following change: In the computation of s_A and s_B, the values r_A and r_B are replaced with $r'_A := \mathcal{H}(K_{\mathsf{mac}}, r_B, transcript_\Pi)$ and $r'_B := \mathcal{H}(K'_{\mathsf{mac}}, r_A, transcript'_\Pi)$, and thus we get $s'_A = f(sk_A, r'_A)$ and $s'_B = f(sk_B, r'_B)$, where $\mathcal{H}(\cdot,\cdot,\cdot)$ is a random oracle. If $\gamma^s_{A,B}$ accepts, the local output is K, and K' for $\gamma^t_{B,A}$.

Lemma 4. *If Π is a key agreement protocol secure against passive adversaries, then it is impossible that three different oracles accept with the same (secret) state $(k, transcript_\Pi)$, where $k = (K, K_{\mathsf{mac}})$ is the secret value computed by Π, and $transcript_\Pi$ is the transcript of all protocol messages.*

Proof. If this was the case, then A, B and the (active) adversary E all would be able to compute k, but the adversary would not have modified any message exchanged between A and B (since the transcripts are identical). Thus E, acting as a passive adversary, would be able to compute k, a contradiction.

Lemma 5. *In $Comp(\Pi, \Gamma)$, any two oracles $\gamma^s_{A,B}$ and $\gamma^t_{B,A}$ with matching conversations have access to a unique random oracle that is defined as $\mathcal{H}_{A^tB^s}(\cdot) := \mathcal{H}(K_{\mathsf{mac}}, \cdot, transcript_\Pi)$. Neither E, nor any other oracle has access to this random oracle.*

Proof. Since the pair $(K_{\mathsf{mac}}, transcript_\Pi)$ is unique for any pair of oracles, $\mathcal{H}_{A^tB^s}(\cdot)$ is unique, too.

Theorem 2. *If Γ is a secure authentication protocol, then Γ' as defined in Fig. 3 also is a secure authentication protocol.*

Proof. Let ${\gamma'}^s_{A,B}$ and ${\gamma'}^t_{B,A}$ be two process (oracle) instances of A and B in Γ'. It should be clear that if ${\gamma'}^s_{A,B}$ and ${\gamma'}^t_{B,A}$ have matching conversations, then both oracles will accept.

We have to show that the probability that ${\gamma'}^s_{A,B}$ or ${\gamma'}^t_{B,A}$ accepts without a matching conversation is negligible. Now assume on the contrary that there is an adversary E' that is able to make ${\gamma'}^s_{A,B}$ or ${\gamma'}^t_{B,A}$ accept without a matching conversation, with non-negligible probability ϵ. Then we can define an adversary E that achieves the same goal with the protocol Γ: Since E' has no access to the random oracle $\mathcal{H}_{AB}$, she can only try to guess the challenge r'_A (r'_B, resp.). Now E is simply ignoring the challenge r_X she sees, and simply guesses a random challenge r''_X, and tries to compute s'_Y from this challenge. This strategy succeeds with non-negligible probability ϵ, and we have thus contradicted our assumption that Γ is a secure authentication protocol.

Theorem 3. *If Π is a key agreement protocol secure against passive adversaries, and if Γ is a secure authentication protocol, then $Comp(\Pi, \Gamma)$ is a secure authenticated key agreement protocol.*

Proof (Sketch). $\gamma^s_{A,B}$ and $\gamma^t_{B,A}$ will accept in Γ' if and only if they have access to the same random oracle $\mathcal{H}_{A^tB^s}(\cdot)$. (Otherwise they have to guess the challenge r'_X, which succeeds only with negligible probability.) If they have access to the same random oracle, then $\pi^s_{A,B}$ and $\pi^t_{B,A}$ completed Π with the same state $(k, transcript_\Pi)$. If $\gamma^s_{A,B}$ and $\gamma^t_{B,A}$ accept, Π and Γ were both completed by the same endpoints A and B. This excludes active attacks on Π (since the transcript is unchanged), and UKS attacks on Γ. Thus E may only mount a passive attack on Π, which succeeds only with negligible probability.

References

1. Mihir Bellare, Ran Canetti, and Hugo Krawczyk. A modular approach to the design and analysis of authentication and key exchange protocols (extended abstract). In *STOC*, pages 419–428, 1998.
2. Mihir Bellare and Phillip Rogaway. Entity authentication and key distribution. In *CRYPTO*, pages 232–249. Springer, 1993.
3. Mihir Bellare and Phillip Rogaway. The security of triple encryption and a framework for code-based game-playing proofs. In *EUROCRYPT*, pages 409–426. Springer, 2006.
4. Steven M. Bellovin and Michael Merritt. Encrypted key exchange: Password-based protocols secure against dictionary attacks. In *IEEE SYMPOSIUM ON RESEARCH IN SECURITY AND PRIVACY*, pages 72–84, 1992.
5. Simon Blake-Wilson and Alfred Menezes. Unknown key-share attacks on the station-to-station (STS) protocol. In *Public Key Cryptography*, pages 154–170, 1999.
6. Colin Boyd and Anish Mathuria. *Protocols for Authentication and Key Establishment*. Springer, 1 edition, September 2003.

7. J. Brandt, I. B. Damgård, P. Landrock, and T. Pedersen. Zero-knowledge authentication scheme with secret key exchange. In *CRYPTO*, pages 583–588. Springer, 1990.
8. Emmanuel Bresson, Olivier Chevassut, and David Pointcheval. Dynamic group Diffie-Hellman key exchange under standard assumptions. In *EUROCRYPT*, pages 321–336. Springer, 2002.
9. Emmanuel Bresson, Olivier Chevassut, David Pointcheval, and Jean-Jacques Quisquater. Provably authenticated group Diffie-Hellman key exchange. In *ACM Conference on Computer and Communications Security*, pages 255–264, 2001.
10. Emmanuel Bresson and Mark Manulis. Securing group key exchange against strong corruptions. In Masayuki Abe and Virgil D. Gligor, editors, *ASIACCS*, pages 249–260. ACM, 2008.
11. Ran Canetti and Hugo Krawczyk. Analysis of key-exchange protocols and their use for building secure channels. In *EUROCRYPT*, pages 453–474. Springer, 2001.
12. Ran Canetti and Hugo Krawczyk. Security analysis of IKE's signature-based key-exchange protocol. In *CRYPTO*, pages 143–161. Springer, 2002.
13. Kim-Kwang Raymond Choo, Colin Boyd, and Yvonne Hitchcock. Errors in computational complexity proofs for protocols. In *ASIACRYPT*, pages 624–643. Springer, 2005.
14. Kim-Kwang Raymond Choo, Colin Boyd, and Yvonne Hitchcock. Examining indistinguishability-based proof models for key establishment protocols. In *ASIACRYPT*, pages 585–604. Springer, 2005.
15. Kim-Kwang Raymond Choo, Colin Boyd, and Yvonne Hitchcock. On session key construction in provably-secure key establishment protocols. In *Mycrypt*, pages 116–131. Springer, 2005.
16. Cas J. F. Cremers. Session-state reveal is stronger than ephemeral key reveal: Attacking the NAXOS authenticated key exchange protocol. In *ACNS*, pages 20–33, 2009.
17. Rachna Dhamija, J. D. Tygar, and Marti A. Hearst. Why phishing works. In Rebecca E. Grinter, Tom Rodden, Paul M. Aoki, Edward Cutrell, Robin Jeffries, and Gary M. Olson, editors, *CHI*, pages 581–590. ACM, 2006.
18. Whitfield Diffie and Martin Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22:644–654, 1976.
19. Sebastian Gajek, Tibor Jager, Mark Manulis, and Jörg Schwenk. A browser-based kerberos authentication scheme. In *ESORICS*, pages 115–129. Springer, 2008.
20. Sebastian Gajek, Mark Manulis, Olivier Pereira, Ahmad-Reza Sadeghi, and Jörg Schwenk. Universally composable security analysis of TLS. In *ProvSec*, pages 313–327. Springer, 2008.
21. Tibor Jager, Florian Kohlar, Sven Schäge, and Jörg Schwenk. Generic compilers for authenticated key exchange. Full version. To appear on eprint `http://eprint.iacr.org/`, 2010.
22. Jonathan Katz and Moti Yung. Scalable protocols for authenticated group key exchange. In *CRYPTO*, pages 110–125. Springer, 2003.
23. Hugo Krawczyk. HMQV: A high-performance secure Diffie-Hellman protocol. In *CRYPTO*, pages 546–566. Springer, 2005.
24. Brian A. LaMacchia, Kristin Lauter, and Anton Mityagin. Stronger security of authenticated key exchange. In *ProvSec*, pages 1–16. Springer, 2007.
25. Paul Morrissey, Nigel P. Smart, and Bogdan Warinschi. A modular security analysis of the TLS handshake protocol. In *ASIACRYPT*, pages 55–73. Springer, 2008.
26. Claus P. Schnorr. Efficient identification and signatures for smart cards. In *CRYPTO*, pages 239–252. Springer, 1989.

27. V. Shoup. On formal models for secure key exchange. *IBM Research Report RZ*, 3120, 1999.
28. Victor Shoup. Sequences of games: a tool for taming complexity in security proofs. Cryptology ePrint Archive, Report 2004/332, Nov 2004.