New Modular Compilers for Authenticated Key Exchange

Yong Li[1⋆], Sven Schäge[2⋆⋆], Zheng Yang[1], Christoph Bader[1], and Jörg Schwenk[1]

[1] Horst Görtz Institute for IT Security, Ruhr-University Bochum, Germany
{yong.li,christoph.bader,joerg.schwenk}@rub.de
[2] University College London, United Kingdom
s.schage@ucl.ac.uk
[3] Chongqing University of Technology
zheng.yang@rub.de

Abstract. We present two new compilers that generically turn passively secure key exchange protocols (KE) into authenticated key exchange protocols (AKE) where security also holds in the presence of active adversaries. Security is shown in a very strong security model where the adversary is also allowed to i) reveal state information of the protocol participants and ii) launch theoretically and practically important PKI-related attacks that model important classes of unknown-key share attacks. Although the security model is much stronger, our compilers are more efficient than previous results with respect to many important metrics like the additional number of protocol messages and moves, the additional computational resources required by the compiler or the number of additional primitives applied. Moreover, we advertise a mechanism for implicit key confirmation. From a practical point of view, the solution is simple and efficient enough for authenticated key exchange. In contrast to previous results, another interesting aspect that we do not require that key computed by the key exchange protocol is handed over to the compiler what helps to avoid additional and costly modifications of existing KE-based systems.

Keywords: Key Exchange, Authenticated Key Exchange, Security Model

1 Introduction

Authenticated key exchange (AKE) protocols are among the most important building blocks of secure network protocols. They allow a party A to i) authenticate a communication partner B and ii) securely establish a common session key with B. In many existing systems both of these tasks are addressed by a single protocol. This can yield very efficient solutions. However, there are several

⋆ supported by secure eMobility grant number 01ME12025.
⋆⋆ supported by EPSRC grant number EP/G013829/1.

scenarios where these two tasks are actually addressed by separate protocols. For example in typical browser-based applications, the user relies on TLS to exchange a session key k with an authenticated server. The user, on the other hand, often uses a simple username/password combination which is encrypted with k to authenticate himself. In this paper, we consider generic and very efficient constructions that securely combine authentication protocols (AP) and passively secure key exchange protocols (KE) to yield authenticated key exchange.

While combined solutions may be more efficient in general, there are several advantages for the modular design of AKE systems. One is flexibility as one can resort to a rich collection of existing authentication and key exchange protocols that can be combined to yield new AKE systems which are specifically crafted to fit a certain application scenario. The second reason is applicability, as a generic compiler (ideally) does not require any modifications in existing implementations of the input protocols (which are often costly or error-prone in practice). Instead, security can be established by simply 'adding' the implementation of the compiler to the system. Third, a generic compiler can considerably simplify the security analysis, as only the input protocols have to be analysed to meet their respective security requirements. Security of the entire AKE protocol follows from the security proof of the compiler. This greatly pays off in the setting of key exchange protocols, as here, we usually only require the underlying key exchange protocol to be passively secure (which is a comparably simple security notion) while the output protocol must be secure even under active attacks (where the adversary is granted several additional attack capabilities). Finally, generic compilers may help to lower the assumptions on practical security protocols which have not been designed with provable security in mind and where a proof of AKE is not known. If such protocols are used in higher-level systems it is typically simply assumed that they constitute secure AKE systems. If we apply a generic compiler to such a protocol we can relax the security assumption to only requiring that the protocol is secure in the presence of passive adversaries.

1.1 Contribution

We present two very efficient compilers that construct secure AKE systems from authentication protocols (AP) and passively secure key exchange protocols (KE). To the best of our knowledge, they are the first such compilers that are efficient and truly generic, i.e. they do not require any modifications in the underlying AP and KE protocols. Thus, they are easily applicable to existing systems, what makes them very useful in practice. For example, imagine a wide-spread network software which internally consists of an initial key exchange protocol call that computes a session key k. Next, the software directly applies a symmetric primitive which uses k as the corresponding key. By design, the application does not provide any mechanism to output the security-critical session key to any other software. An ideal compiler should treat the entire network application as black-box except for the fact that the key exchange protocol is an interactive protocol and any party in the network can obtain the transcript of the protocol execution. Previous compilers require costly modifications on the key exchange

protocol such that either the messages have to be modified or the secret session key k also has to be output to the compiler. A new session key is computed using e.g. the requested key derivation function (KDF), i.e. the compilers require the session key of the underlying key exchange KE protocol as input. We stress that in some scenarios it is very difficult or impossible (for example because the network application is closed-source) to realize these modifications. Our compilers, in contrast, avoid such problems as they only require the public transcript of the key exchange protocol but not the secret session key from the passively secure KE protocol as input. Our compilers are very efficient but restrict the class of KE protocols to those which do not rely on long-term keys. We have chosen to restrict our attention to this class of key exchange protocols because they i) allow for efficient protocols with very high security guarantees (like forward secrecy) and ii) they can efficiently be recognized. Let us elaborate on this. As a consequence of our restriction long-term keys are only used in the authentication protocol, whereas in the KE protocol, all values are freshly drawn in each new communication session. We stress that important key exchange mechanisms like ephemeral Diffie-Hellman key exchange fall into this class.[1] Our restriction is useful to design protocols with forward secrecy, which states that even after the compromise of long-term keys previously executed sessions remain secure. The same restriction (that requires the KE protocol without long-term key) [2] is made on the KE protocols which are used in the recent compiler by Jager, Kohlar, Schäge, and Schwenk (JKSS) [8]. The well-known compiler by Katz and Yung (KY) uses a slightly different approach by directly requiring that the input protocol provides forward secrecy [11].

We present two compilers each of which relies on a different authentication mechanism. Our first compiler is very efficient. It relies on signature schemes and only requires two additional moves in which signatures are exchanged. The second compiler relies on public key encryption systems, one-time message authentication schemes, and collision resistant hash functions. Although the first compiler is more efficient, the second compiler accounts for scenarios where the parties do not have (certified) signature keys but only encryption keys. This can often occur in practice. For example, the most efficient (for the client) and most wide-spread key exchange mechanism in TLS is RSA key transport. Here the server certificate only contains an RSA encryption key. The latter can be extended to symmetric-based authentication systems in which the communication parties have secure pre-shared keys.

In this paper, we focus on practicality and efficiency of our solutions. We stress that we could generalize our compiler in a straight-forward way to also

[1] Also, we can still use long-term key based key exchange mechanisms like encrypted key transport with a slight loss of efficiency: we just require that the long-term keys are drawn freshly in each session and then are exchanged with the communication partner as a first move in the protocol execution. This at most adds two additional moves to the key exchange protocol.

[2] The restriction is implicit in [8]. An explicit statement can be found in the full version of the paper [9].

work with a general class of authentication protocols. The JKSS compiler has made a first step in that direction by proposing an abstract class of authentication schemes called tag-based authentication scheme (TBAS) that can be used in the JKSS compiler. However, the encryption-based authentication mechanism of our second compiler is not covered by this abstract description. This is mainly because the TBAS definition of the JKSS does not account for verifiers with *secret* state information. In terms of efficiency, the recent JKSS compiler is less efficient than any of our compilers as it additionally requires two random nonces, two MAC values to be exchanged (it's too costly for data transmission) and an additional computation of the new session key for authenticated key exchange. All our solutions work in the standard model, i.e. without assuming random oracles.

Technical Contribution. Our efficiency improvements rely on the following techniques. First, we do not use explicit key confirmation to thwart unknown-key share attacks. Instead we use a form of implicit key confirmation where we include the identities of the partners in the messages that are authenticated. At the same time, this helps to also counter strong attacks that an adversary might launch with the help of the extended attack capabilities (state reveals and PKI-based attacks) of our strong security model. In terms of efficiency, this helps us to save the exchange of two MAC values (as compared to the JKSS compiler). As our second efficiency improvement, we formally show that for security we do not have to exchange uniformly random nonces after the key exchange protocol as in the JKSS compiler. In the JKSS compiler these nonces are solely used to make every session's transcript unique. We can prove that instead it is sufficient to use the public ephemeral keys which are exchanged in the key exchange protocol. Technically, we show that if a key exchange protocol that does not rely on long-term keys is passively secure, then with negligible probability there are no collisions among the ephemeral public keys. This is sufficient to show that even in the presence of active attackers each transcript is unique as long as one party is uncorrupted. Finally, our efficient compilers only require the public transcript of the key exchange protocol, denoted here as KE, but not the secret key k_{KE} from KE as input. Our approach helps us to save the additional computation of a new session key for authenticated key exchange (as compared to previous compiler). In other words, our compilers require no cryptographic session key generator other than KE itself.

1.2 The Security Model

Our proofs of our compilers hold in two very strong security models respectively. These models rely on the concept of indistinguishability of session keys which first emerged in the seminal work of Bellare and Rogaway [2] and later extended by [6,18] to the public key setting. However, our model substantially extends these models. In contrast to previous works, we explicitly model the revelation of state information of sessions (via a $\mathsf{RevealState}$ query) and strong

and practical PKI-based attacks (via a RegCorruptParty query) like the public key substitution attack (PKS) [4,16] or the duplicate-signature key selection (DSKS) attack [16,12]. We believe that the revelation of state information is much more realistic than (just) the revelation of keys. To model strong and practical PKI-related attacks we use the RegCorruptParty query into our models that allows attackers to register adversarially chosen public keys and identities. Observe that the adversary does not have to know the corresponding secret key. In practice, most certification authorities (CAs) do not require the registrant to deliver proofs of knowledge of the secret key. Using RegCorruptParty query the adversary may easily register a public key which has already been registered by another honest user U. Since the public keys are equal, all the signatures that are produced by U can be re-used by the adversary. Such attacks can have serious security effects [4,16,12]. Our model also formalizes perfect forward secrecy. Forward secrecy is a very strong form of security which guarantees that past sessions remain secure even if the long-term keys get exposed in later sessions. We use a formal definition of forward secrecy that is adopted from [10].

1.3 Related Work

In 1998, Bellare, Canetti and Krawczyk (BCK) were the first to consider a modular way for the development of AKE [1]. They propose to first design a protocol in the authenticated link model, an idealized model where the links between parties are always authenticated. Then they systematically transform the protocol into a protocol which is also secure in the unauthenticated link model, in which the adversary has control over all the message flows in the network, by applying a so-called authenticator. Basically, for every message A needs to transmit to B there will be some additional communication with B in which B sends a random nonce to A and A responds with an application of an authentication mechanism on this nonce (in a challenge-response like fashion). For example, when instantiated with a signature scheme or with a combination of an encryption system and a message authentication code, the authenticator adds another two messages to every message sent in the original protocol. Altogether, this amounts for a 200% increase in the number of moves of the protocol and the number of messages sent. Also, because of the (asymmetric) authentication mechanism every party needs considerable additional resources to *compute and verify* authenticators.

In 2003, Katz and Yung presented a generic compiler for group key agreement [11]. The KY compiler first adds an initial round to a passively secure group key exchange protocol where each party chooses a random nonce and broadcasts it to its communication partner. In the next step, the compiler basically adds to every message of the original protocol a signature which is also computed over all the random values that have been computed in the first phase. Their idea can be understood as a generalization of the the BCK compiler to the group key exchange setting where they take a passively secure group key agreement protocol and turn it into an actively secure one. When restricted to the two-party

case, this compiler is much more efficient in terms of protocol moves, in contrast to the BCK compiler, each message sent does not need to be authenticated interactively. The KY compiler only accounts for a single round that is added to the input protocol. However, the compiler still modifies each message sent in the protocol by basically adding a signature to that message. As before, this approach amounts for a huge decrease in efficiency due to the additional signature generation and verification operations each user has to execute. Additionally KY show that the famous group key exchange protocol by Burmester-Desmedt [5] fulfills their notion of passive security. (In the two party case, however, it basically reduces to the standard ephemeral Diffie-Hellman key exchange protocol). The KY compiler outputs protocols which guarantee forward secrecy. However, it does require that the input group key protocols already provide forward secrecy. This assumption is similar to our (and the JKSS) assumption on the KE protocol to not rely on long-term keys. Our restriction is, in some sense rougher than that of KY but it allows for a very simple verification by inspection. We stress that we could adapt the KY definition and yield a slightly more general result. We think, however, that in scenarios where a complex, practical protocol is given it might be hard to inspect if the KY compiler is applicable at all. This would make the compiler less useful when we apply it to relax the security assumptions on a particular key exchange mechanism (from AKE to KE security). Intuitively, our approach implies forward-secrecy because if all values which are used to generate the session keys are freshly computed in each session of the passively secure key exchange protocol then the keys computed in the different sessions are independent. This intuition is formalized in the security proofs of the subsequent sections. In 2010 Jager et al. presented the first compiler which accounts only for a constant number of additional messages (which is independent of the KE protocol) to be exchanged [8], denoted here as JKSS compiler. In terms of efficiency, this compiler is closest to our results. Basically, the compiler, after executing the KE protocol, makes A and B additionally exchange 1) random nonces, 2) signatures over these nonces and the KE transcript and 3) two MAC values (using a MAC-key K_{mac} generated using

the session key from the passively secure KE protocol) which have been computed over all the previous messages. As mentioned above this compiler is less efficient than our solution. At the same time all of the above compilers do neither consider state reveals nor PKI-related attacks in their security analysis.

2 Security Assumptions

3 Preliminaries and Definitions

Let $[n] = \{1, \ldots, n\} \subset \mathbb{N}$ be the set of integers between 1 and n. We write $a \xleftarrow{\$} S$ to denote the action of sampling a uniformly random element a from a set S. Let '||' denote the operation concatenating two binary strings. Let $\mathcal{IDS}$ denote an identity space associated with security parameter κ.

3.1 Key Exchange Protocols

A two party key-exchange (KE) protocol is a protocol that enables those two parties to compute a shared secret key. In the following, we formally provide a very technical definition of KE which is more detailed than in most other works. This is solely for the purpose of deriving a technical result on general KE protocols without long-term keys. In other words, we require that every secret keys used to generate the session keys must be chosen freshly in each session. For simplicity we first focus on the practically most important class of two-move key exchange protocols. We stress that our definitions and results can easily be generalized to y-move key exchange protocols as sketched below.

A key exchange scheme $\mathsf{KE} = (\mathsf{KE.Setup}, \mathsf{KE.EKGen}, \mathsf{KE.SKGen})$ consists of three algorithms which may be called by a party $\mathsf{ID} \in \mathcal{IDS}$ in each session. Let $\mathcal{M}_{\mathsf{KE}}$ be the message space and $\mathcal{ESK}$ be the space for ephemeral secret key and $\mathcal{EPK}$ be the space for ephemeral public key. Let T be the transcript of all messages exchanged in a KE protocol instance (see Figure 1).

- $pms^{ke} \leftarrow \mathsf{KE.Setup}(1^{\kappa})$: This probabilistic polynomial time algorithm takes as input the security parameter κ and outputs a set of system parameters pms^{ke}. The parameters pms^{ke} might be implicitly used by other algorithms for simplicity.
- $(esk_{\mathsf{ID}}, epk_{\mathsf{ID}}, m_{\mathsf{ID}}) \xleftarrow{\$} \mathsf{KE.EKGen}(pms^{ke}, \mathsf{in})$: The probabilistic polynomial time algorithm takes as input the system parameters pms^{ke} and message $\mathsf{in} \in \mathcal{M}_{\mathsf{KE}}$ and outputs an ephemeral key pair $(esk_{\mathsf{ID}}, epk_{\mathsf{ID}})$, where $esk_{\mathsf{ID}} \in \mathcal{ESK}$ and $epk_{\mathsf{ID}} \in \mathcal{EPK}$, and a message $m_{\mathsf{ID}} \in \mathcal{M}_{\mathsf{KE}}$ that requires to be sent in a protocol move. The execution of this algorithm might be determined by the input message (in) which could be any information including for example identities of session participants, ephemeral public key or just empty string $\emptyset$. If $m_{\mathsf{ID}} = \emptyset$, for simplicity we may write $(esk_{\mathsf{ID}}, epk_{\mathsf{ID}}) \xleftarrow{\$} \mathsf{KE.EKGen}(pms^{ke}, \mathsf{in})$.
- $k \leftarrow \mathsf{KE.SKGen}(esk_{\mathsf{ID}}, T)$: The session key generator is a deterministic polynomial time algorithm which takes as input esk_{ID} of a session participant ID and transcript T of all messages exchanged in this session, and outputs a session key k.

Correctness. We say a correct key exchange protocol without long-term key if for any protocol instance with session key generated as $k := \mathsf{KE.SKGen}(esk_{\mathsf{ID}}, T)$ it holds that esk_{ID} is generated freshly by $\mathsf{KE.EKGen}$ in corresponding protocol instance. That is, each party computes each session key using only ephemeral secret key which is freshly generated by $\mathsf{KE.EKGen}$ in corresponding protocol instance. We consider key exchange protocols with perfect correctness that is

$$Pr\left[\begin{array}{c} \mathsf{KE.SKGen}(esk_{\mathsf{ID}_1}, T) = \mathsf{KE.SKGen}(esk_{\mathsf{ID}_2}, T); \\ (esk_{\mathsf{ID}_1}, epk_{\mathsf{ID}_1}, m_{\mathsf{ID}_1}) \xleftarrow{\$} \mathsf{KE.EKGen}(pms^{ke}, \mathsf{in}_1), \\ (esk_{\mathsf{ID}_2}, epk_{\mathsf{ID}_2}, m_{\mathsf{ID}_2}) \xleftarrow{\$} \mathsf{KE.EKGen}(pms^{ke}, \mathsf{in}_2), \\ (m_{\mathsf{ID}_1}, m_{\mathsf{ID}_2}) \in T. \end{array}\right] = 1.$$

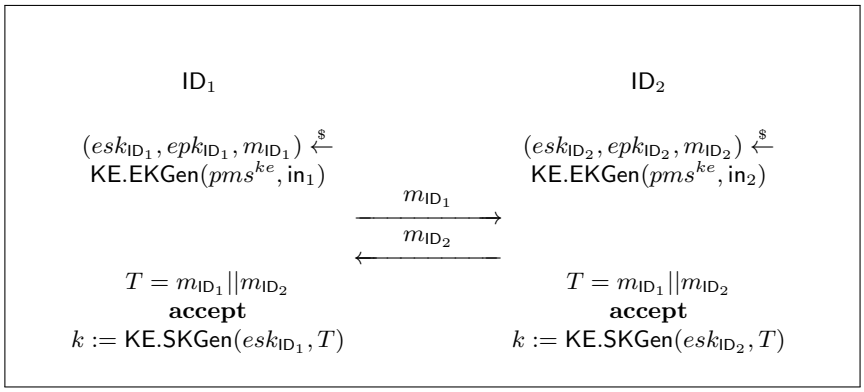

Fig. 1. General Two-move KE Protocol

We observe that in a passively secure key exchange protocol where we do not rely on long-term keys it is necessary that the values epk_{ID_1} and epk_{ID_2} are non-empty and 'meaningful'. This is because both parties have to keep the session key secret from a curious adversary. For example in ephemeral Diffie-Hellman key exchange (EDH) [7], the $\mathsf{KE.EKGen}$ is executed without any additional message, i.e. $\mathsf{in}_1 = \mathsf{in}_2 = \emptyset$, and the generated messages such that $m_{\mathsf{ID}_1} = epk_{\mathsf{ID}_1}$ and $m_{\mathsf{ID}_2} = epk_{\mathsf{ID}_2}$. In some KE protocols, the KE algorithms of the initiator ID_1 can be very different from those of the responder ID_2 like for example in encrypted key transport with freshly chosen key material (FEKT), in which case we could instantiate those messages in Figure 1 as: $\mathsf{in}_1 = \emptyset$, $\mathsf{in}_2 = m_{\mathsf{ID}_1} = epk_{\mathsf{ID}_1}$. We stress that the key pairs $(esk_{\mathsf{ID}_1}, epk_{\mathsf{ID}_1})$ and $(esk_{\mathsf{ID}_2}, epk_{\mathsf{ID}_2})$ may have distinct forms depending on specific KE protocol, which are also determined by the forms of messages $(\mathsf{in}_1, \mathsf{in}_2)$ while running $\mathsf{KE.EKGen}$.

In case both parties 'contribute' values which are used to computed the session key, i.e. $k \neq esk_{\mathsf{ID}_2}$ and $k \neq esk_{\mathsf{ID}_1}$, this is very obvious as the contribution of ID_1 has to be transmitted to ID_2 and vice versa. However, if only one party $\mathsf{ID}_c \in \{\mathsf{ID}_1, \mathsf{ID}_2\}$ decides on the session key $esk_{\mathsf{ID}_c} = k$, then k has to securely be transferred to the other party (ID'_c) via some form of encryption of k. In order to guarantee that only the single party ID'_c can decrypt the session key, the encryptor has to encrypt the session key exclusively for ID'_c using an ephemeral public key of ID'_c. As we do not rely on long-term keys, ID'_c has to generate this key freshly and send it to ID_c as $epk_{\mathsf{ID}'_c}$ in the first move of the key exchange protocol, resulting in $\mathsf{ID}'_c = \mathsf{ID}_1$ and $\mathsf{ID}_c = \mathsf{ID}_2$.

In order to model passive attacks we define an $\mathsf{Execute}(\mathsf{ID}_1, \mathsf{ID}_2)$ query. The adversary can use the query to perform passive attacks in which the attacker initiates and eavesdrops on honest executions between parties ID_1 and ID_2. Note that each identity should be uniquely chosen from the identity space $\mathcal{IDS}$. By using this query the adversary can obtain the transcripts that were exchanged during the honest execution of the protocol. For each $\mathsf{Execute}(\mathsf{ID}_1, \mathsf{ID}_2)$ query, an instance of KE protocol is executed between ID_1 and ID_2. After simulation

this query returns the transcript T of all messages exchanged in corresponding protocol instance and a session key[3].

Definition 1. *We say that a correct key-exchange protocol* KE *is* $(t, \epsilon_{\mathsf{KE}})$ *passively secure if for all probabilistic polynomial-time (*PPT*) adversary* $\mathcal{A}$ *holds that* $|[\mathsf{EXP}^{ps}_{\mathsf{KE},\mathcal{A}}(\kappa) = 1] - 1/2| \leq \epsilon_{\mathsf{KE}}$ *for some negligible function* $\epsilon_{\mathsf{KE}}(\kappa)$ *in the security parameter* κ *in the following experiment* $\mathsf{EXP}^{ps}_{\mathsf{KE},\mathcal{A}}(1^\kappa)$*: On input security parameter* 1^κ*, the security experiment is proceeded as a game between a challenger* $\mathcal{C}$ *and an adversary* $\mathcal{A}$ *based on a key exchange protocol* KE*, where the following steps are performed:*

1. *$\mathcal{C}$ generates a set of identities $\{\mathsf{ID}_1, \ldots, \mathsf{ID}_\ell\}$ for potential protocol participants where $\ell \in \mathbb{N}$. $\mathcal{A}$ is given all identities as input and is allowed to interact with $\mathcal{C}$ via making $\mathsf{Execute}(\mathsf{ID}_i, \mathsf{ID}_j)$ query at most d times for each party where $d \in \mathbb{N}$ and $i, j \in [\ell]$. As response, $\mathcal{C}$ returns (T, K_0) to $\mathcal{A}$.*
2. *At some point, $\mathcal{A}$ outputs a special symbol $\top$ and a fresh protocol test instance and sends them to $\mathcal{C}$. Given $\top$ and test instance, $\mathcal{C}$ runs a new protocol instance and outputs the transcript T^* and the session key K_0^*. Then, $\mathcal{C}$ samples K_1^* uniformly at random from the key space of the protocol, and tosses a fair coin $b \in \{0, 1\}$. Then $\mathcal{C}$ returns (T^*, K_b^*) to $\mathcal{A}$. After that $\mathcal{A}$ may continually perform $\mathsf{Execute}(\mathsf{ID}_i, \mathsf{ID}_j)$ queries. Finally, $\mathcal{A}$ may terminate with returning a bit b' as output.*
3. *At the end of the experiment, 1 is returned if $b' = b$; Otherwise 0 is returned.*

In the following, we formally show that for every passively secure key exchange protocol after polynomially calls to $\mathsf{KE.EKGen}$ there cannot be any collisions among the ephemeral public keys generated by certain type of $\mathsf{KE.EKGen}$. This lemma will be useful in the security proofs of our compilers to show that a compiler does not have to exchange additional random values after the KE run to guarantee that the transcripts which are authenticated with the authentication mechanism are unique. We can therefore discard the random values which are used in the JKSS compiler. Please note that for a two-move and two-party (ID_1 and ID_2) KE-protocol there exist at most two types of $\mathsf{KE.EKGen}$ algorithms which may be determined by input messages in_1 and in_2. We here explicitly classify the algorithm $\mathsf{KE.EKGen}$ into two types denoted by $\mathsf{KE.EKGen}_{\mathsf{ID}_1}$ for party ID_1 and $\mathsf{KE.EKGen}_{\mathsf{ID}_2}$ for party ID_2. While considering the collisions among ephemeral keys, let $Coll$ denote the event that: after a polynomial number q times execution of $\mathsf{KE.EKGen}$ algorithm there exist at least two ephemeral public keys epk and epk' generated by the ephemeral key generator $\mathsf{KE.EKGen}$ are identical, where the number q is determined by time t. Let probability ϵ_{coll} denote the event $Coll$ occurred within time t. We say all ephemeral keys generated by $\mathsf{KE.EKGen}$ are (q, t, ϵ_{coll})-distinct if those ephemeral keys are generated by $\mathsf{KE.EKGen}$ after q times execution of $\mathsf{KE.EKGen}$ algorithm within time t and there exists no collision among those ephemeral keys except for probability ϵ_{coll}. For space reasons we only provide a sketch of the proof.

[3] The attacker is also allowed to get the ephemeral keys by calling $\mathsf{Execute}$-query.

Lemma 1. *Assume* KE *is a* $(t, \epsilon_{\mathsf{KE}})$*-passively secure protocol without long-term key as defined above. Then all ephemeral public keys generated by* $\mathsf{KE.EKGen}$ *in the runs of* KE *are* (q, t, ϵ_{coll})*-distinct such that* $\epsilon_{coll} \leq q \cdot \epsilon_{\mathsf{KE}}$.

Proof. We first consider the case that the ephemeral keys are generated by different types of ephemeral key generators, i.e. $\mathsf{KE.EKGen}_{\mathsf{ID}_1} \neq \mathsf{KE.EKGen}_{\mathsf{ID}_2}$. Obviously, in this case there is no collision between ephemeral keys epk_{ID_1} and epk_{ID_2}, because those keys are assumed to be generated from different key spaces, so we only need to evaluate the collision probability among ephemeral keys generated by the same type of ephemeral key generators, i.e. $\mathsf{KE.EKGen}_{\mathsf{ID}_1} = \mathsf{KE.EKGen}_{\mathsf{ID}_2}$. For this case, we assume that with non-negligible probability ϵ_{coll} there will be a collision among the epk_{ID_1} after q protocol runs, or a collision among the epk_{ID_2} after q protocol runs. According to the protocol specification the epk_{ID_1} values are computed by randomized runs of $\mathsf{KE.EKGen}_{\mathsf{ID}_1}$ while the epk_{ID_2} values have been computed by randomized runs of $\mathsf{KE.EKGen}_{\mathsf{ID}_2}$. In particular, the computation of the epk_{ID_1} and epk_{ID_2} are deterministic in system parameters pms^{ke}, message in_1 (resp. in_2) and the internal random coins ω_{ID_1} used by ID_1 and ω_{ID_2} used by ID_2. The ω_{ID_1} and ω_{ID_2} are drawn uniformly random and in particular independently.

Let $epk^*_{\mathsf{ID}_1}$ and $epk^*_{\mathsf{ID}_2}$ be the ephemeral public keys that are exchanged in the test session and given, together with the challenge key k^*_b and transcript T^*, to the adversary. Let $esk^*_{\mathsf{ID}_1}$ and $esk^*_{\mathsf{ID}_2}$ be the corresponding ephemeral secret keys. These keys have also been computed using $\mathsf{KE.EKGen}_1$ (resp. $\mathsf{KE.EKGen}_2$) with random coin ω_{ID_1} (resp. ω_{ID_2}) and in_1 (resp. in_2). The adversary first guesses whether the collision occurs among the epk_{ID_1} or the epk_{ID_2} with probability $\geq 1/2$. In the first case, the adversary can re-run $\mathsf{KE.EKGen}_{\mathsf{ID}_1}$ $(q-1)$ times with $\omega_{\mathsf{ID}_1,i}$ and $\mathsf{in}_{1,i}$ to output $\{esk_{\mathsf{ID}_1,i}, epk_{\mathsf{ID}_1,i}\}$ for $i \in [1; q-1]$ in time less than t. With the same probability ϵ_{coll} it obtains two values $epk'_{\mathsf{ID}_1}, epk''_{\mathsf{ID}_1}$ among the q values $epk^*_{\mathsf{ID}_1}, epk_{\mathsf{ID}_1,1}, \ldots, epk_{\mathsf{ID}_1,(q-1)}$ with $epk'_{\mathsf{ID}_1} = epk''_{\mathsf{ID}_1}$. Since it holds with probability $\geq 2/q$ that either $epk'_{\mathsf{ID}_1} = epk^*_{\mathsf{ID}_1}$ or $epk''_{\mathsf{ID}_1} = epk^*_{\mathsf{ID}_1}$. In this case the adversary knows one pair $(\omega_{\mathsf{ID}_1,i}, \mathsf{in}_{1,i})$ that maps to $epk^*_{\mathsf{ID}_1}$. Let esk'_{ID_1} be the corresponding ephemeral secret key. We now have to show that esk'_{ID_1} helps us to break the passive security. This simply follows from the determinism of $\mathsf{KE.SKGen}$ and correctness of KE. Since we have perfect correctness the adversary $\mathcal{A}$ can compute the session key k by using the ephemeral secret key esk'_{ID_1} and transcript T^*. Next the adversary can compare whether $k^*_b = k$ and correctly guess the value b. In case there is a collision among the epk_{ID_2} the situation is similar. Hence, due to the security of KE protocol, we have that the probability bound $\frac{\epsilon_{coll}}{q} \leq \epsilon_{\mathsf{KE}}$.

3.2 General KE Protocols

The above definition of KE, the corresponding security definition, and the results of the above lemma can easily be extended to y-move KE protocols. Concretely, besides the $\mathsf{KE.Setup}$ and $\mathsf{KE.SKGen}$ algorithms, each party may run at most

$\lceil y/2 \rceil$ different *types* of $\mathsf{KE.EKGen}$ algorithms in each protocol instance depending on the input messages $\mathsf{in}_i : 1 \leq i \leq y$. Namely, each session participant can call at most $\lceil y/2 \rceil$ of times of $\mathsf{KE.EKGen}$ algorithms during protocol execution. We let each invocation of algorithm $\mathsf{KE.EKGen}$ in i-move ($1 \leq i \leq y$) as $\mathsf{KE.EKGen}_i$ which is used to compute the message for i-move. Consequently, we may have (for instance when y is even) a series of executions:$\{(esk_{\mathsf{ID}_1,1}, epk_{\mathsf{ID}_1,1}, m_{\mathsf{ID}_1,1}) \xleftarrow{\$} \mathsf{KE.EKGen}_1(pms^{ke}, \mathsf{in}_1), (esk_{\mathsf{ID}_2,2}, epk_{\mathsf{ID}_2,2}, m_{\mathsf{ID}_2,2}) \xleftarrow{\$} \mathsf{KE.EKGen}_2(pms^{ke}, \mathsf{in}_2), \ldots, (esk_{\mathsf{ID}_1,(y-1)}, epk_{\mathsf{ID}_1,(y-1)}, m_{\mathsf{ID}_1,(y-1)}) \xleftarrow{\$} \mathsf{KE.EKGen}_{(y-1)}(pms^{ke}, \mathsf{in}_{(y-1)}), (esk_{\mathsf{ID}_2,y}, epk_{\mathsf{ID}_2,y}, m_{\mathsf{ID}_2,y}) \xleftarrow{\$} \mathsf{KE.EKGen}_y(pms^{ke}, \mathsf{in}_y)\}$. We could therefore apply the result of Lemma 1 to y-move KE protocols, namely with overwhelming probability there is for instance no collision among all epk_{ID_b} generated by $\mathsf{KE.EKGen}_{\mathsf{ID}_b}$ with $b \in \{1, 2\}$.

3.3 Digital Signature Schemes

A digital signature scheme Σ is defined by three PPT algorithms ($\mathsf{SIG.Gen}$, $\mathsf{SIG.Sign}$, $\mathsf{SIG.Vfy}$) with associated public/private key spaces $\{\mathcal{PK}, \mathcal{SK}\}$, message space $\mathcal{M}_{\mathsf{SIG}}$ and signature space $\mathcal{S}_{\mathsf{SIG}}$ in the security parameter κ:

- $(sk, pk) \xleftarrow{\$} \mathsf{SIG.Gen}(1^\kappa)$: this algorithm takes as input κ and outputs $pk \in \mathcal{PK}$ and $sk \in \mathcal{SK}$;
- $\sigma \xleftarrow{\$} \mathsf{SIG.Sign}(sk, m)$: this algorithm generates a signature $\sigma \in \mathcal{S}_{\mathsf{SIG}}$ for $m \in \mathcal{M}_{\mathsf{SIG}}$ using signing key sk;
- $\{0, 1\} \leftarrow \mathsf{SIG.Vfy}(pk, m, \sigma)$: this algorithm outputs 1 if σ is a valid signature for m under key pk, and 0 otherwise.

Definition 2. *We say that Σ is $(q, t, \epsilon_{\mathsf{SIG}})$-secure against* existential forgery under adaptive chosen message attacks, *if* $\Pr[\mathsf{EXP}^{\mathsf{EUF-CMA}}_{\Sigma,\mathcal{A}}(\kappa) = 1] \leq \epsilon_{\mathsf{SIG}}$ *for every* PPT *adversary $\mathcal{A}$ running in time at most t in the following experiment:*

$\mathsf{EXP}^{\mathsf{EUF-CMA}}_{\Sigma,\mathcal{A}}(\kappa)$

$(sk, vk) \xleftarrow{\$} \mathsf{SIG.Gen}(1^\kappa)$;
$(\sigma^*, m^*) \leftarrow \mathcal{A}^{\mathcal{SIG}(sk,\cdot)}$, *which can make up to q queries to the signing oracle $\mathcal{SIG}(sk, \cdot)$ with arbitrary m;*
Output *1, if the following conditions are being held:*
1. $\mathsf{SIG.Vfy}(pk, m^*, \sigma^*) = 1$, *and*
2. m^* *is not among the previously submitted to $\mathcal{SIG}(sk, \cdot)$ oracle;*

Output *0, otherwise;*

where ϵ_{SIG} is a negligible function in κ, on input message m the oracle $\mathcal{SIG}(sk, \cdot)$ returns signature $\sigma \leftarrow \mathsf{SIG.Sign}(\mathsf{sk}, \mathsf{m})$ and the number of queries q is bound by time t.

Definition 3 (Uniqueness). *Let* SIG *= (*$\mathsf{SIG.Gen}$*,* $\mathsf{SIG.Sign}$*,* $\mathsf{SIG.Vfy}$*) be a secure signature scheme with the security requirements in the above security experiment and let pk be the output of* $\mathsf{SIG.Gen}$ *algorithm. Let (m, σ) be a valid signature pair by a signature function* $\mathsf{SIG.Sign}$*. We say that* SIG *is ϵ-unique if the probability that there exists a new signature $\sigma' \neq \sigma$ for which* $\mathsf{SIG.Vfy}(vk, m, \sigma')$ *= 1 is at most ϵ. For perfectly unique if $\epsilon = 0$.*

3.4 Collision-Resistant Hash Functions

Let $\mathsf{CRHF} : \mathcal{K}_{\mathsf{CRHF}} \times \mathcal{M}_{\mathsf{CRHF}} \to \mathcal{Y}_{\mathsf{CRHF}}$ be a family of keyed-hash functions, where $\mathcal{K}_{\mathsf{CRHF}}$ is the key space, $\mathcal{M}_{\mathsf{CRHF}}$ is the message space and $\mathcal{Y}_{\mathsf{CRHF}}$ is the hash value space. The public key $hk_{\mathsf{CRHF}} \in \mathcal{K}_{\mathsf{CRHF}}$ defines a hash function, denoted by $\mathsf{CRHF}(hk_{\mathsf{CRHF}}, \cdot)$, which is generated by a PPT algorithm $\mathsf{CRHF.KG}(1^\kappa)$ on input security parameter κ. On input a message $m \in \mathcal{M}_{\mathsf{CRHF}}$, $\mathsf{CRHF}(hk_{\mathsf{CRHF}}, m)$ generates a hash value $y \in \mathcal{Y}_{\mathsf{CRHF}}$.

Definition 4. CRHF *is called* $(t_{\mathsf{CRHF}}, \epsilon_{\mathsf{CRHF}})$*-secure if all* t_{CRHF}*-time adversaries* $\mathcal{A}$ *have negligible advantage* $\epsilon_{\mathsf{CRHF}} = \epsilon_{\mathsf{CRHF}}(\kappa)$ *with*

$$\Pr\left[\begin{array}{c} hk_{\mathsf{CRHF}} \leftarrow \mathsf{CRHF.KG}(1^\kappa), (m, m') \leftarrow \mathcal{A}(1^\kappa, hk_{\mathsf{CRHF}}); \\ m \neq m', (m, m') \in \mathcal{M}_{\mathsf{CRHF}}, \\ \mathsf{CRHF}(hk_{\mathsf{CRHF}}, m) = \mathsf{CRHF}(hk_{\mathsf{CRHF}}, m') \end{array}\right] \leq \epsilon_{\mathsf{CRHF}},$$

where the probability is over the random coins of the adversary and CRHF.KG. *If the hash key* hk_{CRHF} *is obvious from the context, we write* $\mathsf{CRHF}(m)$ *for* $\mathsf{CRHF}(hk_{\mathsf{CRHF}}, m)$.

A *collision-resistant hash function* CRHF is a deterministic algorithm which given a key $k_{\mathsf{CRHF}} \in \mathcal{K}_{\mathsf{CRHF}}$ and a bit string x outputs a hash value $y = \mathsf{CRHF}(k_{\mathsf{CRHF}}, x)$ in the hash space $\{0,1\}^\chi$ (with χ polynomial in κ). If k_{CRHF} is clear from the context we write $\mathsf{CRHF}(\cdot)$ short for $\mathsf{CRHF}(k_{\mathsf{CRHF}}, \cdot)$.

Definition 5. *We say that* CRHF *is a* (t, ϵ)-secure *collision-resistant hash function, if any* t*-time adversary* $\mathcal{A}$ *that is given* $k_{\mathsf{CRHF}} \xleftarrow{\$} \mathcal{K}_{\mathsf{CRHF}}$ *has an advantage of at most* ϵ *to compute two inputs* (x, x') *with* $x \neq x'$ *and* $\mathsf{CRHF}(x) = \mathsf{CRHF}(x')$.

3.5 Public Key Encryption Schemes

In our second compiler we can use a public key scheme (PKE) that satisfies security of indistinguishability under adaptive chosen-ciphertext attacks (IND-CCA2). A PKE scheme consists of three polynomial time algorithms (PKE.KGen, PKE.Enc, PKE.Dec) with the following semantics:

- $(pk, sk) \xleftarrow{\$} \mathsf{PKE.KGen}(1^\kappa)$: is a probabilistic polynomial-time key generation algorithm which on input the κ security parameter κ in unary outputs a pair of encryption/decryption keys (pk, sk);
- $C \xleftarrow{\$} \mathsf{PKE.Enc}(pk, m)$: is a probabilistic polynomial-time encryption algorithm which takes as inputs a public key pk and a message $m \in \mathcal{M}_{\mathsf{PKE}}$, outputs ciphertext $C \in \mathcal{C}_{\mathsf{PKE}}$, where $\mathcal{M}_{\mathsf{PKE}}$ is a message space and $\mathcal{C}_{\mathsf{PKE}}$ is a ciphertext space.
- $m \leftarrow \mathsf{PKE.Dec}(sk, C)$: is a deterministic polynomial-time decryption algorithm which takes as input a key sk and a ciphertext $C \in \mathcal{C}_{\mathsf{PKE}}$, and outputs either a message $m \in \mathcal{M}_{\mathsf{PKE}}$ or an error symbol $\perp$.

Definition 6. *We say that a public key encryption scheme* PKE *is* $(q, t, \epsilon_{\mathsf{PKE}})$*-secure against adaptive chosen-ciphertext attacks, if* $\Pr[\mathsf{EXP}^{ind-cca2}_{\mathsf{PKE},\mathcal{A}}(\kappa) = 1] \leq 1/2 + \epsilon_{\mathsf{PKE}}$ *holds for all* PPT *adversaries* $\mathcal{A}$ *that make a polynomial number of oracle queries* q *while running in time at most* t *in the following experiment:*

$\mathsf{EXP}^{ind-cca2}_{\mathsf{PKE},\mathcal{A}}(\kappa)$
$(pk, sk) \leftarrow \mathsf{PKE.KGen}(1^\kappa)$;
$(m_0, m_1, st) \leftarrow \mathcal{A}^{\mathcal{DEC}(sk,\cdot)}(pk)$, $(m_0, m_1) \in \mathcal{M}_{\mathsf{PKE}}$;
$b \xleftarrow{\$} \{0,1\}$, $(C^*) \leftarrow \mathsf{PKE.Enc}(pk, m_b)$;
$b' \leftarrow \mathcal{A}^{\mathcal{DEC}(sk,\cdot)}(C^*, st)$;
if $b = b'$ *then return 1, otherwise return 0;*

where ϵ_{PKE} *is a function in* κ*, and on input* C *the oracle* $\mathcal{DEC}(sk, C)$ *returns* $m \leftarrow \mathsf{PKE.Dec}(sk, C)$ *with the restriction that* $\mathcal{A}$ *is not allowed to query* $\mathcal{DEC}(sk, \cdot)$ *on the challenge ciphertext* C^**.*

3.6 Message Authentication Code Schemes

We first recall the notions of general message authentication codes scheme. A message authentication code MAC consists of three algorithms ($\mathsf{MAC.KGen}$, $\mathsf{MAC.Tag}$, $\mathsf{MAC.Vfy}$) with associated key space $\mathcal{K}_{\mathsf{MAC}}$, message space $\mathcal{M}_{\mathsf{MAC}}$ and tag space $\mathcal{T}_{\mathsf{MAC}}$.

- $\mathsf{K}_{\mathsf{MAC}} \xleftarrow{\$} \mathsf{MAC.KGen}(1^\kappa)$: The probabilistic key-generation algorithm takes as input a security parameter κ and outputs a secret key $\mathsf{K}_{\mathsf{MAC}} \in \mathcal{K}_{\mathsf{MAC}}$.
- $t_{tag} \leftarrow \mathsf{MAC.Tag}(\mathsf{K}_{\mathsf{MAC}}, m)$: The algorithm $\mathsf{MAC.Tag}$ takes as input a secret key $\mathsf{K}_{\mathsf{MAC}} \in \mathcal{K}_{\mathsf{MAC}}$ and a message $m \in \mathcal{M}_{\mathsf{MAC}}$ and outputs an authentication tag $t_{tag} \in \mathcal{T}_{\mathsf{MAC}}$.
- $b \in \{0,1\} \leftarrow \mathsf{MAC.Vfy}(\mathsf{K}_{\mathsf{MAC}}, m, t_{tag})$: The deterministic verification algorithm $\mathsf{MAC.Vfy}$ takes as input a secret key $\mathsf{K}_{\mathsf{MAC}} \in \mathcal{K}_{\mathsf{MAC}}$, a message $m \in \mathcal{M}_{\mathsf{MAC}}$ and a tag $t_{tag} \in \mathcal{T}_{\mathsf{MAC}}$ and outputs $b = 1$ if it accepts. Otherwise, it returns $b = 0$.

Definition 7. *We say that a message authentication code scheme* MAC *is* $(q, t, \epsilon_{\mathsf{MAC}})$*-secure against forgeries under adaptive chosen-message attacks, if probability bound* $\Pr[\mathsf{EXP}_{\mathsf{MAC},\mathcal{A}}(\kappa) = 1] \leq \epsilon_{\mathsf{MAC}}$ *holds for every* PPT *adversary* $\mathcal{A}$ *running in time at most* t *in the following experiment:*

$\mathsf{EXP}_{\mathsf{MAC},\mathcal{A}}(\kappa)$
$K \xleftarrow{\$} \mathsf{MAC.KGen}(1^\kappa)$;
$(m^*, t^*_{tag}) \leftarrow \mathcal{A}^{\mathcal{MAC}(K,\cdot)}$*, which can make up to* q *queries to oracle* $\mathcal{MAC}(K, \cdot)$*;*
Output *1, if it holds that*
1. $1 = \mathsf{MAC.Vfy}(\mathsf{K}_{\mathsf{MAC}}, m^*, t^*_{tag})$;
2. $\mathcal{A}$ *didn't submit* m^* *to* $\mathcal{MAC}(K, \cdot)$ *oracle;*
Output *0, otherwise;*

where $\epsilon_{\mathsf{MAC}} = \epsilon_{\mathsf{MAC}}(\kappa)$ *is a negligible function in the security parameter* κ*, on input message* m *the oracle* $\mathcal{MAC}(K, m)$ *returns tag* $t_{tag} \leftarrow \mathsf{MAC.Tag}(K, m)$ *and the number of queries* q *is bound by time* t*.*

Definition 8 (Uniqueness). *Let* MAC *=* ($\mathsf{MAC.KGen}$, $\mathsf{MAC.Tag}$, $\mathsf{MAC.Vfy}$) *be a secure* MAC *scheme with the security requirements in the above security experiment and let* $\mathsf{K_{MAC}}$ *be the output of* $\mathsf{MAC.KGen}$ *algorithm. Let* (m, t_{tag}) *be a valid tag pair by tagging algorithm* $\mathsf{MAC.Tag}$*. We say that* MAC *is* ϵ*-unique if the probability that there exists a new authentication tag* $t'_{tag} \neq t_{tag}$ *for which* $\mathsf{MAC.Vfy}(\mathsf{K_{MAC}}, m, t'_{tag}) = 1$ *is at most* ϵ*. For perfectly unique if* $\epsilon = 0$*.*

One-time message authentication code schemes We consider one-time message authentication code scheme OTMAC=($\mathsf{OTMAC.KGen}$, $\mathsf{OTMAC.Tag}$, $\mathsf{OTMAC.Vfy}$) to be three algorithms with associated key space $\mathcal{K}_{\mathsf{OTMAC}}$, message space $\mathcal{M}_{\mathsf{OTMAC}}$ and tag space $\mathcal{T}_{\mathsf{OTMAC}}$, where those algorithms have the same semantics as the regular MAC=($\mathsf{MAC.KGen}$, $\mathsf{MAC.Tag}$, $\mathsf{MAC.Vfy}$).

Definition 9. *We say that a one-time message authentication scheme* OTMAC *is* $(t, \epsilon_{\mathsf{OTMAC}})$*-secure, if* OTMAC *is a* $(1, t, \epsilon_{\mathsf{OTMAC}})$*-secure message authentication code scheme in the sense of Definition 7.*

4 Security Model

In this section we present a formal security model for a two-party PKI-based authenticated key-exchange (AKE) protocol. We follow the important line of research that was initiated by Bellare and Rogaway [2], and later modified and extended in [6,13,15]. In these models the adversary is provided with an execution environment, which emulates the real-world capabilities of an active adversary.

Execution Environment. Let $\mathcal{K} \in \{0,1\}^\kappa$ be the key space of session keys, and $\{\mathcal{PK}, \mathcal{SK}\} \in \{0,1\}^\kappa$ be key spaces of long-term public/private keys respectively. Fix a set of honest parties $\{P_1, \ldots, P_\ell\} \in \{0,1\}^\kappa$ for $\ell \in \mathbb{N}$, where each honest party $P_i \in \{P_1, \ldots, P_\ell\}$ is a potential protocol participant and has a pair of long-term public/private key $(pk_i, sk_i) \in (\mathcal{PK}, \mathcal{SK})$ that corresponds to its identity i. In order to formalize several sequential and parallel executions of the protocol, each party P_i is characterized by a polynomial number of oracles $\{\pi_i^s\}$ where $s \in [d]$ is an index for a range such that $d \in \mathbb{N}$. An oracle π_i^s represents a process in which the party P_i executes the s-th protocol instance with access to the long-term key pair (pk_i, sk_i) of party P_i and to all public keys of the other parties. Moreover, we assume each oracle π_i^s maintains a list of independent internal state variables as described in Table 1.

The internal state of each oracle π_i^s is initialized as (PID_i^s, Φ_i^s, K_i^s, STA_i^s, T_i^s) = ($\emptyset$, $\emptyset$, $\emptyset$, $\emptyset$, $\emptyset$), where $\emptyset$ denotes the empty string. We assume that the session key is assigned to the variable K_i^s such that $\mathsf{K}_i^s \neq \emptyset$ iff each oracle completes the execution with an internal state $\Phi_i^s = \texttt{accept}$.

Adversary Model. An active adversary $\mathcal{A}$ is able to interact with the execution environment by issuing the following queries:

Variable	Decryption
PID_i^s	records the identity $j \in \{1, \ldots, \ell\}$ of intended communication partner P_j
Φ_i^s	denotes $\Phi_i^s \in \{\texttt{accept}, \texttt{reject}\}$
K_i^s	records the session key $K_i^s \in \mathcal{K}$
STA_i^s	records some secret states used to compute the session key K_i^s
T_i^s	records all messages sent and received in the order of appearance by oracle π_i^s

Table 1. Internal States of Oracles

- $\mathsf{Send}(\pi_i^s, m)$: $\mathcal{A}$ can use this query to send any message m of his own choice to oracle π_i^s. The oracle will respond according to the protocol specification and depending on its internal state. If m consists of a special symbol $\top$ ($m = \top$), then π_i^s will respond with the first protocol message.
- $\mathsf{Corrupt}(P_i)$: Oracle π_i^1 responds with the long-term private key sk_i of party P_i. If $\mathsf{Corrupt}(P_i)$ is the τ-th query issued by $\mathcal{A}$, then we say that P_i is τ-corrupted. For parties that are not corrupted we define $\tau := \infty$.
- $\mathsf{RegCorruptParty}(pk_c, P_c)$: This query allows $\mathcal{A}$ to register a new party P_c ($\ell < c < \mathbb{N}$), with a static public key pk_c on behalf of P_c. If the same party P_c is already registered (either via $\mathsf{RegCorruptParty}$-query or $r \in [\ell]$), a failure symbol $\perp$ is returned to $\mathcal{A}$. Otherwise, P_c is registered, the pair (P_c, pk_c) is distributed to all other parties, and a symbol of success $\triangle$ is returned. This query formalizes a malicious insider setting which can be used to model unknown key share (UKS) attacks and other chosen public key attacks [4,18,17]. We here formalize the arbitrary key registration policy via this query. Parties established by this query are called corrupted or adversary-controlled.
- $\mathsf{Reveal}(\pi_i^s)$: Oracle π_i^s responds to this query with the contents of variable K_i^s to $\mathcal{A}$. This query models the attacks that loss of a session key should not be damaging to other sessions.[4]
- $\mathsf{RevealState}(\pi_i^s)$: Oracle π_i^s responds with the contents of the secret state stored in variable STA_i^s.
- $\mathsf{Test}(\pi_i^s)$: This query may only be asked once throughout the game. Oracle π_i^s handles this query as follows: if the oracle has state $\Phi_i^s \neq \texttt{accept}$, then it returns some failure symbol $\perp$. Otherwise it flips a fair coin b, samples a random element $k_0 \xleftarrow{\$} \mathcal{K}$, sets $k_1 = \mathsf{K}_i^s$ to the 'real' session key, and returns k_b.

Security Definitions. We model the partnership of two oracles via the concept of *matching conversations* which was first introduced by Bellare and Rogaway [2] and later refined in [10,14]. Let T_i^s denote the transcript of messages sent and received by oracle π_i^s. We assume that messages in a transcript T_i^s are represented as binary strings. Let $|T_i^s|$ denote the number of the messages in the transcript T_i^s. Assume there are two transcripts T_i^s and T_j^t, where $w := |T_i^s|$ and $n := |T_j^t|$. We say that T_i^s is a prefix of T_j^t if $0 < w \leq n$ and the first w messages in transcripts T_i^s and T_j^t are pairwise equivalent as binary strings.

[4] Note that we have $\mathsf{K}_i^s \neq \emptyset$ if and only if $\Phi_i^s = \texttt{accept}$.

Definition 10 (Matching Conversations). *We say that π_i^s has a matching conversation to oracle π_j^t, if*

- *π_i^s has sent the last message(s) and T_j^t is a prefix of T_i^s, or*
- *π_j^t has sent the last message(s) and T_i^s is a prefix of T_j^t.*

We say that two oracles π_i^s and π_j^t have matching conversations if π_i^s has a matching conversation to process π_j^t or vice versa.

Definition 11 (Correctness). *We say that a two-party* AKE *protocol, Σ, is* correct *if for any two oracles, π_i^s and π_j^t, that have matching conversations it holds that $\Phi_i^s = \Phi_j^t = \texttt{accept}$, $\mathsf{PID}_i^s = j$ and $\mathsf{PID}_j^t = i$ and $\mathsf{K}_i^s = \mathsf{K}_j^t$.*

Definition 12 (Security Game). *We formally consider a security experiment that is played between an adversary $\mathcal{A}$ and a challenger $\mathcal{C}$. The challenger $\mathcal{C}$ implements the collection of oracles $\{\pi_i^s : i \in [\ell], s \in [d]\}$. At the beginning of the game, long-term public/private key pairs (pk_i, sk_i) for each honest entity i are generated by $\mathcal{C}$. The adversary receives public keys $pk_1, \ldots, pk_\ell$ as input. Now the adversary may start issuing* Send, RevealState, Corrupt, RegCorruptParty *and* Reveal *queries, as well as one* Test *query at some point of the game. Finally, the adversary outputs a bit b' and terminates.*

For the security definition, we need the notion of freshness of oracles.

Definition 13 (Freshness). *Let π_i^s be an accepting oracle held by a party P_i with intended partner P_j. Meanwhile, let π_j^t be an oracle (if it exists), such that π_i^s and π_j^t have matching conversations. Then the oracle π_i^s is said to be τ_0-fresh when the adversary $\mathcal{A}$ issues its τ_0-th query and none of the following conditions holds:*

- *P_i or P_j has been established by the adversary $\mathcal{A}$ via the* RegCorruptParty *query,*
- *P_i is τ_i-corrupted with $\tau_i \leq \tau_0$ and P_j is τ_j-corrupted with $\tau_j \leq \tau_0$,*
- *$\mathcal{A}$ has either made a $\mathsf{RevealState}(\pi_i^s)$ query or a $\mathsf{RevealState}(\pi_j^t)$ query (if π_j^t exists),*
- *$\mathcal{A}$ has either made a query $\mathsf{Reveal}(\pi_i^s)$ query or a $\mathsf{Reveal}(\pi_j^t)$ query (if π_j^t exists).*

Definition 14. *We say that a two-party* AKE *protocol Σ is (t, ϵ)-secure, if for all adversaries $\mathcal{A}$ running the* AKE *security game within time t while having some negligible probability $\epsilon = \epsilon(\kappa)$, it holds that:*

1. *When $\mathcal{A}$ terminates, there exists no τ_0-fresh oracle π_i^s (except with probability ϵ), such that*
   - *π_i^s has internal states $\Omega = \texttt{accept}$ and $\Psi = j$, and*
   - *there is no unique oracle π_j^t such that π_i^s and π_j^t have matching conversations.*

   Otherwise, we say that π_i^s accepts maliciously.
2. *When $\mathcal{A}$ returns b' such that*

– $\mathcal{A}$ *has issued a* Test*-query to oracle* π_i^s*, and*
– *the oracle* π_i^s *is* τ_0*-fresh throughout the security game,*

then the probability that b' *equals the bit* b *sampled by the* Test*-query is bounded by*

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

5 Authenticated Key Exchange Compile from Signature

In this section we present an efficient generic compiler that turns passively secure KE protocols as defined to AKE protocols fulfilling the security guarantees as specified in Section 4.

5.1 Protocol Description

The compiler takes as input the following building blocks: a passively secure key exchange protocol KE and a digital signature scheme SIG = (SIG.Gen, SIG.Sign, SIG.Vfy). Each party A is assumed to possess a pair of long-term keys generated as $(sk_A, pk_A) \xleftarrow{\$} \mathsf{SIG.Gen}(1^\kappa)$. In the sequel, we will use the superscript 'A' to highlight the message recorded at party A (resp. party B).

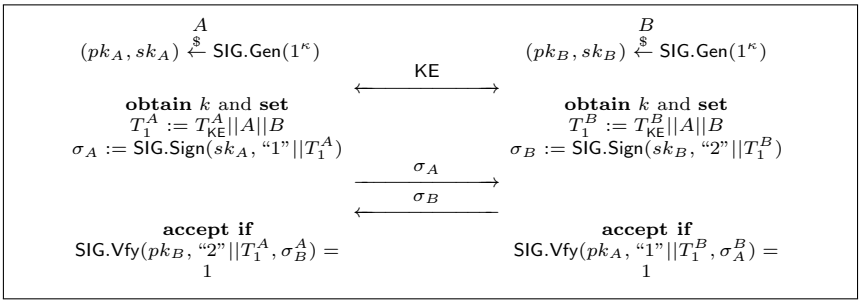

Fig. 2. AKE Protocol from Signature

Protocol Execution: The compiled protocol between two parties A and B proceeds as follows, which is also informally depicted in Figure 2.

1. First, A and B run the key exchange protocol KE. They obtain the secure key k from the key exchange phase (as the session key of AKE) and record the transcript as T_{KE}^A and T_{KE}^B, where T_{KE}^D consists of the list of all messages sent and received by party $D \in \{A, B\}$.
2. A sets $T_1^A := T_{\mathsf{KE}}^A \,||\, A \,||\, B$, computes $\sigma_A := \mathsf{SIG.Sign}(sk_A, \text{“1”} \,||\, T_1^A)$ and sends σ_A to B. Meanwhile, B sets $T_1^B := T_{\mathsf{KE}}^B \,||\, A \,||\, B$, computes $\sigma_B := \mathsf{SIG.Sign}(sk_B, \text{“2”} \,||\, T_1^A)$ and sends σ_B to A.
3. Upon receiving signature on each side, A accepts if and only if $\mathsf{SIG.Vfy}(pk_B, \text{“2”} \,||\, T_1^A, \sigma_B^A)$=1. B accepts if and only if $\mathsf{SIG.Vfy}(pk_A, \text{“1”} \,||\, T_1^B, \sigma_A^B)$=1.

Session States: In the following we assume that the ephemeral secret vector *esk* used in each KE protocol instance will be stored in the variable STA. This is consistent with the usual distinction (made formal by providing different types of queries) between the compromise of long-term keys via the $\mathsf{Corrupt}$ query on the one hand and the session keys via the Reveal query on the other hand.

5.2 Security Analysis

Theorem 1. *Assume that the* KE *protocol without long-term key is* $(t, \epsilon_{\mathsf{KE}})$*-passively secure (with respect to Definition 1), and the signature scheme* SIG *is unique and* $(q_{sig}, t, \epsilon_{\mathsf{SIG}})$*-secure (with respect to Definition 3.3), then the above protocol is a* (t', ϵ)*-secure* AKE *protocol in the sense of Definition 14 with* $t' \approx t$*, and* $q_{sig} \geq d$*, and it holds that*

$$\epsilon \leq 2\ell \cdot \epsilon_{\mathsf{SIG}} + d\ell(d\ell + 2) \cdot \epsilon_{\mathsf{KE}}.$$

We prove Theorem 1 in two stages. First, we show that the AKE protocol is a secure authentication protocol except for probability ϵ_{auth}, that is, the protocol fulfills security property 1.) of the AKE definition. In the next step, we show that the session key of the AKE protocol is secure except for probability ϵ_{ind} in the sense of the Property 2.) of the AKE definition. We require that $q_{sig} \geq d$, since we may need to correctly simulate all signatures for the d oracles of the party that is attacked in the security game. Then we have the overall probability ϵ that an adversary breaking the protocol is at most $\epsilon \leq \epsilon_{auth} + \epsilon_{ind}$. To prove the following lemmas, we proceed in games as in [19,3].

Lemma 2. *For any adversary* $\mathcal{A}$ *running in time* $t' \approx t$*, the probability that there exists an oracle* π_i^s *that accepts maliciously in the sense of Definition 14 is at most*

$$\epsilon_{auth} \leq d\ell \cdot \epsilon_{\mathsf{KE}} + \ell \cdot \epsilon_{\mathsf{SIG}},$$

where all quantities are as the same as stated in the Theorem 1.

Proof. Let $\mathsf{break}_{\delta}^{(1)}$ be the event that there exists a τ and a τ-fresh oracle $\pi_i^{s^*}$ that has internal state $\Phi = \mathtt{accept}$ and $\mathsf{PID}_i^s = j$, but there is no unique oracle π_j^t such that π_i^s and π_j^t have matching conversations, in Game δ.

Game 0. This is the original security game. We have that

$$\Pr[\mathsf{break}_0^{(1)}] = \epsilon_{auth}.$$

Game 1. In this game, the challenger proceeds exactly like the challenger in Game 0, except that we add an abortion rule. The challenger raises event $\mathsf{abort}_{\mathsf{eph}}$ and aborts, if during the simulation an ephemeral key epk_i^s is computed by an oracle π_i^s but it has been sampled by another oracle π_u^w before with the same type of ephemeral key generator. From the result of Lemma 1, we know that the collision probability among ephemeral keys is related to the polynomial number

of execution of $\mathsf{KE.EKGen}$ and the probability ϵ_{KE}. Since there are $d\ell$ oracles at all each of which would execute one general $\mathsf{KE.EKGen}$, therefore the event $\mathsf{abort}_{\mathsf{eph}}$ occurs with probability $\Pr[\mathsf{abort}_{\mathsf{eph}}] \leq d\ell \cdot \epsilon_{\mathsf{KE}}$. We have that

$$\Pr[\mathsf{break}_0^{(1)}] \leq \Pr[\mathsf{break}_1^{(1)}] + d\ell \cdot \epsilon_{\mathsf{KE}}.$$

GAME 2. This game proceeds exactly as before, but the challenger raises event $\mathsf{abort}_{\mathsf{sig}}$ and aborts if the following condition holds:

- there exists a τ-fresh oracle π_i^s that has $\mathsf{PID}_i^s = j$ and $T_1^{i,s} = T_{\mathsf{KE}}^{i,s}||\mathsf{ID}_i||\mathsf{ID}_j$ and $\Phi_i^s = \mathtt{accept}$,
- there is no unique oracle π_j^t which has the transcript $T_1^{j,t} = T_{\mathsf{KE}}^{j,t}||\mathsf{ID}_i||\mathsf{ID}_j$ such that $T_1^{i,s} = T_1^{j,t}$,
- the signature received by π_i^s that is computed over "1"$||T_1^{i,s}$ (resp. "2"$||T_1^{i,s}$) and verified correctly under the long-term public key pk_{ID_j}.

We have

$$\Pr[\mathsf{break}_1^{(1)}] \leq \Pr[\mathsf{break}_2^{(1)}] + \Pr[\mathsf{abort}_{\mathsf{sig}}].$$

If the event $\mathsf{abort}_{\mathsf{sig}}$ happens with non-negligible probability, then we could construct a signature forger $\mathcal{F}$ as follows. The forger $\mathcal{F}$ receives a public key pk^* as input, and runs the adversary $\mathcal{A}$ as a subroutine simulating the challenger for $\mathcal{A}$. It first guesses an index $\theta \xleftarrow{\$} [\ell]$ pointing to the public key for which the adversary is able to forge, and sets $pk_{\mathsf{ID}_\theta} = pk^*$. Next $\mathcal{F}$ generates all other long-term public/secret keys honestly like the challenger in the previous game. Then the $\mathcal{F}$ proceeds as the challenger in Game 2, except that it uses its chosen-message oracle to generate a signature under pk_{ID_θ} for the oracles of party ID_θ.

When $\mathsf{abort}_{\mathsf{sig}}$ is raised, this means the adversary $\mathcal{A}$ has forged a signature on behalf of an uncorrupted party ID_j. If the simulator guessed the party ID_j the adversary attacked (such that $\theta = j$) correctly, which happens with probability $1/\ell$, then the $\mathcal{F}$ can use the signature received by π_i^s to break the EUF-CMA security of the underlying signature scheme with success probability ϵ_{SIG}. The event $\mathsf{abort}_{\mathsf{sig}}$ happens with the probability $\frac{\Pr[\mathsf{abort}_{\mathsf{sig}}]}{\ell} \leq \epsilon_{\mathsf{SIG}}$. Therefore we have

$$\Pr[\mathsf{break}_1^{(1)}] \leq \Pr[\mathsf{break}_2^{(1)}] + \ell \cdot \epsilon_{\mathsf{SIG}}.$$

Note that the $\mathsf{RegCorruptParty}$ query does not affect security, since all registered identities should be distinct to the identities of honest parties. So in Game 2 each accepting oracle π_i^s has a *unique* 'partner' oracle π_j^t sharing the same transcript T_1. With respect to other queries, they will be simulated honestly as in the previous game without any modification since those values are not used for authentication. Therefore we have $\Pr[\mathsf{break}_2^{(1)}] = 0$. Sum up probabilities from Game 0 to Game 2, we proved Lemma 2.

Lemma 3. *For any adversary $\mathcal{A}$ running in time $t' \approx t$, the probability that $\mathcal{A}$ correctly answers the* Test*-query is at most $1/2 + \epsilon_{ind}$ with*

$$\epsilon_{ind} \leq \ell \cdot \epsilon_{\mathsf{SIG}} + d\ell(d\ell + 1) \cdot \epsilon_{\mathsf{KE}},$$

where all quantities are as the same as stated in the Theorem 2.

Proof. Let $\mathsf{break}_{\delta}^{(2)}$ denote the event that the $\mathcal{A}$ correctly guesses the bit b sampled by the Test-query in Game δ, and $\mathsf{Test}(\pi_i^{s^*})$ is the τ-th query of $\mathcal{A}$, and $\pi_i^{s^*}$ is a τ-fresh oracle that is ∞-revealed throughout the security game. Let $\mathsf{Adv}_{\delta} := \Pr[\mathsf{break}_{\delta}^{(2)}] - 1/2$ denote the advantage of $\mathcal{A}$ in Game δ. Consider the following sequence of games.

Game 0. This is the original security game. Thus we have that

$$\Pr[\mathsf{break}_0^{(2)}] = \epsilon_{ind} + 1/2 = \mathsf{Adv}_0 + 1/2.$$

Game 1. The challenger $\mathcal{C}$ in this game proceeds as before, which aborts if the test oracle accepts without unique partner oracle. Thus we have

$$\mathsf{Adv}_0 \leq \mathsf{Adv}_1 + \epsilon_{\mathsf{auth}} \leq \mathsf{Adv}_1 + d\ell \cdot \epsilon_{\mathsf{KE}} + \ell \cdot \epsilon_{\mathsf{SIG}},$$

where ϵ_{auth} is an upper bound on the probability that there exists an oracle that accepts without unique partner oracle in the sense of Definition 14 (cf. Lemma 2). We have now excluded active adversaries between test oracle and its partner oracle.

Game 2. This game proceeds exactly as the previous game but the challenger aborts if it fails to guess the test oracle $\pi_i^{s^*}$ and its partner oracle $\pi_j^{t^*}$ such that $\pi_i^{s^*}$ and $\pi_j^{t^*}$ have matching conversations. Since there are ℓ honest parties and d oracles for each party, the probability that the adversary guesses correctly is at least $1/(d\ell)^2$. Thus we have that

$$\mathsf{Adv}_1 \leq (d\ell)^2 \cdot \mathsf{Adv}_2.$$

Game 3. Finally, we replace the key k^* of the test oracle $\pi_i^{s^*}$ and its partner oracle $\pi_j^{t^*}$ with the same random value $\widetilde{k^*}$. If there exists an adversary $\mathcal{A}$ who can distinguish this game from the previous game, then we use it to construct an algorithm $\mathcal{B}$ to break the passive security of key exchange protocol KE as follows. Assuming that the adversary $\mathcal{B}$ interacts with the challenger $\mathcal{C}_{\mathsf{KE}}$ via $\mathsf{Execute}$ query which simulates the passive security game as in the security definition of key exchange. More specifically, $\mathcal{B}$ simulates the challenger in this game for $\mathcal{A}$ which is illustrated as follows:

- At the beginning of the game, $\mathcal{B}$ implements the collection of oracles $\{\pi_i^s : i \in [\ell], s \in [d]\}$. All long-term public/private key pairs $(pk_{\mathsf{ID}_i}, sk_{\mathsf{ID}_i})$ for each honest entity i are generated honestly. The adversary $\mathcal{A}$ receives the public keys $pk_{\mathsf{ID}_1}, \ldots, pk_{\mathsf{ID}_\ell}$ as input.
- Meanwhile, $\mathcal{B}$ generates a random ephemeral key vector epk_i^s and corresponding ephemeral secret vector esk_i^s for each oracle π_i^s as described in the protocol specification and answers all oracle queries honestly except for the test oracle and its partner oracle.
- As for the correctly guessed test oracle $\pi_i^{s^*}$ and its partner oracle $\pi_j^{t^*}$, $\mathcal{B}$ queries $\mathcal{C}_{\mathsf{KE}}$ for executing a test protocol instance and obtains (K_b, T) from

$\mathcal{C}_{\mathsf{KE}}$. $\mathcal{B}$ simulates the test oracles using the transcript T and giving $\mathcal{A}$ K_b in return. $\mathcal{A}$ may keep asking oracle queries. Meanwhile, if the ephemeral keys of the test oracle and its partner oracle have been sampled by $\mathcal{B}$ before it can trivially win the KE game and halts.
- Eventually, $\mathcal{B}$ returns the bit b' obtained from $\mathcal{A}$ to $\mathcal{C}_{\mathsf{KE}}$.

The simulation of $\mathcal{B}$ is perfect since $\mathcal{B}$ can always correctly answer all queries from $\mathcal{A}$. In particular for those $\mathsf{RevealState}$ queries, $\mathcal{B}$ can answer them using those ephemeral secret keys esk_i^s chosen by himself. If $\mathcal{A}$ is able to correctly answer the bit b of Test-query with non-negligible probability, so does the adversary $\mathcal{B}$ (which breaks the passive security of the KE protocol). Exploiting the security of key exchange protocol, we obtain that

$$\mathsf{Adv}_2 \leq \mathsf{Adv}_3 + \epsilon_{\mathsf{KE}}.$$

In this game, the response to the Test query always consists of a uniformly random key, which is independent to the bit b flipped in the Test query. Thus we have $\mathsf{Adv}_3 = 0$. Lemma 3 is proved by putting together of probabilities from Game 0 to Game 3.

6 Authenticated Key Exchange Compile from Public Key Encryption

In this section we present a public key encryption based AKE compiler that turns passively secure key exchange protocols as defined to AKE protocols fulfilling the security guarantees as specified in Section 4.

6.1 Protocol Description

The compiler takes the following building blocks as input: a passively secure key exchange protocol KE, a public encryption scheme PKE, a collision resistant hash function CRHF and a one-time message authentication scheme OTMAC.[5]

Protocol Execution: The compiled protocol between two parties A and B proceeds as follows, which is also depicted in Figure 3.

1. First, A and B run the key exchange protocol KE, then both parties obtain the key k from the key exchange phase (as the session key of AKE protocol) and record the transcripts as T^A_{KE} and T^B_{KE}, where T^D_{KE} consists of the list of all messages sent and received by party $D \in \{A, B\}$.

[5] During the initialization phase, the hash key is generated as $hk_{\mathsf{CRHF}} \xleftarrow{\$} \mathsf{CRHF.KG}(1^\kappa)$. Each party A is assumed to possess a pair of long-term private and public keys generated as $(sk_A, pk_A) \xleftarrow{\$} \mathsf{PKE.KGen}(1^\kappa)$.

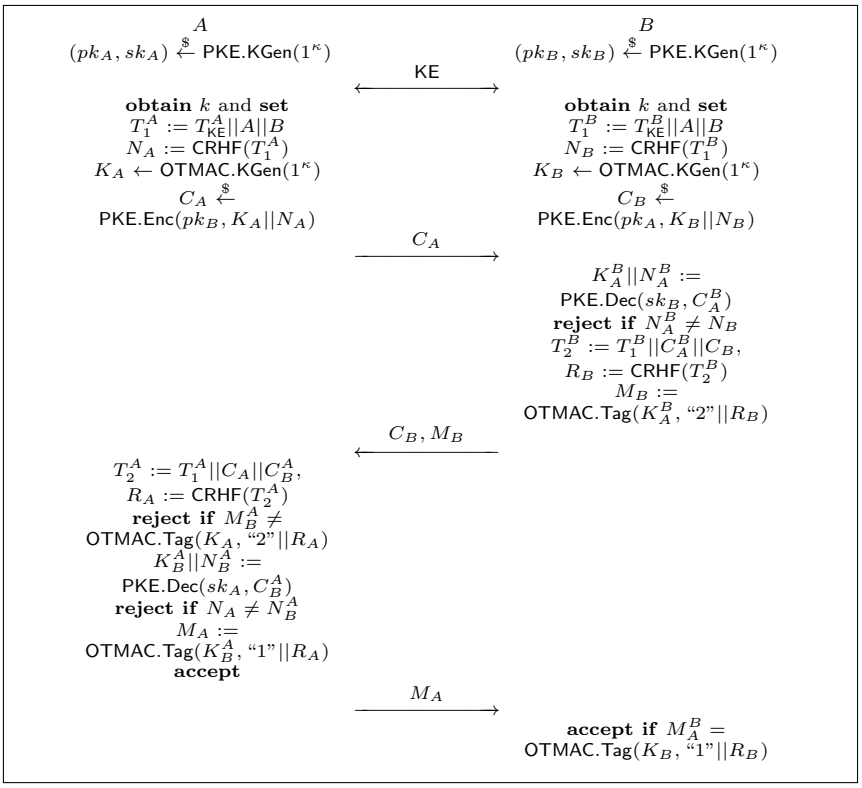

Fig. 3. AKE Compiler from PKE and OTMAC

2. A sets the transcript $T_1^A := T_{\mathsf{KE}}^A \,||\, A \,||\, B$ and computes $N_A := \mathsf{CRHF}(T_1^A)$. Then, it runs $K_A \xleftarrow{\$} \mathsf{OTMAC.KGen}(1^\kappa)$ and computes a ciphertext $C_A \xleftarrow{\$} \mathsf{PKE.Enc}(pk_B, K_A \,||\, N_A)$ under B's public key pk_B and transmits C_A to B. Meanwhile, B sets $T_1^B := T_{\mathsf{KE}}^B \,||\, A \,||\, B$ and computes $N_B := \mathsf{CRHF}(T_1^B)$. It runs $K_B \xleftarrow{\$} \mathsf{OTMAC.KGen}(1^\kappa)$ and computes $C_B \xleftarrow{\$} \mathsf{PKE.Enc}(pk_A, K_B \,||\, N_B)$ under A's public key pk_A.
3. Upon receiving the ciphertext C_A^B, B sets $T_2^B := T_1^B \,||\, C_A^B \,||\, C_B$ and computes $R_B := \mathsf{CRHF}(T_2^B)$. It decrypts C_A^B (i.e. $K_A^B \,||\, N_A^B := \mathsf{PKE.Dec}(sk_B, C_A^B)$). Then B checks whether $N_A^B = N_B$. If the check is not passed, then B rejects. Otherwise, it computes $M_B := \mathsf{OTMAC.Tag}(K_A^B, \text{“2”} \,||\, R_B)$ and transmits (M_B, C_B) to A.
4. Upon receiving messages (M_B^A, C_B^A), A sets $T_2^A := T_1^A \,||\, C_A \,||\, C_B^A$ and computes $R_A := \mathsf{CRHF}(T_2^A)$. A rejects if $M_B^A \neq \mathsf{OTMAC.Tag}(K_A, \text{“2”} \,||\, R_A)$. Then it decrypts the ciphertext C_B^A (i.e. $K_B^A \,||\, N_B^A := \mathsf{PKE.Dec}(sk_B, C_B^A)$) and checks whether $N_A = N_B^A$. If the check is not passed, then A rejects. Otherwise, A computes $M_A := \mathsf{OTMAC.Tag}(K_B^A, \text{“1”} \,||\, R_A)$, and sends M_A to B. Finally, A accepts the session.

5. Upon receiving M_A^B, B accepts if and only if $M_A^B = \mathsf{OTMAC.Tag}(K_B, \text{“1”}||R_B)$.

Session States: We assume the ephemeral secret vector *esk* used in each KE protocol instance and the random key K_A and K_B used by PKE.Enc will be stored in the variable STA.

6.2 Security Analysis

In this section, we present our results for the PKE-based compiler.

Theorem 2. *Assume that the* KE *protocol without long-term key is* $(t, \epsilon_{\mathsf{KE}})$*-secure (with respect to Definition 1), the public key encryption scheme* PKE *is* $(q_{pke}, t, \epsilon_{\mathsf{PKE}})$*-secure (IND-CCA2) (with respect to Definition 3.5), the hash function* CRHF *is* $(t, \epsilon_{\mathsf{CRHF}})$*-secure (with respect to Definition 3.4), and the one-time authentication code scheme* OTMAC *is unique and* $(t, \epsilon_{\mathsf{OTMAC}})$*-secure (with respect to Definition 3.6). Then the above protocol is a* (t', ϵ)*-secure* AKE *protocol in the sense of Definition 14 with* $t' \approx t$ *and* $q_{pke} \geq d$ *and holds that*

$$\epsilon \leq 2\epsilon_{\mathsf{CRHF}} + d\ell \cdot (2\ell \cdot \epsilon_{\mathsf{PKE}} + 2\epsilon_{\mathsf{OTMAC}} + 2\epsilon_{\mathsf{KE}}) + (d\ell)^2 \cdot \epsilon_{\mathsf{KE}}.$$

We prove the theorem 2 with two lemmas, similar to the proof of Theorem 1. For space reasons we only provide a sketch of the proof.

Lemma 4. *For any adversary* $\mathcal{A}$ *running in time* $t' \approx t$*, the probability that there exists an oracle* π_i^s *that accepts maliciously in the sense of Definition 14 is at most*

$$\epsilon_{auth} \leq \epsilon_{\mathsf{CRHF}} + d\ell \cdot (\epsilon_{\mathsf{KE}} + \ell \cdot \epsilon_{\mathsf{PKE}} + \epsilon_{\mathsf{OTMAC}}),$$

where all quantities are as the same as stated in the Theorem 2.

Proof. The proof proceeds in a *sequence of games*.

Game 0. This is the original security game. Thus we have that

$$\Pr[\mathsf{break}_0^{(1)}] = \epsilon_{auth}.$$

Game 1. In this game, the challenger proceeds exactly like the challenger in Game 0, except that we add an abortion rule. The challenger raises event $\mathsf{abort}_{\mathsf{eph}}$ and aborts, if two oracles output the same ephemeral keys. Thus, the event $\mathsf{abort}_{\mathsf{eph}}$ occurs with probability $\Pr[\mathsf{abort}_{\mathsf{eph}}] \leq d\ell\epsilon_{\mathsf{KE}}$ in terms of Lemma 1. We have that

$$\Pr[\mathsf{break}_0^{(1)}] \leq \Pr[\mathsf{break}_1^{(1)}] + d\ell \cdot \epsilon_{\mathsf{KE}}.$$

Game 2. This game proceeds as the previous game, but we add an abort condition $\mathsf{abort}_{\mathsf{cr}}$ that the challenger aborts if there are two distinct inputs to the CRHF that map to the same output value. Obviously the $\Pr[\mathsf{abort}_{\mathsf{cr}}] \leq \epsilon_{\mathsf{CRHF}}$ in each case, according to the security property of hash functions. Thus we have

$$\Pr[\mathsf{break}_1^{(1)}] \leq \Pr[\mathsf{break}_2^{(1)}] + \epsilon_{\mathsf{CRHF}}.$$

GAME 3. This game proceeds exactly as before, but the challenger aborts if it fails to guess the first fresh oracle $\pi_i^{s^*}$ which accepts without matching conversation, where $i \in [\ell]$ and $s^* \in [d]$. Thus we have that

$$\Pr[\mathsf{break}_2^{(1)}] \leq d\ell \cdot \Pr[\mathsf{break}_3^{(1)}].$$

GAME 4. This game proceeds exactly as before, but we replace the key $K_i^{s^*}$ that is used to verify the hash value $M_j^{s^*}$ with a random key $\widetilde{K}_i^{s^*}$. We apply the same modification to the oracle $\pi_j^{t^*}$ which shares the same transcript T_1 with oracle $\pi_i^{s^*}$, and use $\widetilde{K}_i^{s^*}$ to compute the confirmation hash value for such oracle $\pi_j^{t^*}$. Since ID_j is uncorrupted and there is no collision among the hashed transcripts (due to the previous games), any $\mathcal{A}$ that distinguishes this game from the previous game can be used to break the security of the PKE scheme. Due to the security of the PKE scheme, the advantage of $\mathcal{A}$ in distinguishing between this game and the previous game is bound by ϵ_{PKE}. Thus we have that

$$\Pr[\mathsf{break}_3^{(1)}] \leq \Pr[\mathsf{break}_4^{(1)}] + \ell \cdot \epsilon_{\mathsf{PKE}}.$$

GAME 5. This game proceeds exactly like the previous game except that the challenger aborts if the fresh oracle $\pi_i^{s^*}$ accepts a confirmation message $M_j^{s^*}$ but it has not been sent by any oracle of its intended partner ID_j. In this game, the fresh oracle $\pi_i^{s^*}$ accepts if and only if it has a unique partner oracle. Thus no adversary can break the authentication property, and we have $\Pr[\mathsf{break}_5^{(1)}] = 0$. Please note that the key $\widetilde{K}_i^{s^*}$ of the OTMAC in computation of $\pi_i^{s^*}$ is a random value which is independent of the ciphertext $C_i^{s^*}$ and only would be used once at most to authenticate the transcript T_2^{i,s^*}. Applying the security of OTMAC we have that

$$\Pr[\mathsf{break}_4^{(1)}] \leq \Pr[\mathsf{break}_5^{(1)}] + \epsilon_{\mathsf{OTMAC}}.$$

We could obtain the overall advantage of adversary showed in Lemma 4 via collecting the probabilities from Game 0 to Game 5.

Lemma 5. *For any adversary $\mathcal{A}$ running in time $t' \approx t$, the probability that $\mathcal{A}$ correctly answers the* Test*-query is at most $1/2 + \epsilon_{ind}$ with*

$$\epsilon_{ind} \leq \epsilon_{\mathsf{CRHF}} + d\ell \cdot (\epsilon_{\mathsf{KE}} + \ell \cdot \epsilon_{\mathsf{PKE}} + \epsilon_{\mathsf{OTMAC}}) + (d\ell)^2 \cdot \epsilon_{\mathsf{KE}},$$

where all quantities are as the same as stated in the Theorem 2.

Proof. The proof proceeds in a *sequence of games*.

GAME 0. This is the original security game. Thus we have that

$$\Pr[\mathsf{break}_0^{(2)}] = \epsilon_{ind} + 1/2 = \mathsf{Adv}_0 + 1/2.$$

GAME 1. The challenger in this game proceeds as before, but it aborts if the test oracle accepts without unique partner oracle. Applying the security of authentication property of this protocol, we thus have

$$\mathsf{Adv}_0 \leq \mathsf{Adv}_1 + \epsilon_{\mathsf{auth}} \leq \mathsf{Adv}_1 + \epsilon_{\mathsf{CRHF}} + d\ell \cdot (\epsilon_{\mathsf{KE}} + \ell \cdot \epsilon_{\mathsf{PKE}} + \epsilon_{\mathsf{OTMAC}}).$$

GAME 2. This game is similar to the previous game. However, the challenger $\mathcal{C}$ now guesses the partner oracle $\pi_j^{t^*}$ that stays fresh and participates with $\pi_i^{s^*}$ in the test session. $\mathcal{C}$ aborts if its guess is not correct. Thus we have

$$\mathsf{Adv}_1 \leq (d\ell)^2 \mathsf{Adv}_2.$$

GAME 3. Finally, we replace the session key k^* of the test oracle $\pi_i^{s^*}$ and its partner oracle $\pi_j^{t^*}$ with the random value $\widetilde{k^*}$. Applying the security of key exchange protocol, we obtain that

$$\mathsf{Adv}_2 \leq \mathsf{Adv}_3 + \epsilon_{\mathsf{KE}}.$$

In this game, the response to the Test query consists always of a uniformly random key, the view of the adversary $\mathcal{A}$ is statically independent of the challenge bit b sampled in the Test query. Thus we have $\mathsf{Adv}_3 = 0$. Finally, we obtained the advantage of $\mathcal{A}$ shown in Lemma 5 by putting together all probabilities from Game 0 to Game 3.

7 Conclusions

In this paper we have presented two efficient compilers for building AKE protocols from passively secure KE protocols. Our compilers are secure in a very strong security model that allows for state reveals and PKI-related attacks. At the same time they are more efficient than all previous solutions. A practical benefit of our compilers is that they do not require the key generated in the underlying KE protocol as input. This makes them also applicable to existing systems without requiring any modification. All our solutions work in the standard model, i.e. without random oracles.

Acknowledgements. We would like to thank the anonymous referees for their valuable comments and suggestions.

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 Douglas R. Stinson, editor, *CRYPTO*, volume 773 of *Lecture Notes in Computer Science*, 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 Serge Vaudenay, editor, *EUROCRYPT*, volume 4004 of *Lecture Notes in Computer Science*, pages 409–426. Springer, 2006.
4. Simon Blake-Wilson and Alfred Menezes. Unknown key-share attacks on the station-to-station (sts) protocol. In Hideki Imai and Yuliang Zheng, editors, *Public Key Cryptography*, volume 1560 of *Lecture Notes in Computer Science*, pages 154–170. Springer, 1999.
5. Mike Burmester and Yvo Desmedt. A secure and efficient conference key distribution system (extended abstract). In Alfredo De Santis, editor, *EUROCRYPT*, volume 950 of *Lecture Notes in Computer Science*, pages 275–286. Springer, 1994.
6. Ran Canetti and Hugo Krawczyk. Analysis of key-exchange protocols and their use for building secure channels. In Birgit Pfitzmann, editor, *EUROCRYPT*, volume 2045 of *Lecture Notes in Computer Science*, pages 453–474. Springer, 2001.
7. Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, 1976.
8. Tibor Jager, Florian Kohlar, Sven Schaege, and Joerg Schwenk. Generic compilers for authenticated key exchange. In Masayuki Abe, editor, *ASIACRYPT*, volume 6477 of *Lecture Notes in Computer Science*, pages 232–249. Springer, 2010.
9. Tibor Jager, Florian Kohlar, Sven Schäge, and Jörg Schwenk. Generic compilers for authenticated key exchange (full version). *IACR Cryptology ePrint Archive*, 2010:621, 2010.
10. Tibor Jager, Florian Kohlar, Sven Schäge, and Jörg Schwenk. On the security of tls-dhe in the standard model. In Reihaneh Safavi-Naini and Ran Canetti, editors, *CRYPTO*, volume 7417 of *Lecture Notes in Computer Science*, pages 273–293. Springer, 2012.
11. Jonathan Katz and Moti Yung. Scalable protocols for authenticated group key exchange. *J. Cryptology*, 20(1):85–113, 2007.
12. Neal Koblitz and Alfred Menezes. Another look at security definitions. *IACR Cryptology ePrint Archive*, 2011:343, 2011.
13. Hugo Krawczyk. Hmqv: A high-performance secure diffie-hellman protocol. In Victor Shoup, editor, *CRYPTO*, volume 3621 of *Lecture Notes in Computer Science*, pages 546–566. Springer, 2005.
14. Hugo Krawczyk, Kenneth G. Paterson, and Hoeteck Wee. On the security of the tls protocol: A systematic analysis. In *CRYPTO (1)*, pages 429–448, 2013.
15. 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*, volume 4784 of *Lecture Notes in Computer Science*, pages 1–16. Springer, 2007.
16. Alfred Menezes and Nigel P. Smart. Security of signature schemes in a multi-user setting. *Des. Codes Cryptography*, 33(3):261–274, 2004.
17. Alfred Menezes and Berkant Ustaoglu. Comparing the pre- and post-specified peer models for key agreement. *IJACT*, 1(3):236–250, 2009.
18. Tatsuaki Okamoto. Authenticated key exchange and key encapsulation in the standard model. In Kaoru Kurosawa, editor, *ASIACRYPT*, volume 4833 of *Lecture Notes in Computer Science*, pages 474–484. Springer, 2007.
19. Victor Shoup. Sequences of games: a tool for taming complexity in security proofs. Cryptology ePrint Archive, Report 2004/332, 2004. `http://eprint.iacr.org/`.